

LEUPHANA
UNIVERSITY LÜNEBURG

From Reactive Control to Holistic Planning: Evolutionary Prompt Optimization for Hybrid Deterministic-Neural Agents in Long-Horizon Environments

Author:
King Wai Siu

First Supervisor:
Prof. Dr. Ricardo Usbeck

Second Supervisor:
Prof. Dr. Burkhardt Funk

A thesis submitted in fulfillment of the requirements for the degree of Master of Science (M.Sc.) in

Management & Data Science

Bonn, 5 October 2026

Abstract

Large language models are increasingly used as agents that act over many steps. They plan poorly, however, when decisions are irreversible and their consequences appear late. It is unclear whether such an agent gains most from a better observation, a better prompt or a longer plan. This thesis studies that question for a hybrid agent in which a deterministic engine and a language model share the work.

The testbed is Magic Tower, a deterministic game decided by the order of fights, doors and items. The engine lists the legal actions with exact costs and effects, the model chooses one, and an exact solver gives each single floor's optimum. Four models play single floors and the 24-floor tower in repeated episodes and runs against a rule-based heuristic; prompt optimisation and committed plans are also tested with two further models.

With the engine's annotations, Qwen3.5-397B wins all 15 episodes on the floor used for development (ten earlier and five in a re-run), each at 2 hit points (HP) against an optimum of 13. The heuristic and the model without annotations fail. With model and prompt fixed, two annotation lines that state both sides of a trade-off raised its success from 0 of 5 episodes to 10 of 10. No model wins, from a general prompt, the floors whose decisive choice is a long order of doors and fights. On the tower, whose normal ending is on floor 21, the highest floor reached rose from 7 to 18 across three successive configurations of observation, engine and prompt; the heuristic reaches floor 8. Reflective prompt optimisation in the style of GEPA rediscovers rules that follow from the combat formula and the floor structure, rarely the limit on HP spent per fight or the scarcity of keys, and with the annotations present its gains stay within the variation between repeated evaluations. Committed plans won 1 of 70 episodes on the same floor against 14 of 20 for reactive play; on single floors they matched reactive play only where a published solver's route was in the prompt.

For agents in long-horizon tasks, strategic knowledge is better placed in the engine's observation than in the prompt, and on single floors committed plans pay off only when the order is already known. Evaluations of such agents need repeated episodes and logged engine interventions: here, unlogged overrides were part of every early win on the floor used for development, and a published solver reported a wrong optimum. The framework and its evaluation platform are open source.

Keywords: LLM Agents; Hybrid Deterministic-Neural Agents; Observation Design; Long-Horizon Planning; Evolutionary Prompt Optimisation; GEPA; Reasoning Models; Magic Tower

Contents

List of Figures vii

List of Tables viii

List of Listings x

List of Algorithms xi

Nomenclature xii

1 Introduction 1
- 1.1 Research Questions . . . 2
- 1.2 Contributions . . . 3
- 1.3 Structure of the Thesis . . . 4

2 Foundations 6
- 2.1 From Language Models to Large Language Models . . . 6
  - 2.1.1 The Transformer . . . 7
  - 2.1.2 From Transformers to LLMs . . . 7
- 2.2 Reasoning Models, Decoding and Non-Determinism . . . 9
  - 2.2.1 Reasoning Models . . . 9
  - 2.2.2 Non-Determinism . . . 10
- 2.3 Prompting and In-Context Learning . . . 10
- 2.4 LLM Agents and the Planning Horizon . . . 12
  - 2.4.1 The Planning Horizon . . . 13
- 2.5 From Weight Optimisation to Prompt Optimisation . . . 14
- 2.6 Automatic Prompt Optimisation and GEPA . . . 15
  - 2.6.1 GEPA . . . 15
- 2.7 State-Space Search and Exact Solvers . . . 17

3 Related Work 19
- 3.1 Literature Search . . . 19
- 3.2 LLM Agents in Games and Long-Horizon Environments . . . 19
- 3.3 Planning Horizon . . . 21
- 3.4 Automatic Prompt Optimisation . . . 22
- 3.5 Prompt Sensitivity and the Reliability of LLM Evaluations . . . 23
- 3.6 Grounding LLMs in Symbolic Components . . . 23

3.7 Positioning 24

4 Task and Environment **26**
4.1 Magic Tower 26
4.2 Combat 27
4.3 Strategy Principles 29
4.4 Task Definition 31
4.5 Floors 32
4.5.1 The Running Example: the Simple Floor 33
4.5.2 The Tower 35
4.5.3 Scripted Events of the Tower 36
4.6 The Exact Reference Solver 37
4.6.1 From Grid to Graph 37
4.6.2 Search 39
4.6.3 The Solver as Baseline and as Design Tool 41
4.7 Reference Agents 43

5 Method **45**
5.1 The Engine 45
5.2 The Observation 46
5.2.1 Exact Costs and Status Tags 49
5.2.2 Lookahead Tree, Lookahead Value and Effective Cost 50
5.2.3 Net Stat-Route Line 51
5.2.4 Fight-Order Line 52
5.2.5 Key Lines and Key Budget 52
5.2.6 Boss Line and HP Budget 52
5.2.7 The Bare Observation 53
5.3 From Answer to Action 53
5.4 The Two System Prompts 54
5.5 Decision Loop and Planning Modes 55
5.5.1 The Reactive Loop 55
5.5.2 Committed Planning Modes 56
5.5.3 The Tower Loop 56
5.5.4 Plan Memory 59
5.6 Engine Assists and Overrides 60
5.7 Prompt Optimisation 61
5.8 Implementation 64
5.9 Measures 65
5.9.1 Success Rate 65
5.9.2 HP Relative to the Optimum and State Value 66
5.9.3 Goal Score 66

5.9.4 Paired Comparisons 66
5.9.5 Plan Check, Tower Progress and Cost 67

6 Experiments **68**
6.1 Experimental Setup and Configurations 68
6.1.1 Models and Inference Service 68
6.1.2 Final Configuration 69
6.1.3 Episodes and Repetitions 70
6.2 Main Results 70
6.3 Ablations 72
6.4 Error Analysis 75
6.4.1 Engine Overrides 76
6.4.2 One-Sided Annotations 76
6.4.3 Repair Sequence 77
6.4.4 Reasoning at the Decisive Turn 78
6.4.5 The Complex Floor: Where the Episodes Are Lost 80
6.5 Prompt Optimisation (RQ2) 81
6.5.1 Study Design and Hypotheses 81
6.5.2 Results 82
6.5.3 Second Study: Population Versus Single Lineage 87
6.5.4 Evolving a Route Without the Validator 88
6.6 Planning Horizon (RQ3) 88
6.7 Hand-Designed Floors, Generated Floors and the Tower 93
6.7.1 Hand-Designed Floors 93
6.7.2 Generated Floors 95
6.7.3 The Tower 96

7 Discussion and Conclusion **99**
7.1 Discussion 99
7.1.1 The Observation Versus the Model 99
7.1.2 The Engine Versus the Model 100
7.1.3 Finding Versus Following an Order 101
7.1.4 Observation Design on the Tower 102
7.1.5 Prompt Optimisation Versus Annotations 102
7.1.6 Reactive Control Versus Committed Planning 104
7.1.7 Model Scale and Family 105
7.2 Rival Explanations 106
7.3 Methodological Lessons 107
7.4 Limitations 108
7.5 Answers to the Research Questions 110
7.6 Future Work: A Proposal for the Next Study 111

7.7 Conclusion . . . 113

References **114**

A Prompts and Observation **126**
A.1 The Single-Floor Prompt . . . 126
A.2 The Tower Prompt . . . 128
A.3 The Rules-Only Prompt . . . 131
A.4 The Evolved Prompt of the Paired Test . . . 131
A.5 Compiled-Knowledge Prompts for the Complex Floor . . . 133
A.6 The Complete Observation of Listing 5.1 . . . 138
A.7 The Planning Instruction . . . 141
A.8 The Reflection Template and the Validator . . . 142

B Combat Rules, Exact Solver and Floors **145**
B.1 The Damage Algorithm . . . 145
B.2 The Exact Solver . . . 146
B.2.1 Graph Construction and Merging . . . 146
B.2.2 Published and Corrected Dominance Test . . . 146
B.2.3 Per-Decision Analysis . . . 147
B.3 Generated Floors . . . 147
B.4 Hand-Designed Floors . . . 148
B.5 Maps and Monster Tables . . . 149
B.5.1 Simple Floor . . . 149
B.5.2 Complex Floor . . . 149
B.5.3 hand01 . . . 151
B.5.4 hand02 . . . 153
B.5.5 hand03 . . . 153
B.5.6 hand04 . . . 154
B.6 Where the Complex-Floor Episodes Are Lost . . . 154
B.7 Scripted Events of the Tower . . . 156

C The Evaluation Platform **159**
C.1 Structure . . . 159
C.2 Observing and Replaying Runs . . . 160
C.3 Experiments and Reproduction . . . 161
C.4 Prompt Optimisation . . . 162
C.5 Logging and Tests . . . 162

D Configuration and Result Files **164**
D.1 Environment . . . 164

D.2 Settings 164
D.3 The Repair Call 169
D.4 Names and Labels 169
D.5 Result Files 171
D.6 Results Excluded from Chapter 6 176

E Validity Incidents **180**

F Prompt Lineage **184**

G Reasoning Excerpts **186**
G.1 The Decisive Turn of the Simple Floor 186
G.2 A Positive Order Line Read as an Instruction 189
G.3 The First Decision on hand01 190

Note on AI Usage **195**

Declaration of Academic Honesty **196**

List of Figures

1.1 The hybrid agent . . . 5
2.1 Simplified decoder-only transformer, after Vaswani et al. [83] . . . 8
2.2 Planning horizons over nine decisions . . . 13
2.3 The GEPA optimisation loop as described by Agrawal et al. [1] . 16
2.4 Upper part of the simple floor (rows 0 to 5 of 11), drawn with the game's own tiles . . . 18
4.1 The simple floor, drawn with the game's own tiles; the numbers 1 to 17 mark the actions of the optimal route in the order of Table 4.4 . . . 36
4.2 The exact solver's node graph of the simple floor before merging, drawn at the map positions . . . 38
4.3 The first decision of the simple floor in the solver's search tree . . 41
5.1 The hybrid agent (lookahead depth 3 on single floors, 2 on the tower) . . . 45
5.2 A one-sided and a two-sided annotation of the same two doors at turn 9 on the simple floor (Section 6.4.4) . . . 51
5.3 The platform's log page . . . 65
6.1 Success rate on the simple floor after each step of the repair sequence of Table 6.5 . . . 78
6.2 Exact state value after each turn of every complex-floor episode of Table B.3 . . . 80
6.3 Paired test of the evolved prompt for DeepSeek-V4-Flash (no thinking) under the bare observation . . . 86
6.4 Success rate by planning mode on the simple floor and on the complex floor with the solver-route prompt . . . 93
B.1 The complex floor, drawn with the game's own tiles; positions as in Figure 4.1 . . . 150
B.2 hand01, drawn with the game's own tiles; positions as in Figure 4.1 152
C.1 Structure of the platform . . . 159
C.2 Live page . . . 160
C.3 Play page after a heuristic episode on the simple floor . . . 161
C.4 Experiments page . . . 162
C.5 GEPA page . . . 163

List of Tables

2.1 Prompting terminology as used in this thesis (adapted from [67]) . 10
3.1 Positioning of the thesis against the closest related works 24
4.1 Special abilities of monsters that occur on the floors used in this thesis 28
4.2 Strategy principles, their source and how the agent meets them . 29
4.3 Floors used in this thesis 32
4.4 The optimal route of the simple floor 34
4.5 Start HP against the optimum and the number of distinct winning sets, from the exact solver 42
5.1 Annotations in the observation 48
5.2 The parts of GEPA (Section 2.6.1) and their use in the two loops of this thesis 62
6.1 Models used in the experiments 68
6.2 Success rate k/N and final HP relative to the exact optimum (HP/opt, mean over successful episodes, in parentheses) per floor 70
6.3 Ablations and temperature repetitions; k/N successes 73
6.4 Map-only condition on the simple floor ($N = 2$ per model) 74
6.5 Cumulative repair sequence on the simple floor; k/N successes . . 77
6.6 Held-out generated floors (13, one episode each): kills and mean goal score per prompt 82
6.7 Origin of the prompt rules of the hand-written prompts and how often prompt evolution rediscovered them 84
6.8 Second study: training score of the best candidate and held-out result (13 floors × 3, DeepSeek-V4-Flash with thinking, bare observation) 87
6.9 Success k/N and model calls per episode (in parentheses) by planning mode 89
6.10 Plan check 90
6.11 Hand-designed floors 94
6.12 Generated floors 96
6.13 Tower runs from MT0 with three tower observations 96
B.1 Published and corrected solver 147
B.2 Design mechanisms of the hand-designed floors 148
B.3 The decision after which the boss of the complex floor can no longer be defeated, per configuration 155
B.4 Scripted events of the tower, from the floor files of the game data 156
D.1 Model presets 164
D.2 Settings 165

D.3 Names used in this thesis and the corresponding internal labels . . 169
D.4 Result files (in `results/reliability/` unless stated) 171
D.5 Tower observations (Section 5.5.3, Table 6.13) and their result-file labels 174
D.6 Sources of computed facts 175
D.7 Excluded results (prefixes in `results/reliability/`) 177
E.1 Validity incidents in the order in which they were found 180
F.1 System prompts 184

List of Listings

2.1 Zero-shot, few-shot and chain-of-thought prompts for the same question . . . 11
5.1 The observation at the first decision on the simple floor (final configuration) . . . 48
5.2 The plan-memory block at the start of a tower request . . . 60
6.1 Reasoning of Qwen3.5-397B at turn 9 of the simple floor, before and after the net stat-route and fight-order lines were added . . . 79
6.2 Qwen3.6-35B reading a positive order line as an instruction . . . 79
A.1 The single-floor system prompt . . . 127
A.2 The tower system prompt . . . 129
A.3 The rules-only prompt . . . 131
A.4 The evolved prompt of the paired test of Section 6.5.2 . . . 132
A.5 The solver-route prompt (with the survival rule) . . . 134
A.6 The exact-route prompt . . . 137
A.7 The static floor map and the complete observation at the first decision on the simple floor (final configuration) . . . 138
A.8 The planning instruction appended to the user message (long plan, max-steps wording) . . . 142
A.9 The reflection template (single floor, full observation, no example rules) . . . 143
G.1 Qwen3.5-397B at turn 9, row 2 (failure, 3,896 characters; run log `20260924_092937_769885_simple_llm.json`) . . . 186
G.2 Qwen3.5-397B at turn 9, row 4 (win, 1,329 characters, complete; run log `20260924_170639_455104_simple_llm.json`) . . . 189
G.3 Qwen3.6-35B at turn 4, row 8 (failure, 6,459 characters, end of the reasoning; run log `20260927_023900_034601_simple_llm.json`) . . . 190
G.4 The observation at the first decision on hand01 (final configuration) 191
G.5 DeepSeek-V4-Flash at the first decision on hand01 (1,368 characters, complete; run log `20260929_073109_306644_hand01.txt_llm.json`) . . . 193
G.6 Qwen3.5-397B at the first decision on hand01 (12,168 characters, end of the reasoning; run log `20260929_185121_900081_hand01.txt_llm.json`) . . . 194

List of Algorithms

4.1 Exact search over decisions with dominance (corrected version) . . 40
5.1 The decision loop 57
5.2 Reflective prompt optimisation with a validator (single lineage, elitist) 62
B.1 Damage of one fight 145

Nomenclature

Abbreviations

AI	Artificial Intelligence
APE	Automatic Prompt Engineer
API	Application Programming Interface
APO	Automatic Prompt Optimisation
ASCII	American Standard Code for Information Interchange
ATK	Attack (hero or monster attribute)
BFS	Breadth-First Search
CoT	Chain of Thought
DEF	Defence (hero or monster attribute)
DSPy	Declarative Self-improving Python (prompt-programming framework)
EXP	Experience points
GEPA	Genetic-Pareto prompt optimisation
GRPO	Group Relative Policy Optimization
GWDG	Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen
HP	Hit points
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
LLM	Large Language Model
MDEF	Magic defence (hero attribute subtracted from every fight's damage)
MoE	Mixture of Experts
MT0–MT23	Floors 0 to 23 of the 24-floor tower
OPRO	Optimization by PROmpting
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RQ	Research Question
SAIA	Scalable AI Accelerator (inference service of the GWDG)
STRIPS	Stanford Research Institute Problem Solver

Symbols

Game, engine and solver

a_h, d_h	the hero's ATK and DEF
h_m, a_m, d_m	a monster's HP, ATK and DEF
T	number of hero strikes needed to defeat a monster
d	a monster's damage per strike after the hero's DEF (Table 4.1)
D	HP cost of a fight
(S, s_0, A, δ, G)	floor task: set of states, initial state, legal actions $A(s)$, transition, goal (Definition 4.1)
$\mathrm{HP}_{\mathrm{final}}$	the hero's HP at the end of an episode
$\mathrm{HP}_{\mathrm{opt}}$	exact optimum of a floor: the largest final HP of any winning action sequence
L	net lookahead value of an action
v_i	HP-equivalent value of an unlocked item i
λ_j	factor for items behind j gateways (1 for $j = 0$; 0.7, 0.5 and 0.35 for $j = 1, 2, 3$)
c_g	HP cost of gateway g
C_{route}, S_{route}	HP of fighting needed to reach the stat items behind an action, and the HP these items save on the other listed fights (net stat-route line)

Measures

N	number of episodes of a configuration
k	commitment: number of actions per model call; in k/N, the number of successful episodes
$\hat{p}$	success rate k/N
z	quantile of the standard normal distribution in the Wilson interval ($z = 1.96$)
HP/opt	$\mathrm{HP}_{\mathrm{final}}$ divided by $\mathrm{HP}_{\mathrm{opt}}$ for a successful episode; reported as the mean over successful episodes
S_{goal}	goal score of one episode
C	share of a floor's monsters defeated (in S_{goal})
n, b	in the sign test: number of discordant pairs, and the number of these that favour the first prompt (in Chapter 2, a sequence length and a branching factor)

General symbols (Chapter 2)

w_t	the t-th token of a sequence of n tokens

$\Pr(\cdot)$	probability
Q, K, V	query, key and value matrices of self-attention
d_k	length of the key vectors
z_i	score of token i in the model's last layer
p_i	probability of token i after the softmax
τ	sampling temperature
θ	parameters of a model
P	prompt
s, a	state, action
$\pi(a \mid s, P)$	policy: probability of action a in state s under prompt P
G	number of outputs sampled per input in GRPO (the goal of a floor task in Definition 4.1)
r_i	reward of output i
$\hat{A}_i$	advantage of output i in GRPO: its reward standardised within its group
b	branching factor: number of choices per step of a search
ℓ	search depth: number of actions from the initial state

Models

Models are named by family, version and total size; the endpoint identifiers are listed in Table 6.1 and Appendix D.

Name	**Type and size**	**Reasoning mode**	**Used in**
Qwen3.5-397B	mixture of experts, 397 B parameters, 17 B active	thinking	main experiments; planning study (Section 6.6); reference and teacher in the first configuration of the first prompt-optimisation study

Name	Type and size	Reasoning mode	Used in
Qwen3.6-35B	mixture of experts, 35 B parameters, 3 B active	thinking	main experiments; prompt-optimisation study (Section 6.5); exact-route prompt (Section 6.6)
Qwen3.8-27B	dense, 27 B parameters	thinking	planning study (Section 6.6)
DeepSeek-V4-Flash	mixture of experts, 284 B parameters, 13 B active	thinking	main experiments; planning study (Section 6.6); teacher in the second configuration of the first prompt-optimisation study, task model and teacher in Sections 6.5.3 and 6.5.4
DeepSeek-V4-Flash (no thinking)	the same model with its reasoning mode switched off	none	prompt-optimisation study (Section 6.5)
Mistral Medium 3.5	128 B parameters, architecture not stated by the service	none	main experiments
Llama-3.1-8B	dense, 8 B parameters, instruction-tuned	none	prompt-optimisation study (Section 6.5)

B = billion.

Labels

RQ1–RQ3	research questions (Section 1.1)
HA–HC	hypotheses of the thesis, one per research question (Section 1.1)
R1–R3	rival explanations (Sections 1.1 and 7.2)
S1–S14	strategy principles (Table 4.2)
G1–G5	hypotheses of the prompt-optimisation study, stated before the runs (Section 6.5.1)
P1–P3	predictions for the evolved-route run, stated before the run (Section 6.5.4)
hand01–hand04	hand-designed floors (Table 4.3)
gen01–gen15	generated floors (Table 4.3)
First, second and final tower observation	the three tower observations compared in Table 6.13 (Section 5.5.3); the final one is used for all tower results of Table 6.2

Internal Labels

Two labels of the verbatim observation text appear in listings and appendices:

`effCost`	effective cost
`ORDER` line	fight-order line

The result file behind each reported cell is listed in Table D.4 (Appendix D).

1 Introduction

Large language models (LLMs) can act as agents: systems that observe an environment, reason about it and choose actions over many steps. Frameworks such as ReAct [101] and Reflexion [73] let a model act in text games, on websites and in programming tasks. Benchmarks measure how far such agents get [48, 57, 106]. Many of these tasks are short, and many of their environments forgive mistakes: a wrong click can be undone, a failed attempt repeated.

Other planning problems are different. When resources are consumed irreversibly, a decision at the fifth step can make the fortieth step impossible, and the quality of a decision is visible only much later. In classical planning domains, LLMs that plan on their own produce few valid plans [82]. This has led to designs that pair the model with components that compute exactly, such as classical planners or verifiers [36, 43]. Such pairings raise three questions. The first is what the exact components contribute. The second, whether optimising the prompt automatically from task outcomes improves the agent, follows because a hosted model can only be changed through its input. The third concerns the planning horizon: an agent can decide one action at a time, reacting to every new state, or commit to a longer plan with fewer model calls. It is not clear which is better when the consequences of actions unfold over many steps.

Answering these questions requires an environment in which the quality of every decision can be measured exactly. Magic Tower, a grid-based puzzle role-playing game, offers such an environment. It is fully deterministic, and every resource in it is spent irreversibly. The order in which fights, doors and items are taken decides whether a floor can be won and with how many hit points (HP) remaining. Because the game is deterministic, an exact search can compute the best possible outcome of a single floor. Every episode on such a floor, one complete play from start to end, can be compared with that optimum. Studies of LLM agents in games mostly use stochastic environments without such an optimum. Prompt optimisers have been evaluated on short tasks with cheap, repeatable scores (Chapter 3).

This thesis investigates a hybrid deterministic-neural agent for Magic Tower. A game engine enumerates the legal actions and computes the exact cost of every fight and what every action unlocks. It writes these facts into the model's input as an annotated, numbered list of actions; the model reads the list, reasons, and chooses one action by its number. On this agent, the thesis measures the contribution of the engine's annotations, of evolutionary prompt optimisation and of the planning horizon (Section 1.1). Every single-floor configuration is evaluated over repeated episodes. The success rate is reported with a confidence interval and the final HP relative to the exact optimum; tower runs, which climb the game's 24 floors, are

reported by the highest floor reached. The code and the evaluation platform are publicly available.[1]

1.1 Research Questions

The work in this thesis is organised around three research questions:

- **RQ1:** How does engine grounding, meaning enumerated actions, exact costs and lookahead annotations, influence the success of an LLM agent on long-horizon floors, relative to a rule-based heuristic and to the same model without annotations?
- **RQ2:** When does evolutionary prompt optimisation improve such an agent, relative to the hand-written single-floor prompt and a prompt that states only the rules, and how does this depend on the observation the agent receives?
- **RQ3:** How does the planning horizon, from one action per model call to a single plan for the whole floor, affect success and cost on floors of different structure?

The hypotheses use the following terms, which later chapters define in full. The *simple floor* is an 11×11 boss floor from the test maps of a published solver, on which the observation was developed and which serves as the running example; its optimal route ends with 13 HP. The *final configuration* is the setting that every single-floor episode uses unless an experiment states otherwise; it shows all annotations. The engine's *assists* collect items and keys and take fights that cost 0 HP automatically. A *mechanical override* replaces a choice that cannot be executed usefully: a door that leads nowhere, an unwinnable fight or a lethal fight. The *rule-based heuristic* is an agent that acts on the same engine output by fixed rules, without a language model. A *strategic override* replaces the model's choice by the judgement of the rule-based heuristic. The *bare observation* keeps the enumerated actions and exact costs but removes all annotations. The *rules-only prompt* states only the game rules, the goal and the answer format, and an *evolved prompt* is the best prompt that prompt optimisation produces from it.

The following hypotheses guide the analysis:

- **HA (RQ1):** With the engine's annotations, its assists and only mechanical overrides, the agent wins floors that the rule-based heuristic and the same model without annotations do not win. *Test:* success rates on the simple floor for the final configuration, the bare observation and the heuristic (Sections 6.2 and 6.3).

[1] `https://github.com/thetobysiu/mt-ai-lab`

- **HB (RQ2):** When the observation already carries the annotations, an evolved prompt does not exceed the single-floor prompt or the rules-only prompt by more than the variation between two evaluations of the same prompt. *Test:* paired comparison on thirteen held-out floors, with the noise floor measured directly (Section 6.5).
- **HC (RQ3):** Committing to several actions per call lowers the number of model calls but also the success rate when the order of actions must be derived from annotations that appear state by state. *Test:* each planning mode against the reactive mode of the same model on the same floor (Section 6.6).

Three rival explanations could account for the results instead and are examined in Section 7.2:

- **R1:** apparent effects are caused by the non-determinism of the inference service, which is tested by repeating every configuration;
- **R2:** apparent gains of prompt optimisation are the luck of single episodes, which is tested by paired evaluation on held-out floors;
- **R3:** the engine's strategic overrides, not the model, produce the wins, which is tested by comparing the model's choice with the executed action in every stored episode and by logging every override in the final configuration.

1.2 Contributions

Based on these questions, the thesis makes three main contributions.

First, it presents an engine observation layer for LLM agents and identifies a class of defects in such layers, the *one-sided annotation*. Such an annotation shows a saving without the cost of reaching it, a value without its cost (potions counted as fight value), an efficiency without its affordability, a deficit without its scope, or a count without its multiplicity. In each case, the model's decision followed the annotation correctly and was wrong. Repairing these annotations in the engine, with model and prompt unchanged, took Qwen3.5-397B on the simple floor from 0/5 to 10/10 [0.72, 1.00] wins (Table 6.5), with no strategic override in any win. With the same observation, DeepSeek-V4-Flash wins 9 of 10 [0.60, 0.98] episodes on two hand-designed floors, against 0 of 10 without the annotations.

Second, it provides a repeated-episode evaluation protocol for LLM agents and reports six threats to the validity of its own measurements that the protocol exposed (Section 7.3; eighteen dated incidents in Appendix E). Among them are unlogged strategic overrides of the engine, present in all 21 simple-floor wins recorded before overrides were logged. Replacing a single placeholder value in the observation with the corresponding number lowered the success rate from 5 of 10 episodes per

model to 0 of 20 across two models. A published reference solver reported 22 HP as the optimum of the complex floor, a second test floor, whose exact optimum is 68 HP. The protocol, the exact solver and a web platform that runs and replays every experiment are released with the thesis.

Third, it reports negative and boundary results. The hypotheses of the prompt-optimisation study, its paired confirmation and the predictions for the optimiser runs on the complex floor were stated in advance. Prompt optimisation stays within the measured noise floor when the annotations are present and yields one narrow gain when they are withheld, which passed a paired test that included the floors on which it had been selected. With a reasoning model, a single-lineage and a population-based optimisation loop both win fewer held-out episodes than the rules-only prompt (26 and 24 against 32 of 39). Floors whose decisive choice is an order of several doors and fights are not won from a general prompt. With the published solver's route in the prompt, three of four models win 4 of 5, 3 of 5 and 4 of 10 episodes on the complex floor. The fourth wins none. On the simple floor, Qwen3.5-397B and DeepSeek-V4-Flash win 14 of 20 reactive episodes, which decide one action per call. They win 1 of 70 committed-plan episodes, including plans whose length the model chose itself. On single floors, committed plans reach the same success with about a quarter to a ninth of the model calls only where the solver's route is listed in the prompt. On the tower, with the second tower observation, commit-k and re-planned long plans reach the same floor as reactive play with 58 to 72 calls instead of 86.

1.3 Structure of the Thesis

The remainder of this thesis is structured as follows. Chapter 2 introduces the foundations: language models and transformers, reasoning models and decoding, prompting, LLM agents and the planning horizon, the optimisation of weights and prompts, and state-space search. Chapter 3 reviews related work on LLM agents in games, the planning horizon, prompt optimisation, the reliability of LLM evaluations and symbolic grounding. It closes by positioning the thesis among the closest works. Chapter 4 describes Magic Tower, defines the task formally, presents the floors used in the experiments and the exact solver that provides their optimum. Chapter 5 describes the hybrid agent, its observation, its two prompts, its planning modes, the prompt optimisation loops and the measures. Chapter 6 reports the experiments. Chapter 7 discusses the results comparison by comparison, examines the rival explanations and draws methodological lessons. It then states the limitations, answers the research questions, proposes how the work should be continued and concludes. Appendices A to C contain the prompts and the observation, the combat rules, the exact solver and the floors, and the

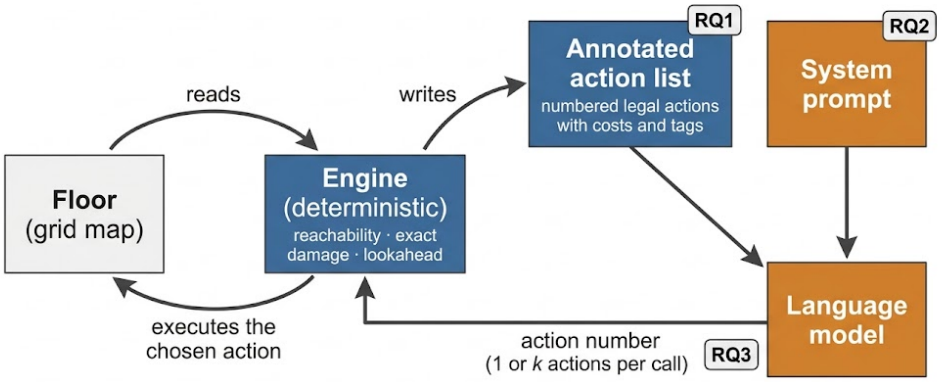


Figure 1.1: The hybrid agent. The engine reads the floor and writes the annotated action list (RQ1); the model receives it with the system prompt (RQ2), answers with an action number and commits to one or k actions per call (RQ3); the engine executes the chosen action.

evaluation platform. Appendices D to G list the configuration and result files, the validity incidents, the prompt lineage and further reasoning excerpts.

2 Foundations

This chapter introduces the concepts that the remainder of the thesis builds on, for readers without a background in machine learning. Sections 2.1 to 2.3 explain how large language models (LLMs) work and introduce prompting, the means by which the main experiments adapt a model to its task. Section 2.4 defines LLM agents and the planning horizon, the dimension studied by the third research question. Sections 2.5 and 2.6 contrast the optimisation of model weights with the optimisation of prompts and present the prompt optimiser GEPA as published. Section 2.7 introduces state-space search and exact solvers, which provide the ground truth against which the agents are measured on single floors.

2.1 From Language Models to Large Language Models

A language model assigns a probability to a sequence of text. Most modern language models do so one piece at a time: given the text so far, they predict a probability distribution over what comes next. For a sequence of units $w_1, \ldots, w_n$, the probability of the whole sequence is factorised into a product of such next-unit predictions:

$$\Pr(w_1, \ldots, w_n) = \prod_{t=1}^{n} \Pr(w_t \mid w_1, \ldots, w_{t-1}).$$

The units are called *tokens*. A *tokenizer* splits text into tokens from a fixed vocabulary of subword pieces, so that frequent words are single tokens and rare words are split into several pieces [70]. Numbers, coordinates and game terms such as "Slime Lord" therefore occupy one or more tokens, and every length limit discussed in this thesis is measured in tokens, not words.

A language model is implemented as a neural network: a function with a large number of adjustable numbers, called *parameters*, whose values are learned from data. The parameter count is the usual measure of model size; the largest model used in this thesis has 397 billion parameters. Before the network can process a token, the token is mapped to a vector of numbers, its *embedding*, which the network learns so that tokens used in similar contexts receive similar vectors.

Earlier language models counted how often short word sequences occur (n-gram models). Later models processed the text one token after another with a recurrent network that carried a summary of everything read so far [35]. Both approaches struggle with long-range dependencies, because information from early in the text must survive many processing steps to influence a later prediction.

2.1.1 The Transformer

The transformer [83] replaced step-by-step processing with *self-attention*: every token can draw information directly from every other token in the input. For each token, the network computes three vectors from its embedding: a *query* (what the token is looking for), a *key* (what the token offers) and a *value* (the information it passes on). Stacking these vectors for all tokens into matrices Q, K and V, attention is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where d_k is the length of the key vectors. The product $QK^\top$ scores how relevant every token is to every other token. The softmax turns each row of scores into weights that sum to one, and the result is a weighted mixture of the value vectors. Each token's new representation therefore depends on the whole input, weighted by relevance. A transformer runs several such attention functions in parallel (*multi-head attention*), follows them with a small feed-forward network, and stacks dozens of these layers (Figure 2.1).

The original transformer had two parts: an encoder that reads an input sentence and a decoder that writes an output sentence, as needed for machine translation. Most current LLMs, including every model of the main experiments, keep only the decoder. A *decoder-only* model reads the text so far and predicts the next token; its attention is masked so that a token can only attend to tokens before it. Text is generated by appending the predicted token and repeating.

2.1.2 From Transformers to LLMs

Three developments turned the transformer into the LLMs in use today. First, *pre-training* on very large text corpora: the model learns to predict the next token on a large part of the public web, books and code, without any task-specific labels. Brown et al. [5] showed with GPT-3, a 175-billion-parameter decoder-only model, that such a model can perform new tasks from a textual description and a few examples, without any change to its parameters. Second, *instruction tuning*: continued training on many tasks phrased as instructions makes the model follow instructions it has not seen before [90]. Training from human preference judgements makes its answers more helpful [56]. Instruction-tuned models are used as chat models: the input is a sequence of messages from a *system* (standing instructions), a *user* (the current request) and the *assistant* (the model's earlier replies). Third, scale: parameter counts grew from millions to hundreds of billions.

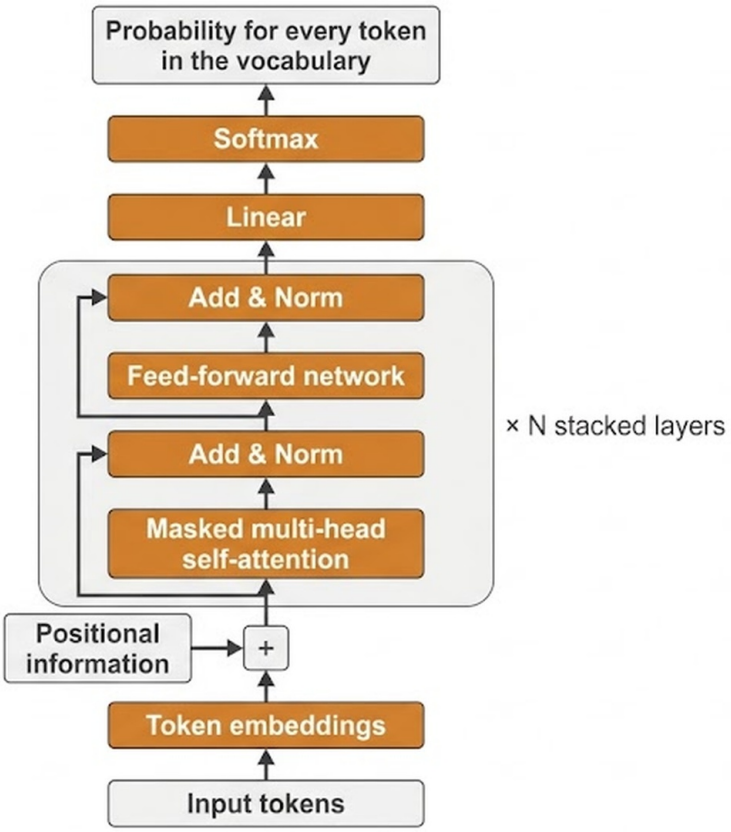


Figure 2.1: Simplified decoder-only transformer, after Vaswani et al. [83]. The input tokens are mapped to embeddings, to which information about each token's position in the text is added. Each of the stacked layers applies masked multi-head self-attention, through which every token attends to all earlier tokens, and then a feed-forward network. After each of the two, "Add & Norm" adds the input of the step to its output and normalises the result. The final linear layer and softmax produce a probability for every token in the vocabulary.

Two further properties matter for this thesis. The *context window* is the maximum number of tokens the model can attend to at once. Everything the model knows about the current game must fit into it, including the instructions, the game state and any earlier turns. Models also use information in the middle of a long context less reliably than information at its start or end [45]. A *mixture-of-experts* (MoE) model replaces the single feed-forward network of a layer by many smaller "expert" networks and a router that sends each token to only a few of them [72]. Only a fraction of the parameters is active for each token, which makes a very large model affordable to run. Model names reflect this: Qwen3.5-397B-A17B has 397 billion

parameters in total, of which about 17 billion are active per token. A model without experts, in which every parameter is used for every token, is called *dense*.

Finally, the models in this thesis are *open-weight*: their parameters are published. They are nevertheless accessed through a hosted service, the GWDG SAIA platform [15], which accepts text and returns text. The parameters of these models therefore cannot be changed in the main experiments; only the input can.

In the following section, the process by which a trained model turns an input into an answer is described in more detail.

Further reading: Jurafsky and Martin [35] give a textbook introduction to language models and transformers; Vaswani et al. [83] is the primary source for the architecture.

2.2 Reasoning Models, Decoding and Non-Determinism

At every step, the model's last layer produces one score z_i for every token i in the vocabulary. A softmax with *temperature* τ turns these scores into probabilities:

$$p_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}.$$

Decoding is the rule that picks the next token from this distribution. *Greedy decoding* always takes the most probable token and corresponds to the limit $\tau \to 0$. *Sampling* draws a token at random according to the probabilities p_i; a higher temperature flattens the distribution and makes less likely tokens more frequent. Nucleus sampling restricts the draw to the smallest set of tokens that together hold most of the probability [31]. All experiments in this thesis use temperature 0 unless stated otherwise.

2.2.1 Reasoning Models

A *reasoning model* is an LLM trained to write out an intermediate reasoning text before it gives its answer. DeepSeek-R1 [14] obtained this behaviour mainly through reinforcement learning (Section 2.5) with rewards for correct final answers. The Qwen3 family offers a "thinking" and a "non-thinking" mode in one model [98]. The reasoning is part of the generated output. It consumes tokens and time: a single decision can take from a few seconds to several minutes, depending on the model and the length of the reasoning. A *token budget*, called the *reasoning cap* in Chapter 6, caps the output. When the model reaches the cap before it has written its answer, the output is *truncated* and contains no usable answer. The

reasoning text is also not guaranteed to be a faithful account of how the answer was produced [41, 81]. In this thesis, reasoning excerpts are therefore quoted as evidence of what the model wrote, not as proof of why it acted.

2.2.2 Non-Determinism

Temperature 0 suggests that the same input always yields the same output. On hosted inference services this does not hold. Requests from many users are processed together in batches, and the batch composition changes the order in which floating-point numbers are added. Because floating-point addition is not associative, the scores z_i can differ in the last digits, and a near-tie between two tokens can flip [30]. One different token early in a long reasoning text can change everything that follows. Several studies therefore recommend that LLM evaluations report results over repeated runs instead of a single output [2, 78]. Sections 5.8 and 6.1.3 describe how this thesis follows that recommendation.

In the following section, the focus moves from how a model produces text to how its input is written.

Further reading: Holtzman et al. [31] compare decoding strategies; He and Thinking Machines Lab [30] explain the sources of non-determinism in served models.

2.3 Prompting and In-Context Learning

A *prompt* is the complete input given to an LLM to direct its output. Liu et al. [46] describe the resulting paradigm as "pre-train, prompt, and predict". Instead of training a model for each task, a single pre-trained model is adapted to a task by the way the task is presented to it. Because the terminology in this area is not settled, Table 2.1 defines the terms as they are used in this thesis, following Schulhoff et al. [67].

Table 2.1: Prompting terminology as used in this thesis (adapted from [67]). The examples are taken from the agent's prompt; Chapter 5 describes the full prompt. HP = hit points, ATK = attack, DEF = defence.

Term	Description	Example in this thesis
Prompt	The complete input to the LLM for one decision.	System prompt, current game state and numbered action list
System prompt	Standing instructions that apply to every decision of an episode.	The game rules, how to read the engine's annotations, five prompt rules (six in the tower prompt)

Term	Description	Example in this thesis
Prompt template	A function that inserts variable content into a fixed structure.	"Hero: HP=[hp] ATK=[atk] DEF=[def] … Available actions: [numbered list]"
Directive	The instruction that states the task.	"Choose the best action."
Input	The task-specific content to be processed.	Hero HP, ATK, DEF, keys; the list of reachable actions
Example	A demonstration of the expected output for a given input.	Not used (all prompts are zero-shot)
Output contract	The required form of the answer.	"FINAL_CHOICE: <number>", or a structured object with one integer field

Three foundational prompting techniques differ in the components they include (Listing 2.1). *Zero-shot* prompting presents only the directive and the input and relies on the model's pre-trained ability to generalise. *Few-shot* prompting adds input-output demonstrations. Brown et al. [5] showed that this improves many tasks without any change to the model, although the result depends on the choice and order of the examples [49]. *Chain-of-thought* (CoT) prompting adds step-by-step solutions to the demonstrations, which improves multi-step tasks such as arithmetic [91]. Kojima et al. [38] obtained a similar effect without examples by appending a cue such as "Let's think step by step". Self-consistency samples several reasoning chains and takes the majority answer [87]. Reasoning models (Section 2.2.1) produce such a chain on their own, without examples or cues.

```
(a) Zero-shot
    Hero: ATK 9, DEF 4. Green Slime: HP 50, ATK 7, DEF 3.
    How much HP does the hero lose in this fight?

(b) Few-shot
    Hero: ATK 10, DEF 5. Bat: HP 100, ATK 17, DEF 2.  ->  HP lost: 144
    Hero: ATK 9, DEF 4. Green Slime: HP 50, ATK 7, DEF 3.  ->  HP lost: ?

(c) Chain-of-thought
    Hero: ATK 10, DEF 5. Bat: HP 100, ATK 17, DEF 2.
    Hero deals 10 - 2 = 8 per hit and needs ceil(100 / 8) = 13 hits.
    The hero strikes first, so the Bat strikes 12 times for 17 - 5 = 12 each:
    ↪  144.
    Hero: ATK 9, DEF 4. Green Slime: HP 50, ATK 7, DEF 3.  ->  ?
```

Listing 2.1: Zero-shot, few-shot and chain-of-thought prompts for the same question. HP = hit points, ATK = attack, DEF = defence. The damage rule is introduced in Chapter 4; for the Green Slime the answer is 24 HP.

Two further techniques shape the prompts of this thesis. *Structured output*, or constrained decoding, restricts the model's output to a given format, for example a JSON object with one integer field. During decoding, it excludes every token that would violate the format [93]. It guarantees that the answer can be parsed. *Prompt sensitivity* is the observation that changes to a prompt that preserve its meaning can change the result substantially. Sclar et al. [69] report differences of up to 76 accuracy points from formatting changes alone, and Errica et al. [17] find similar inconsistencies across tasks. Prompt sensitivity is one reason why prompts are worth optimising (Section 2.6) and one reason why single runs are not sufficient evidence (Section 2.2.2).

In the following section, prompting is placed in the context of agents that act in an environment over many steps.

Further reading: Schulhoff et al. [67] survey prompting techniques and terminology; Liu et al. [46] give a formal account of prompt-based learning.

2.4 LLM Agents and the Planning Horizon

An *agent* is a system that repeatedly receives an *observation* from an *environment*, chooses an *action*, and thereby changes the environment's *state* [65]. The rule that maps observations to actions is the agent's *policy*. One complete play from the initial state to the end, for example until the game is won or lost, is an *episode*. An LLM with a fixed prompt can act as a policy: the observation is written into the prompt, and the answer is read as an action. Sumers et al. [79] describe such *language agents* in terms of their memory, their action space and their decision procedure.

Two early designs illustrate the range. ReAct [101] interleaves a reasoning step and an action at every step and feeds the result of each action back into the next prompt. Reflexion [73] adds a written self-assessment after a failed episode, which is carried into the next attempt. Both rely on the model to understand the environment from its textual description. When the environment has strict rules, this is a known weakness: in classical planning problems, LLMs that plan autonomously produce many invalid or unsuccessful plans [82]. Kambhampati et al. [36] therefore argue that LLMs should be paired with external verifiers that check every proposed step, and LLM+P hands the search itself to a classical planner [43]. Giving the model external programs that it can call during a task is called *tool use* [59]. In this thesis, such pairings are called *grounding*: the model's

choices are tied to facts that an external component computes exactly, either by checking its output or, as here, by writing the facts into its input.

2.4.1 The Planning Horizon

The *planning horizon* of an agent, as used in this thesis, is the number of actions it commits to per model call. A *reactive* agent chooses one action, observes the result and calls the model again; this is closed-loop control, because every decision sees the latest state. A *plan-and-execute* agent, called *commit-k* in this thesis, asks the model for a sequence of k actions and executes them without consulting the model in between. This is open-loop control over k steps [85, 97]. A *long plan* lets the model choose how many actions to commit to. The extreme case is a *single-call* plan, written in one model call for the whole episode. The trade-off is between cost and correction: a longer horizon needs fewer model calls, but an error in the plan cannot be corrected until the plan ends (Figure 2.2). The title of this thesis names the two ends of this axis: reactive control and holistic planning.

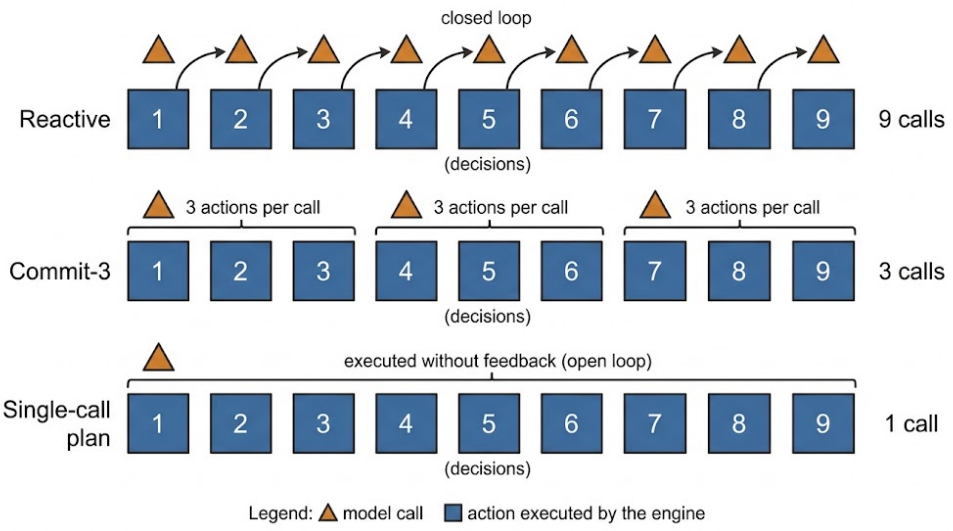


Figure 2.2: Planning horizons over nine decisions. A reactive agent calls the model at every decision (closed loop); a commit-k agent calls it every k decisions; an agent with a single-call plan calls it once and executes the rest without feedback (open loop).

Long episodes make this trade-off sharper. The consequence of a decision may appear many steps later, for example when a key spent early is missing at a door near the end. Errors also compound: an agent that succeeds at each step with probability 0.99 completes a 100-step episode without error with probability $0.99^{100} \approx 0.37$.

In the following section, the question changes from how an agent acts to how its behaviour can be improved.

Further reading: Russell and Norvig (2021, Ch. 2) define agents and environments; Sumers et al. [79] give a framework for language agents.

2.5 From Weight Optimisation to Prompt Optimisation

A model's behaviour can be changed in two places: in its parameters or in its input. *Fine-tuning* changes the parameters by continuing training on task-specific data, typically pairs of inputs and desired outputs. It requires access to the parameters and substantial computing hardware.

Reinforcement learning (RL) is used when desired outputs are not available but outcomes can be scored. The agent acts, receives a scalar *reward*, and its policy is changed so that high-reward actions become more likely [80]. For LLMs, the policy is the model itself, and the change is a gradient step on its parameters. Reinforcement learning from human feedback (RLHF) trains a reward model on human preference judgements and then optimises the LLM against it with Proximal Policy Optimization [10, 56, 68].

Group Relative Policy Optimization (GRPO [71]) simplifies this procedure. For each input it samples a group of G outputs, scores each with a reward r_i, and uses each output's standing within its group as the learning signal:

$$\hat{A}_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}.$$

Outputs better than their group are made more likely, worse ones less likely, without a separate network that estimates expected rewards. GRPO was used to train DeepSeek-R1 [14]. Like other policy-gradient methods, it learns only from the scalar score: the optimiser does not see why an output failed.

When a model is accessed through a service, as in the main experiments of this thesis, its parameters θ cannot be changed. The only remaining parameter is the prompt P, and the policy becomes $\pi(a \mid s, P)$ with fixed weights. The outer loop stays the same: run episodes, observe outcomes, update the parameter, repeat. The update itself differs. A prompt is text, and text can be revised by a model that has read what went wrong. Ramnath et al. [62] accordingly classify prompt optimisers first by the feedback they use: a scalar score, or a textual description of the failure.

In the following section, the methods that optimise prompts automatically are introduced.

Further reading: Sutton and Barto [80] is the standard textbook on reinforcement learning; Shao et al. [71] introduce GRPO.

2.6 Automatic Prompt Optimisation and GEPA

Automatic prompt optimisation (APO) replaces the manual rewriting of prompts with a search over prompt text [62]. Methods that use a scalar score generate many candidate prompts and keep those that score well. APE samples candidate instructions from an LLM and filters them by score [107]. OPRO shows an LLM the history of prompts and their scores and asks for a better one [99]. Promptbreeder and EvoPrompt evolve populations of prompts with mutation and crossover [20, 25]. MIPROv2 searches over instructions and examples with Bayesian optimisation [55]. Methods that use textual feedback ask an LLM to criticise a failed output and to revise the prompt accordingly. ProTeGi calls such criticisms "textual gradients" [61], and TextGrad propagates them through programs of several LLM calls [103].

2.6.1 GEPA

GEPA [1] is a textual-feedback optimiser that uses the full record of an episode, its *trace*, as feedback. It maintains a pool of candidate prompts and repeats five steps (Figure 2.3):

1. **Select** a parent prompt from the pool.
2. **Roll out** the parent on a small batch of task instances and record, for each, the score and the trace.
3. **Reflect:** a second language model reads the parent prompt, the traces and the scores, diagnoses what went wrong, and writes a revised prompt. Agrawal et al. [1] call this model the *reflection model*; this thesis calls it the *teacher*.
4. **Accept** the revision if it scores better on the same batch, then evaluate it on a validation set and add it to the pool.
5. **Merge** (optional): combine two candidates that succeed on different instances into one prompt.

GEPA, Promptbreeder and EvoPrompt are evolutionary algorithms in the standard sense [16]: a population of candidate solutions, variation operators that produce new candidates, and selection by a fitness measure. In prompt evolution the candidate is the prompt text and mutation is a rewrite by a language model. Recombination is the merge of two prompts, and fitness is the score of the prompt's episodes. GEPA's distinctive choices are that mutation is guided by traces rather than random, and that selection keeps every candidate that is best on at least one instance. The smallest evolutionary algorithm, the (1+1) scheme, keeps a single

parent, produces one offspring per generation and keeps whichever of the two is fitter; in this thesis, a tie goes to the offspring.

Parent selection uses the *Pareto frontier* of the pool: the set of candidates that are the best on at least one validation instance. Sampling from this set keeps alternative strategies alive when different instances reward different behaviour. The frontier and the merge step therefore operate across task instances; Section 5.7 describes which parts of the algorithm are used in this thesis. Across six benchmark tasks, Agrawal et al. [1] report that GEPA outperforms GRPO by 6 % on average and by up to 20 % while using up to 35 times fewer rollouts. GEPA also outperforms MIPROv2 by more than 10 %.

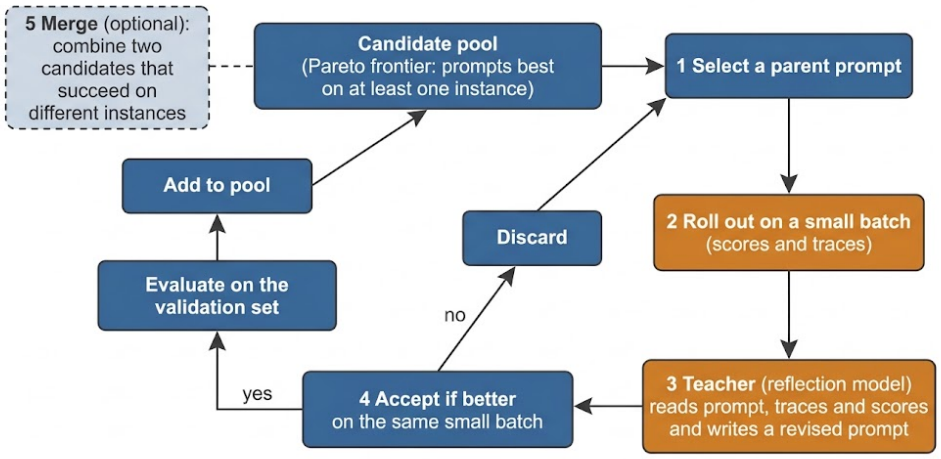


Figure 2.3: The GEPA optimisation loop as described by Agrawal et al. [1].

The output of GEPA is a prompt that a person can read and compare with its predecessor. The same holds for every prompt produced by reflective prompt evolution: the result can be read and compared line by line.

In the following section, the chapter turns from learning-based methods to exact search, which supplies the reference against which the agents are measured on single floors.

Further reading: Ramnath et al. [62] survey automatic prompt optimisation; Agrawal et al. [1] describe GEPA in full.

2.7 State-Space Search and Exact Solvers

A problem in which the outcome of every action is fully determined can be described as a *state space*. A *state* is a complete description of the situation from which the future can be computed. For the game in this thesis, a state consists of the hero's position, hit points (HP), attack (ATK), defence (DEF), magic defence, gold, experience, keys, and the set of tiles already cleared. Each legal action leads from a state to exactly one *successor* state. Starting from the initial state and expanding all successors repeatedly yields a *search tree*, whose leaves are won, lost or unfinished games [65].

Breadth-first search (BFS) expands the tree level by level, so that every state reachable in ℓ actions is visited before any state that needs $\ell + 1$. Exploring the whole tree guarantees that the best leaf is found, here the win with the most HP remaining; a solver with this guarantee is called *exact*. The difficulty is size: with b choices per step and ℓ steps, the tree has on the order of b^ℓ leaves. Exact solvers therefore rely on *pruning*, the safe removal of states that cannot lead to a better result. The central pruning rule is *dominance*. If two states have the same position and the same cleared tiles, and one is at least as good as the other in every attribute and every key colour, the worse one can be discarded. Every continuation of it is available to the better one with at least the same outcome. Applied correctly, dominance pruning preserves exactness; Chapter 4 shows how an incorrect implementation of this rule in a published solver discards the optimum.

A *rule-based heuristic*, by contrast, chooses each action by a fixed rule, for example "take the cheapest available fight". It needs no search, but it offers no guarantee: a locally cheap action can make the game unwinnable later.

Figure 2.4 illustrates these concepts on the upper part of the simple floor, a test map of a published solver for the game and the running example of this thesis (Chapter 4). The hero starts at the top and first picks up the yellow key directly below. It then faces four choices: three monsters (a Slime Lord, a Red Slime and a Black Slime) and a yellow door that the key opens. The exact solver evaluates every continuation of each choice. The highest final HP still reachable from a state is its *state value*. Fighting the Black Slime first keeps the state value at 13 HP; each of the other three choices makes the floor unwinnable. Three of the options look cheap: the door costs no HP, and the Red Slime costs only 1 HP more than the Black Slime. Only search, or equivalent reasoning about the keys and fights ahead, separates them.

In the following chapter, the work that applies these concepts to games, to planning with LLMs and to prompt optimisation is reviewed, and the gap addressed by this thesis is identified.

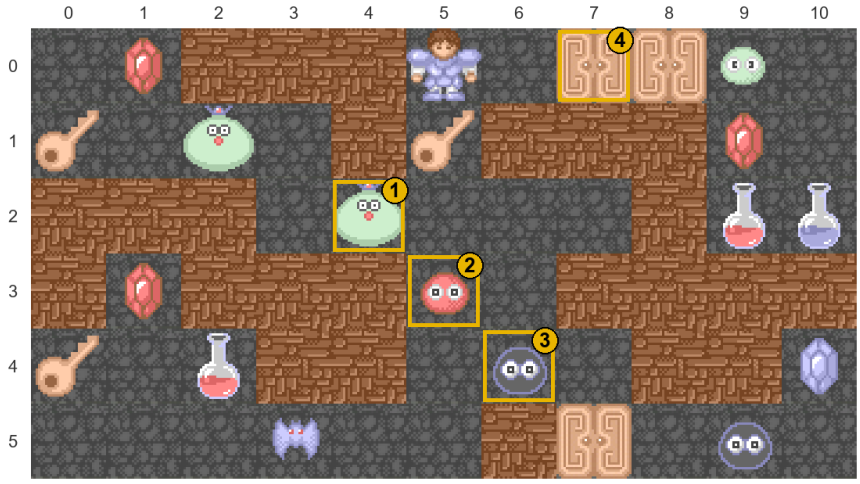


Figure 2.4: Upper part of the simple floor (rows 0 to 5 of 11), drawn with the game's own tiles. Positions are (x, y), with column x counted from the left and row y from the top, both starting at 0. The hero starts at (5,0) and first picks up the yellow key at (5,1). The four choices are then marked: (1) the Slime Lord (4,2), (2) the Red Slime (5,3), (3) the Black Slime (6,4) and (4) the yellow door (7,0). The state value after each choice, the highest final HP the exact solver can still reach, is 13 HP after choice 3 and none after the other three (the boss can no longer be defeated). Other objects: yellow keys and doors, red gems (+1 ATK), blue gems (+1 DEF), red and blue potions (+75 and +200 HP), a Green Slime (9,0), a second Slime Lord (2,1), a Bat (3,5) and a second Black Slime (9,5); brown tiles are walls.

Further reading: Russell and Norvig (2021, Ch. 3) introduce state spaces and uninformed search.

3 Related Work

This chapter reviews the work on which the thesis builds and identifies the gap it addresses. Section 3.1 describes how the literature was collected. Sections 3.2 to 3.6 cover five themes: LLM agents in games, the planning horizon, automatic prompt optimisation, the reliability of LLM evaluations, and the grounding of LLMs in symbolic components. Section 3.7 positions the thesis against the closest works. Each theme follows the same pattern: what a line of work showed, what it assumed, and what it leaves open for the setting studied here.

3.1 Literature Search

The literature was collected in three steps. First, backward and forward citation search started from four anchor works: ReAct [101], GEPA [1], the LLM-Modulo position paper [36] and the BALROG game benchmark [57]. Second, two AI-assisted deep-research tools were given the same written brief of eleven questions. The questions covered GEPA and its follow-ups, the comparison of prompt optimisers, prompt sensitivity, the non-determinism of served models, evaluation methodology, grounding in symbolic components, the planning horizon, risk posture, reasoning faithfulness, game benchmarks and model scale. The keywords of the brief included reflective prompt evolution, automatic prompt optimisation, format sensitivity and non-determinism at temperature 0. Further keywords were pass^k, binomial confidence intervals, LLM-Modulo, plan-and-execute, replanning, risk aversion of LLM agents, chain-of-thought faithfulness and LLM game benchmarks. Third, every returned item was resolved on arxiv.org or the publisher's page before it was cited. Items whose identifier resolved to a different paper, or whose content differed from the tool's summary, were rejected. Inclusion required a peer-reviewed publication or a preprint from 2023 to 2026, or an earlier foundational work, and relevance to at least one of the eleven questions. Grey literature is cited in two places, each marked as such. These are the one prior work on the game that the search found [92] and the explanation of inference non-determinism [30]. The chapter cites 73 works. The verification records are kept with the thesis materials; the use of AI tools in this search is described in the Note on AI Usage.

3.2 LLM Agents in Games and Long-Horizon Environments

Games are long-standing AI benchmarks. Deep reinforcement learning reached human-level play on Atari games [52], and self-play with tree search surpassed human champions in Go [74, 75]. These agents learn a policy over millions of

episodes. LLM agents, by contrast, act from a prompt without game-specific training, and a new generation of benchmarks measures them.

Voyager [84] plays Minecraft with an automatic curriculum and a growing library of code skills and obtains 3.3 times more unique items than earlier agents. Its environment is open-ended and stochastic, which makes it difficult to separate the quality of a plan from the luck of exploration. The NetHack Learning Environment [40], MiniHack [66], Crafter [26] and TextWorld [13] provide roguelike, survival and text-game settings. NetPlay, introduced by its authors as the first zero-shot LLM agent for NetHack, performs best with detailed context and struggles when feedback is missing [33]. BALROG collects several of these environments into one benchmark and finds that current LLMs achieve only partial success even at moderate difficulty [57]. SmartPlay isolates agent capabilities in six games [96], and AgentBoard adds progress-rate metrics [50]. Orak evaluates twelve commercial video games [58], and TextArena ranks agents on 57 competitive text games [23]. In VideoGameBench, the best vision-language models complete under 1 % of the benchmark from raw pixels [104]. Agentick provides 37 procedurally generated tasks with oracle reference policies [6]. In Agentick's 90,000 episodes, a reasoning harness multiplies LLM performance by a factor of 3 to 10, and ASCII observations outperform natural-language ones. These benchmarks measure what current agents can do across many environments. Most of them are stochastic, and none provides a proven optimum for each episode against which the quality of a completed run can be measured. Agentick's oracle policies are the strongest reference behaviours available, and its authors state that they are not optimal on every task.

Two studies are closer to the setting of this thesis. HexMachina evolves a compiled Settlers of Catan player through code refinement between games and reaches a 54 % win rate against the strongest hand-written baseline [3]. The persistent artefact is a program, and the game is stochastic and multi-player. The search of Section 3.1 found one prior LLM study of a Magic Tower style game, an unpublished undergraduate research report [92]. It uses a variant with probabilistic combat and free movement. Humans win 60 to 100 % of games, and Gemini-2.5-Pro wins 40 to 50 % at the hard level; each configuration is run once. Neither work studies a deterministic game in which every outcome can be traced to a decision and compared with an exact optimum.

A further line of work examines the failure that such games isolate: committing scarce, irreversible resources over a long horizon. In EnterpriseArena, 23 LLMs act as a company's chief financial officer over 132 simulated months; only 15.4 % of trials survive, and larger models do not reliably outperform smaller ones [27]. BAGEN finds frontier models over-optimistic about budgets: they keep spending

on tasks that cannot succeed [42]. Studies of risk posture disagree: Jia et al. [34] find loss aversion in LLM choices, while Hartley et al. [29] find GPT-4o close to risk-neutral. Wang [88] measures the gap between an agent's stated rule and its action in poker and finds that misapplications of the agent's own rule are risk-averse in 99.5 % of cases. These studies observe resource failures in simulated economies or card games with stochastic outcomes. In a deterministic game, the same failure can be located at a specific decision and its cost measured exactly, which the thesis does in Chapter 6.

3.3 Planning Horizon

Valmeekam et al. [82] showed that LLMs are poor autonomous planners: in classical planning domains, the best model (GPT-4) generated valid plans in about 12 % of problems on average. Two responses followed. The first adds search around the model. Tree of Thoughts searches over partial solutions [100], and Reasoning via Planning uses the LLM as a world model inside Monte Carlo tree search [28]. Language Agent Tree Search applies the same idea to agent actions [105]. These methods improve results at the price of many model calls per decision. The second response varies the planning horizon, the number of actions the model commits to per call (Section 2.4.1). ReAct interleaves one reasoning step with one action [101]. Plan-and-Solve asks for a plan first and then executes it [85]. ReWOO separates a planner that writes the whole plan in one call from a worker and reports a five-fold saving in tokens over ReAct on HotpotQA [97]. Its authors also name the limit of this design: where little is known about the environment in advance, the planner has to enumerate all plans that could lead to the goal. DEPS and ADaPT replan only when execution fails, and ADaPT argues that a full decomposition up front is neither necessary nor always optimal [60, 89]. Robotics work shows the same design space: Inner Monologue feeds environment feedback into each planning step [32], and ProgPrompt constrains plans to executable program syntax [76].

Some of these studies compare horizons on the same task and model, as ReWOO does against ReAct. Their tasks are question answering, text environments or open-ended games; success is judged by task completion, and plan quality is not compared with an optimum. Whether committing to several actions per model call helps or hurts remains open for a deterministic task with an exact optimum, with the same model and the same observation. It is also open how the answer depends on whether the order of actions is given or must be derived. This is the third research question of the thesis.

3.4 Automatic Prompt Optimisation

Ramnath et al. [62] classify prompt optimisers by the feedback they consume, the search they perform and the budget they need. The scalar-feedback optimisers of Section 2.6 generate many candidates and keep the best. OPRO, for example, improves on human-designed prompts by up to 8 % on GSM8K and up to 50 % on Big-Bench Hard [99], and the DSPy framework provides MIPROv2, which tunes instructions and examples together [37, 55]. EXPO is a scalar method designed for sequential decision-making agents; it treats the non-stationary rewards of such agents as an adversarial bandit problem [39]. The textual-feedback optimisers of Section 2.6 revise the prompt from a description of its failures. Beyond ProTeGi and TextGrad, Trace learns from execution traces [9], PromptAgent searches over edits with a tree [86], and GEPA reflects over full traces [1]. Self-Refine [51] and Reflexion [73] apply the same mechanism to a single output or to an episodic memory rather than to the system prompt.

Recent work qualifies GEPA's results. Naive Prompt Optimization revises a single prompt lineage from recent traces, without a Pareto frontier or population, and "achieves comparable or better performance than GEPA with fewer rollouts" [7]. In 22 interactive games it remains broadly competitive with GEPA, and its authors leave open whether the result extends to long-horizon agent environments. The (1+1) loop of Section 5.7 follows the same single-lineage design. ESPO documents that GEPA's prompts grow up to three times longer without becoming more accurate, and it outperforms GEPA by 3.8 percentage points with prompts 47 % shorter [44]. Reflection in the Dark shows GEPA lowering accuracy from a defective seed prompt, from 23.8 % to 13.5 % on GSM8K [47]. ContraPrompt extracts rules from the difference between a failed and a successful trace on the same input [64]. ETGPO reaches GEPA's accuracy at a third of the budget by first building an error taxonomy [77].

Apart from EXPO and the interactive games of Chang and Chen [7], these methods are evaluated on tasks with many instances, cheap evaluations and a fixed input format, such as question answering, mathematics and classification. They assume that the score of a candidate prompt can be measured precisely enough to rank it. None of them has been evaluated on a long-horizon game in which one episode costs dozens of model calls, the outcome of a single episode varies between repetitions, and the agent's observation is itself engineered. Whether prompt optimisation still adds to an agent whose observation already carries the relevant computations is the second research question of the thesis.

3.5 Prompt Sensitivity and the Reliability of LLM Evaluations

Meaning-preserving changes to a prompt can change results substantially (Section 2.3). Beyond the formatting effects reported there, Sclar et al. [69] find that format performance correlates only weakly across models. Lu et al. [49] show the same sensitivity for the order of few-shot examples. Errica et al. [17] propose sensitivity and consistency metrics over rephrasings, and Ngweta et al. [54] reduce format brittleness by mixing formats in the prompt. The software-engineering response treats prompts as versioned artefacts with regression tests [8, 19]. These studies vary wording, layout and examples written by people. In the studies reviewed here, the structured, machine-generated part of an agent's prompt, such as a serialised game state with numeric fields and placeholder values, has not been examined in the same way.

A second source of variation is the inference service itself. Section 2.2.2 explains, following a technical blog post [30], why hosted models are not deterministic at temperature 0. Atil et al. [2] measure accuracy variations of up to 15 % across nominally deterministic runs of five models on eight tasks. Wimbauer et al. [95] show that such numerical deviations identify the serving system from the generated text alone. Song et al. [78] argue that greedy single-sample evaluation is not a reliable estimate of capability. Surveys of agent evaluation list reliability as a separate evaluation objective [53], and τ-bench reports the probability that an agent succeeds in all of k repeated trials [102]. Binomial confidence intervals for such success rates are standard in statistics [4, 94]. Of the closest works compared in Table 3.1, four report a spread over repeated runs and five report a single result per configuration without one. GEPA, for example, gives one test score per optimiser and benchmark, sampled at temperature 0.6 or 1.0 [1]. Finally, Section 2.2.1 notes that the reasoning text of a model need not reflect the process that produced its answer [41, 81]. Wang [88] finds that reading the decision from an explicit tag reduces reasoning-to-action inconsistency from 22–26 % to 0.7–1.4 %.

Together, these studies show that a single run of a prompt on a hosted model is weak evidence. They leave open how much repeated runs change the conclusions of an agent study, and whether conclusions drawn from single runs survive repetition. The thesis treats this as a methodological question in its own right (Chapter 5 and Chapter 7).

3.6 Grounding LLMs in Symbolic Components

Several lines of work pair an LLM with a component that computes exactly. LLM+P lets the model translate a natural-language problem into the Planning

Domain Definition Language and hands the search to a classical planner [43]. With an optimal planner, it returns optimal plans for most problems, whereas the model planning alone fails to produce even feasible plans for most of them. The LLM-Modulo framework treats the LLM as an approximate generator whose proposals are checked by external sound verifiers [36]. With such verifiers, every plan the system returns satisfies the verified constraints, and success rates rise over the model alone [24]. SayPlan grounds a robot's plans in a 3D scene graph and a classical path planner [63]. Classical planning itself goes back to STRIPS [21] and requires a complete formal domain model [22]. On the output side, constrained decoding guarantees that an answer follows a grammar or schema (Section 2.3 [93]), and tool-use models learn to emit valid API calls [59].

These approaches mostly ground the model's output: they verify a plan after it is proposed, or translate the problem into a form a planner can solve. SayPlan's scene graph enters the input, but as a description of the world, not as the computed consequences of each action. Constrained decoding guarantees that an answer is well formed, not that it denotes a legal action with known consequences. Less studied is grounding on the input side. There, the symbolic component computes the consequences of every legal action in advance and writes them into the model's observation, and the model chooses among them by identifier. How the design of that observation affects decisions, and which kinds of annotation help or mislead, is part of the first research question of the thesis.

3.7 Positioning

Table 3.1 compares the closest works with this thesis on six properties.

Table 3.1: Positioning of the thesis against the closest related works. Deterministic = the environment has no random outcomes; Exact optimum = each episode can be compared with a provably optimal solution; Repeated runs = each configuration is run several times and reported with its spread; Prompt optimisation = the prompt is optimised automatically; Planning horizon = planning horizons are compared on the same task and model; Engine-annotated observation = a symbolic component computes the consequences of actions into the model's input. ✓ = yes, ✗ = no, n/a = not an environment study.

Work	Deterministic	Exact optimum	Repeated runs	Prompt optimisation	Planning horizon	Engine-annotated observation
BALROG [57]	✗	✗	✓	✗	✗	✗
Agentick [6]	✗ (stochastic tasks)	✗ (oracle policies not always optimal)	✓	✗	✗	✗
Voyager [84]	✗	✗	✓	✗	✗	✗

Work	Deterministic	Exact optimum	Repeated runs	Prompt optimisation	Planning horizon	Engine-annotated observation
HexMachina [3]	✗	✗	✓	✗ (evolves code)	✗	✗
Magic Tower report [92]	✗	✗	✗	✗	✗	✗
ReWOO [97]	n/a	n/a	✗	✗	✓	✗
LLM+P [43]	✓	✓	✗	✗	✗	✗
GEPA [1]	n/a	n/a	✗	✓	✗	✗
Naive Prompt Optimization [7]	n/a	n/a	✗	✓	✗	✗
This thesis	✓	✓	✓	✓	✓	✓

Overall, the literature suggests four things. LLM agents in games are evaluated mostly in stochastic environments without an optimum. Where the observation is varied, the variation concerns its format, not the computed consequences of actions. Prompt optimisers are evaluated on short tasks with cheap, repeatable scores. Grounding work verifies what the model outputs rather than shaping what it sees. This combination motivates the design of this thesis: a deterministic game with an exact solver, and an engine that annotates every legal action with its computed consequences. A repeated-episode protocol then measures the contributions of the observation, of prompt optimisation and of the planning horizon separately.

In the following chapter, the game and its formal task definition are introduced.

4 Task and Environment

This chapter describes the environment in which all agents of this thesis act and the exact reference against which they are measured. Section 4.1 introduces Magic Tower and its game rules, and Section 4.2 the deterministic combat formula. Section 4.3 collects the strategy principles that follow from these rules or from expert play, and states which of them the prompts contain. Section 4.4 defines the task formally. Section 4.5 describes the floors used in the experiments, with the simple floor as the running example, and the scripted events of the tower. Section 4.6 presents the exact solver that provides the optimum for every single floor, and the procedure by which the hand-designed floors were built with it. Section 4.7 describes the two simple reference agents.

4.1 Magic Tower

Magic Tower (Chinese *Mota*) is a family of grid-based puzzle role-playing games. This thesis uses the 24-floor edition in the open-source HTML5 implementation of the game [12], whose floor, monster and item data are read directly by the simulation engine (Chapter 5).

The hero moves on a grid of tiles. Each floor of the tower is a grid of 13 by 13 tiles; the single-floor test maps measure 11 by 11 or 13 by 13 tiles. Every tile holds exactly one element:

- **floor** tiles can be walked on, and **walls** cannot;
- **monsters** block the tile they stand on until the hero defeats them;
- **doors** block the tile until they are opened with a key of the same colour (yellow, blue or red), which is consumed;
- **items** are collected by stepping on them: red gems add attack, blue gems add defence, green gems add magic defence, potions add hit points, keys add one key of their colour, and swords and shields add larger amounts of attack or defence;
- **stairs** lead to the next or the previous floor;
- **shops** exchange gold or experience points (EXP) for attributes or keys.

The hero is described by hit points (HP), attack (ATK), defence (DEF), magic defence (MDEF), gold, EXP and the number of keys of each colour. The hero dies when HP reaches 0. Monsters, doors and items disappear permanently once they are defeated, opened or collected.

Two properties make the game suitable for studying planning. First, it is fully deterministic: there are no random events, and the cost of every fight is known before it starts. Every outcome can therefore be traced to a decision of the agent,

not to chance. Second, all resources are irreversible. A key spent on one door is not available for another. HP lost in a fight is recovered only through a limited number of potions and, on the tower, through levels, the Holy Water and shop purchases. The order in which fights, doors and items are taken therefore decides whether a floor can be won and with how much HP.

4.2 Combat

A fight is resolved in rounds. The hero strikes first; then hero and monster strike alternately until the monster's HP reaches 0. Let a_h, d_h be the hero's ATK and DEF and h_m, a_m, d_m the monster's HP, ATK and DEF. The hero deals $a_h - d_m$ damage per strike and therefore needs

$$T = \left\lceil \frac{h_m}{a_h - d_m} \right\rceil$$

strikes; if $a_h \leq d_m$, the hero deals no damage and the fight cannot be won. Because the hero strikes first, the monster strikes $T - 1$ times, each for $\max(0, a_m - d_h)$ damage. The HP cost of the fight is

$$D = \max(0,\ a_m - d_h) \cdot (T - 1),$$

reduced by the hero's MDEF, and the fight is survivable if D is smaller than the hero's HP. Two consequences drive the strategy of the game. Each point of ATK lowers T and thus the number of monster strikes, and each point of DEF lowers the damage of every strike. A stat item collected before a fight can therefore save more HP than its face value, because the point applies to every later strike. A monster whose ATK does not exceed the hero's DEF costs nothing at all.

Monsters can carry special abilities that modify the formula; eight of them occur on the floors used in this thesis (Table 4.1; double and triple attack share a row). The full algorithm, including the order in which the abilities are applied, is given in Appendix B.

Table 4.1: Special abilities of monsters that occur on the floors used in this thesis. d = the monster's damage per strike after DEF. The further abilities that the engine implements (solid, counter attack, purify, mimic) and those the game defines without applying them in combat (poison, weaken, curse) occur on none of these floors.

Ability	Effect on the fight	Where it occurs
Magic attack	The monster's strikes ignore the hero's DEF: $d = a_m$.	simple, complex, all hand-designed floors, 13 of 15 generated floors
First strike	The monster strikes once before the hero: T strikes instead of $T - 1$.	complex, hand01–hand03
Double / triple attack	The monster strikes two or three times per round.	complex, hand03
Four-strike attack	The monster strikes four times per round.	the boss of complex and hand03
Break armour	Adds $\lfloor 0.9 \cdot d_h \rfloor$ damage before the fight; penalises high DEF.	complex, hand03
Fixed damage	Adds a fixed amount of damage to the fight.	tower, wizards from MT6
Vampire	Drains a fixed share of the hero's current HP before the fight.	tower, from MT7

On the simple floor, hand04 and the generated floors, magic attack is the only ability; hand01 and hand02 add one first-strike monster each. The complex floor and hand03 combine six abilities, which is one reason they are harder. On the tower up to MT21, where defeating the Nether Demon King brings the game's *normal ending* (Section 4.5.3), only fixed damage and vampire occur.

On the simple floor, the hero starts with 470 HP, ATK 9 and DEF 4. A Green Slime (HP 50, ATK 7, DEF 3) takes $\lceil 50/6 \rceil = 9$ strikes and costs $3 \cdot 8 = 24$ HP. A Black Slime (HP 45, ATK 5, DEF 5) has magic attack: it takes 12 strikes and costs $5 \cdot 11 = 55$ HP, where the same monster without the ability would cost 11. The boss (HP 50, ATK 20, DEF 12) cannot be fought at all with ATK 9. At the optimal final state of ATK 13 and DEF 8, it takes 50 strikes and costs $12 \cdot 49 = 588$ HP. To win, the hero must therefore reach the boss with ATK above 12 and more HP than the boss fight costs.

4.3 Strategy Principles

The game rules of Sections 4.1 and 4.2 say what is allowed; they do not say what is good. A *strategy principle* is a regularity of good play that the game does not enforce. Table 4.2 lists the principles used in this thesis, each labelled by its source. The sources are mainly the combat formula, the exact solver of Section 4.6 and an expert walkthrough of the 24-floor edition [18]. The walkthrough is a community text in Chinese whose author is unknown; it is archived with the repository. In four cases, the source is a game rule, the game data, the tower's structure or a design choice of this thesis. The last column states how the agent meets each principle: as a numbered *prompt rule* of a system prompt (Section 5.4, Appendix A), as a computed line of the observation (Section 5.2, Section 5.5.3), or not at all. The prompts may state general principles but no floor-specific knowledge (Section 5.4). The walkthrough principles S11 to S14 are specific to floors of the tower and therefore never appear as prompt rules. Where the engine shows a fact that bears on them, the fact is computed for the current state, and the decision is left to the model.

Table 4.2: Strategy principles, their source and how the agent meets them. Rule n = prompt rule n of the tower prompt (Appendix A.2). The single-floor prompt (Appendix A.1) states tower rules 2, 4, 5 and 6 as its rules 1, 3, 4 and 5; its rule 3 uses 40 % instead of 60 % and omits the clause on gold and EXP. It states the lowest-effective-cost preference of tower rule 5 as its rule 2, and tower rules 1 and 3 have no single-floor counterpart.

#	Principle	Source	How the agent meets it
S1	DEF lowers every strike; a monster whose ATK does not exceed the hero's DEF costs nothing.	combat formula	rule 2; `[FREE if DEF>=X]` on every fight except magic-attack fights and, on the tower, vampire fights
S2	ATK lowers the number of rounds; just above a monster's DEF it is winnable but still expensive, and the cost falls only at the values of ATK where $\lceil h_m/(a_h - d_m) \rceil$ drops.	combat formula	rule 3; `[WINNABLE at ATK>=X]` on lethal fights
S3	Take a stat item before the fights it makes cheaper.	combat formula, exact solver	rule 2; net stat-route and fight-order lines (Sections 5.2.3, 5.2.4)

#	Principle	Source	How the agent meets it
S4	Potions add HP whenever they are taken; their order does not change any fight.	combat formula	potions listed separately from the lookahead value (Section 5.2.2)
S5	Never pay HP for a fight that unlocks nothing that is needed.	exact solver	rule 5; lookahead tree
S6	A key spent on one door is lost for every other, and keys lie behind doors; the choice of door can decide the floor.	exact solver (on the simple floor, opening the yellow door (7,0) at the first decision makes the floor unwinnable, Figure 4.3)	rule 6; key lines and key budget (Section 5.2.5)
S7	Do not take a fight that costs a large share of the current HP while anything else can still make progress.	design of this thesis (irreversibility of HP)	rule 4 (60 %); single-floor rule 3 (40 %)
S8	On the tower, see the next floor before spending on the current one.	tower structure	rule 1; the engine takes the staircase to an unvisited floor itself (Section 5.6)
S9	In shops, never buy HP; buy DEF before ATK, but enough ATK for the fights on the way up; buy where a point is cheapest.	combat formula, expert walkthrough [18]	rule 3; way-up line, purchase and shop-price lines (Section 5.5.3)
S10	Hand in the Cross late: its gain is a third of the stats at that moment.	game rule of the Cross	rule 3; Cross line
S11	The first Red Bat on MT8 needs far more ATK than the monsters before it; the walkthrough pays about 72 % of the hero's HP for it.	expert walkthrough [18]	not stated; way-up line computes the cost
S12	Keep a red key for the door in front of the Red Demon King on MT16; red keys are the scarcest resource of the tower.	expert walkthrough [18]	not stated; key-reserve line (Section 5.5.3)

#	Principle	Source	How the agent meets it
S13	Save 500 EXP and 500 gold for the Holy Sword (+120 ATK) and the Holy Shield (+120 DEF) on MT15; the Holy Sword raises ATK towards the King's DEF of 1,000, and the Holy Shield lowers or removes his cost.	expert walkthrough [18], game data	not stated; trade actions and treasure line (Section 5.5.3)
S14	Take the Holy Water (HP doubled) on MT14 as late as possible.	expert walkthrough [18]	not stated; the engine collects the other potions itself but leaves the Holy Water to the model

Principles S1 to S7 are told or shown on every floor, and S8 to S10 on the tower. S11 to S14 are knowledge that an expert brings to the tower and the agent does not; it must find them from the computed lines. Several tower runs of Section 6.7.3 stop where they apply: at the Red Bats on MT8, and at the red door and the Red Demon King on MT16.

4.4 Task Definition

The task studied in this thesis is to win a floor: to defeat its boss and survive, or, on floors without a boss, to reach a given goal. The agent never moves tile by tile. At every turn, the engine enumerates the *targets* the hero can reach and act on, and the agent chooses one of them; the engine then walks the shortest path and performs the action.

Definition 4.1 (Floor task). A floor task is a tuple (S, s_0, A, δ, G).

- A **state** $s \in S$ consists of the hero's position, HP, ATK, DEF, MDEF, gold, experience and keys per colour, together with the set of monsters, doors and items remaining on the floor.
- s_0 is the initial state defined by the floor file.
- The **legal actions** $A(s)$ are the actions on targets reachable from the hero's position without passing a monster or a closed door: fight a reachable monster that the hero can damage ($a_h > d_m$), open a reachable door for which a key is held, collect a reachable item, use a shop, or take a reachable staircase. A fight that

would kill the hero is legal and ends the episode; under the final configuration, the engine lists it with a tag and does not execute it (Section 5.6).

- The **transition** $\delta(s, a)$ is deterministic: it applies the combat formula of Section 4.2 or the effect of the item, door, shop or staircase, and removes the monster, door or item.
- The episode ends when the goal is reached, when the hero dies, or when $A(s)$ contains no action that keeps the hero alive.
- The episode is **successful** if the goal G is reached with HP > 0. Among successful episodes, the outcome is the final HP.

The optimum of a floor, $\mathrm{HP}_{\mathrm{opt}}$, is the largest final HP of any successful action sequence. It exists for every single floor used here, because the state space is finite and at least one winning sequence exists. It is computed exactly by the solver of Section 4.6. Chapter 5 defines how success rates and the ratio of final HP to $\mathrm{HP}_{\mathrm{opt}}$ are measured over repeated episodes.

For the tower, the task spans consecutive floors, the hero keeps its state between them, and staircases connect the floors. In the game, walking between visited floors costs nothing. A tower state's legal actions are therefore the actions on every reachable target of every visited floor. A staircase appears only when it leads to a floor not yet visited, and the engine then takes it itself (Sections 5.5.3 and 5.6). Tower runs are measured by the highest floor reached and by whether the normal ending on MT21 is reached (Section 5.9.5).

4.5 Floors

Table 4.3 lists the floors used in the experiments.

Table 4.3: Floors used in this thesis. Monsters = monster tiles including the boss; Start = hero HP / ATK / DEF at the start; $\mathrm{HP}_{\mathrm{opt}}$ = exact optimum with the hero's final ATK and DEF (the complex optimum also has MDEF 20); Winning sets = distinct sets of monsters and doors cleared on the way to a win, not computed for the generated floors; States = states expanded by the exact solver. "—" = not applicable. Maps of the six fixed single floors in Appendix B.5.

Floor	Size	Monsters	Start	$\mathrm{HP}_{\mathrm{opt}}$ (ATK, DEF)	Winning sets	States	Role
Simple	11 × 11	14	470 / 9 / 4	13 (13, 8)	1	412	development, main evidence and running example
Complex	13 × 13	25	1000 / 10 / 10	68 (16, 20)	32	729,953	boundary of the agent

Floor	Size	Monsters	Start	HP_{opt} (ATK, DEF)	Winning sets	States	Role
hand01	11 × 11	13	400 / 9 / 4	14 (14, 7)	1	461	not used in development
hand02	11 × 11	13	62 / 9 / 4	13 (13, 9)	1	1,539	not used in development
hand03	13 × 13	16	649 / 10 / 10	9 (14, 14)	1	118,351	not used in development
hand04	11 × 11	14	94 / 9 / 4	3 (13, 8)	1	7,917	not used in development
Generated (15)	11 × 11	8–14	470 / 9 / 4	13–250	not computed	36–652	prompt optimisation
Tower MT0–MT23	13 × 13 per floor	—	1000 / 10 / 10	—	—	—	long horizon

The **simple** and **complex** floors are the single-floor test maps of the published reference solver [11]. The simple floor has exactly one winning set and an optimum of 13 HP. This final HP of the optimal route is the floor's *slack*: a decision that wastes 13 HP or more loses the floor. It is the floor on which the agent was developed and on which the error analysis of Chapter 6 is carried out. The complex floor has more doors, monsters with magic attack and break armour, and many more states. Its optimum requires taking the expensive fights only after the stat items that make them cheap.

The four **hand-designed floors** were built for this thesis. Their layouts were drafted by a large language model, Claude Opus 5.5 (Anthropic), used as a coding assistant. Every version was solved exactly. A version was kept only when it met four design targets: exactly one winning set, a small slack, failure of the rule-based heuristic, and no win in 500 random games (Section 4.6.3). They are used only for evaluation. The fifteen **generated floors** come from a generator that joins nine 3 × 3 rooms by a random spanning tree with one or two extra links (Appendix B.3). Each link becomes a door, a monster or an open passage, and the rooms are filled with the simple floor's monsters and items. A floor is kept only if the exact solver wins it with an optimum between 1 and 250 HP. The rule-based heuristic wins 8 of them. They are the training and held-out floors of the prompt optimisation study.

4.5.1 The Running Example: the Simple Floor

Figure 4.1 shows the simple floor in full. The hero starts at the top with 470 HP, ATK 9, DEF 4 and no key; the boss stands at the bottom and is reached only past a Bat. The floor has four yellow doors and five yellow keys, four red and four blue gems, and six potions; besides the boss, the thirteen monsters are slimes, Slime Lords and two Bats.

The floor falls into regions, each opened by a door or a monster. Below the start,

the Black Slime (6,4) or the Red Slime (5,3) opens a *central corridor*. From it, the Bat (3,5) leads to a *left corridor* with a red gem, a key and a potion, the yellow door (7,5) to the *right-hand rooms*, and the Green Slime (5,6) to the *boss corridor*, which ends with the Bat (5,9) and the boss (5,10). Off the boss corridor, the yellow door (4,7) and the Black Slime (3,7) lead into a *lower-left room* with two keys and a blue gem; at its far end, the Slime Lord (1,9) guards a blue gem and a blue potion. At the top, the yellow doors (7,0) and (8,0) and the Green Slime (9,0) lead to a *top-right room* with a red gem and two potions, and the Slime Lords (4,2) and (2,1) guard a *top-left corner* with a red gem and a key. The right-hand rooms hold a potion behind the door, a blue gem and a potion behind the Black Slime (9,5), and a red gem and a blue gem behind the Red Slime (10,7) and the Slime Lord (9,9).

The exact solver finds a single winning set: the optimal route opens all four doors and fights thirteen of the fourteen monsters, ending with 13 HP at ATK 13 and DEF 8. Only the Red Slime at (5,3) is left. Table 4.4 lists the route action by action, and Figure 4.1 numbers it on the map. In terms of regions, the route enters the central corridor, the boss corridor and the lower-left room, then takes the top-right room and the left corridor, enters the right-hand rooms, fights the Slime Lord (1,9) at the far end of the lower-left room, finishes the right-hand rooms and the top-left corner, and ends with the Bat (5,9) and the boss. Counting each monster and door separately, the route has seventeen actions; the published solver merges some neighbouring obstacles and counts twelve decisions (Section 4.6.1). For each of these seventeen actions, the solver also evaluates every legal alternative. At all seventeen, at least one alternative makes the boss unwinnable (the action is *critical*), and at ten of them every alternative does (the action is *forced*). On average, 87 % of the legal alternatives at an action of the optimal route lose the game. The first of these actions answers the first decision of Figure 2.4.

Table 4.4: The optimal route of the simple floor. Cost = HP lost in the action; ATK / DEF = before the action; Behind it = what the action makes reachable (collected without a decision); Losing = legal alternatives after which the boss can no longer be defeated, of all legal alternatives; Turn = turn of the same action on the winning trajectory of the model's episodes (Section 6.4), which takes action 11 before action 8 and ends at 2 HP.

#	Action	Cost	ATK / DEF	Behind it	Losing	Turn
1	Black Slime (6,4)	55	9 / 4	central corridor	3 of 3	2
2	Green Slime (5,6)	24	9 / 4	boss corridor	3 of 5	3
3	door (4,7)	0	9 / 4	Black Slime (3,7)	6 of 6	4
4	Black Slime (3,7)	55	9 / 4	lower-left room: two keys, blue gem	4 of 4	5

#	Action	Cost	ATK / DEF	Behind it	Losing	Turn
5	door (7,0)	0	9 / 5	door (8,0)	5 of 6	9
6	door (8,0)	0	9 / 5	Green Slime (9,0)	5 of 6	10
7	Green Slime (9,0)	16	9 / 5	top-right room: red gem, two potions	5 of 5	11
8	Bat (3,5)	144	10 / 5	left corridor: red gem, key, potion	3 of 4	18
9	door (7,5)	0	11 / 5	right-hand rooms: potion	4 of 4	22
10	Black Slime (9,5)	35	11 / 5	blue gem, potion	4 of 4	24
11	Slime Lord (1,9)	52	11 / 6	blue gem, blue potion	3 of 4	15
12	Red Slime (10,7)	24	11 / 7	Slime Lord (9,9)	3 of 3	27
13	Slime Lord (9,9)	39	11 / 7	red gem, blue gem	3 of 3	28
14	Slime Lord (4,2)	22	12 / 8	Slime Lord (2,1)	1 of 2	31
15	Slime Lord (2,1)	22	12 / 8	top-left corner: red gem, key	1 of 2	32
16	Bat (5,9)	81	13 / 8	boss	1 of 1	35
17	boss (5,10)	588	13 / 8	—	1 of 1	36

4.5.2 The Tower

The tower consists of floors MT0 to MT23; the last floor is split into three parts in the game data. Stairs connect neighbouring floors in both directions, and the hero may go back to any visited floor at no cost to collect what was left. The monsters' attributes are taken from the game data. All six shops of the game are modelled, as are the gift characters on the second floor. With the final tower observation, no shop sells yellow or blue keys, although the original game does: an expert never buys them, and the gold is needed for attributes. Blue keys are offered again only for the continuation from MT18 in Section 6.7.3. Two shops shape the early tower: the gold shop on MT3, reached once its yellow door is open, and the experience shop on MT5, where 30 EXP buy 5 ATK or 5 DEF. The only way to the experience shop passes a Big Bat (150 HP, ATK 65, DEF 30). It costs 1,470 HP at ATK 33 and DEF 35 but 322 HP at ATK 40 and DEF 42. The scripted events of the tower, and which of them the engine models, are summarised in Section 4.5.3.

The two earlier tower observations of Section 5.5.3 ran with the values of nine items (three swords, four shields, the green gem and the pack) from another edition of the game. The engine of the final tower observation uses the item values of the game data. It models the hero's level and the two items that raise it besides the experience shops: the Small Feather on MT6 (one level: +1,000 HP, +10 ATK,

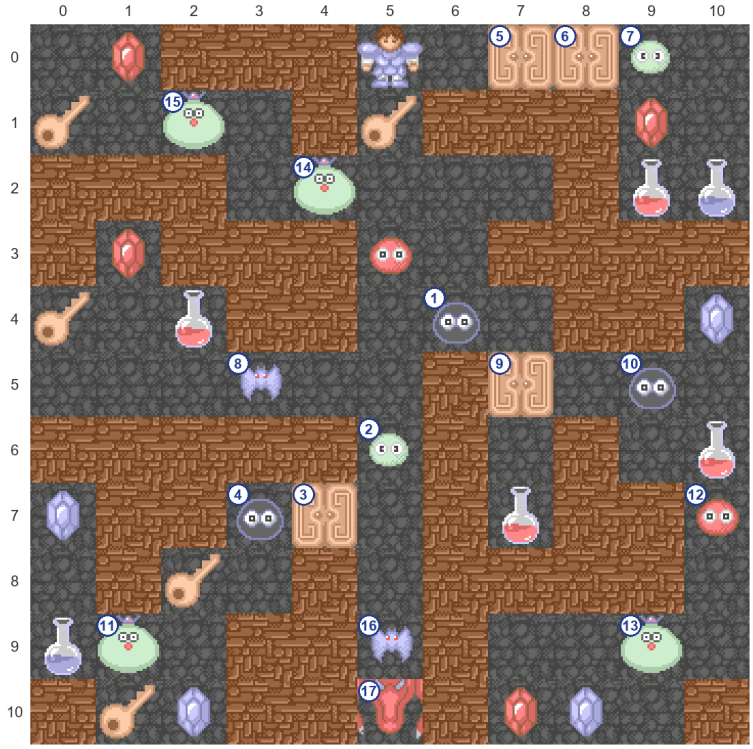


Figure 4.1: The simple floor, drawn with the game's own tiles; the numbers 1 to 17 mark the actions of the optimal route in the order of Table 4.4. Positions are (x, y), with column x counted from the left and row y from the top, both starting at 0. The hero starts at (5,0) with 470 HP, ATK 9, DEF 4 and no key; the boss (5,10) is reached only past the Bat (5,9). Other objects: yellow doors and keys, red gems (+1 ATK), blue gems (+1 DEF), red potions (+75 HP), blue potions (+200 HP), green, red and black slimes, four Slime Lords and a second Bat (3,5); brown tiles are walls. The same map as a text grid is given in Appendix B.5.1.

+10 DEF) and the Big Feather, which the wizard on MT13 hands over (three levels: +3,000 HP, +30 ATK, +30 DEF). It also models the MT4 thief, the Cross at the MT0 fairy and the seal in front of the Cross on MT7, which opens only when all four White Warriors on that floor are dead. Some scripts of the upper floors are not modelled (Section 4.5.3). The engine's map can therefore differ from the game's in the tiles these scripts change; the highest floor reached is a result on the engine, not a claim about the original game.

4.5.3 Scripted Events of the Tower

A *scripted event* is a script attached to a tile that changes the map or the hero beyond what the tile's kind does. Shops are part of the game rules (Section 4.1)

and are not counted here. Table B.4 in Appendix B.7 lists every scripted event of the game data up to MT21, where the normal ending is reached, and summarises the floors above it. It includes the items whose effect is itself a script (key box, Holy Water, feathers, Cross), because the engine had to model each of them separately. The events that change play are of four kinds. Characters give or trade items: the fairy's keys on MT0, the sword and shield on MT2, the Big Feather on MT13 and the two trades on MT15. Scripts change the map: the MT4 thief opens a passage on MT2 and later two tiles on MT18, a seal on MT7 guards the Cross, and the up-staircases of MT18 and MT20 stay hidden until their events. Items act through scripts: the key box, the Small Feather, the Holy Water and the Cross, which raises HP, ATK and DEF by a third at the fairy. Boss fights upgrade monsters: after the Red Demon King on MT16, three monster kinds and the King himself become stronger, and after the first fight with the Nether Demon King on MT19, he becomes stronger for the second fight on MT21. Of the 23 rows of Table B.4, the engine of the final tower observation models 19, one of them only in part. Two need no model, the monster manual and the floor-flying item; two are not modelled, the story item of MT16, which matters only for the ending, and the events after the normal ending.

The two MT15 trades and the red door in front of the Red Demon King decide whether MT16 can be passed. The King (15,000 HP, ATK 1,000, DEF 1,000) can be fought once the hero's ATK exceeds 1,000 and costs nothing once DEF reaches 1,000. The door needs a red key that must be kept or bought (principles S12 and S13 of Table 4.2).

4.6 The Exact Reference Solver

The reference solver was published for the game series by ckcz123 [11]. It searches for the best possible result of a floor, the win with the most HP left, over the decisions that matter instead of over the hero's steps on the grid. Its published implementation contains an error in the pruning of states; the corrected version, described at the end of Section 4.6.2, is the *exact solver* used throughout this thesis. It works in two stages: it first turns the grid into a graph of the things the hero can interact with (Section 4.6.1), and then searches over orders in which those things can be cleared (Section 4.6.2). Section 4.6.3 describes how the solver is used as the baseline of all experiments and as the tool with which the hand-designed floors were built.

4.6.1 From Grid to Graph

Every tile that holds something is a **node**: each monster, door and item, the boss, and the hero's starting position. Two nodes are joined by an **edge** when

the hero can walk from one to the other over empty floor, moving in the four grid directions. The walk may not pass through another node (the hero's start tile counts as empty floor), so an edge means "once one of the two is cleared, the other is directly reachable". Figure 4.2 shows the result for the simple floor before merging: 38 nodes and 55 edges. The graph drops grid distances and keeps only which objects block which. The yellow key at (5,1), directly below the start, is joined to the start, to the yellow door (7,0) and to three monsters, the Slime Lord (4,2), the Red Slime (5,3) and the Black Slime (6,4); once the key is taken, these are the four choices of Figure 2.4. The boss at (5,10) has a single neighbour, the Bat at (5,9), so every win fights the Bat before the boss.

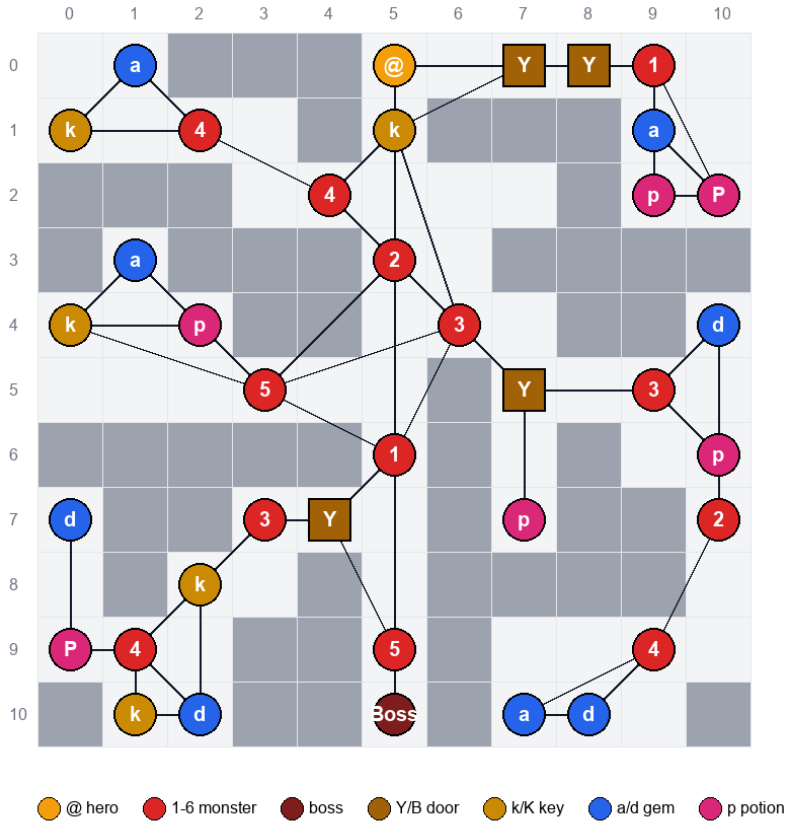


Figure 4.2: The exact solver's node graph of the simple floor before merging, drawn at the map positions. Circles: monsters (numbers as in Appendix B.5.1), boss, keys (k), gems (a = +ATK, d = +DEF), potions (p = +75 HP, P = +200 HP); squares: doors (Y yellow); @ = the hero's start. An edge joins two nodes that are connected by empty floor; grey tiles are walls.

When a node is cleared, the hero stands on its tile and inherits its edges. The set of

nodes adjacent to the hero is therefore always the set of things the hero can reach next, which corresponds to the action list the engine offers the language model (Chapter 5). The solver also merges neighbouring items into one node, because they are always taken together. Under a strict condition (Appendix B.2.1), it merges neighbouring obstacles as well, so that one of its decisions can clear more than one monster or door. Neither merge changes any result.

4.6.2 Search

A **state** of the search is the hero's attributes (HP, ATK, DEF, MDEF, keys), the node the hero stands on, and the set of nodes already cleared. A **decision** clears one adjacent monster or door (or a merged group of them, Section 4.6.1): the solver subtracts the exact damage of Section 4.2 or uses a key. A decision that would kill the hero or needs a missing key is not allowed. After every decision, the solver takes every adjacent item and fights every adjacent monster that costs 0 HP, repeatedly until none is left. On the single floors solved here, doing so never lowers the final result: items only add to the hero's attributes, and a free fight only removes an obstacle. The two abilities under which an item could raise a later cost do not interfere. Vampire, whose drain grows with the hero's HP, occurs on no single floor. Break armour adds $\lfloor 0.9 \cdot d_h \rfloor$, but every break-armour monster on these floors strikes at least twice, so a point of DEF saves more than it adds. This step is why the search runs over decisions rather than moves: collecting a potion is never a choice.

The solver keeps a queue of states, ordered by the number of decisions made and then by HP. It repeatedly takes the first state, stops that line of search when the boss has been defeated or after 45 decisions, and otherwise adds one successor state for every adjacent monster or door. Every decision clears one monster or door node, and no single floor of this thesis has more than 32 such nodes, so the limit of 45 decisions never cuts a route short. A state in which the boss is defeated is a **win**; the set of monsters and doors cleared on the way is its **winning set**, and the optimum $\mathrm{HP}_{\mathrm{opt}}$ is the highest HP among all wins.

The search stays small because of **dominance** (Section 2.7). Since every item on the way is collected automatically, the set of cleared monsters and doors fixes the hero's ATK, DEF, MDEF and keys. Two states with the same cleared set can differ only in HP and in the hero's node. The cleared nodes form one connected region that the hero crosses at no cost, so the node does not change what can be reached next, and the state with less HP can never do better. The solver therefore keeps, for every cleared set, only the state with the most HP. The cleared set is stored as a 64-bit mask, which limits a floor to 64 monsters and doors. Algorithm 4.1 summarises the search.

Algorithm 4.1: Exact search over decisions with dominance (corrected version).

Require: node graph G, start state s0 (attributes, start node, cleared set = $\emptyset$)
Ensure: optimum HP_{opt} and the set W of winning sets

```
1  s0 ← AUTO(s0)                                   ▷ take adjacent items and 0-HP fights
2  Q ← queue ordered by (decisions, -HP); push s0 into Q
3  best[cleared(s0)] ← HP(s0); HP_opt ← none; W ← ∅
4  while Q is not empty do
5      s ← pop(Q)
6      if HP(s) < best[cleared(s)] then continue     ▷ dominated after it was queued
7      if the boss is defeated in s then
8          HP_opt ← max(HP_opt, HP(s)); add cleared(s) to W; continue
9      if decisions(s) = 45 then continue
10     for each monster or door n adjacent to the hero in s do
11         if n is a door and s holds no key of its colour then continue
12         s' ← s with n cleared (exact damage subtracted or key used), hero on n
13         if HP(s') ≤ 0 then continue              ▷ lethal or unwinnable
14         s' ← AUTO(s')
15         if cleared(s') is new or HP(s') > best[cleared(s')] then
16             best[cleared(s')] ← HP(s'); push s' into Q
17 return HP_opt, W
```

Line 6 is the test that the published implementation gets wrong, as described below.

Figure 4.3 adds the solver states to the first decision of Figure 2.4. After the key at (5,1) has been taken, four choices are possible. Each leads to a new state, and the value shown is its state value, the best HP that any continuation reaches.

In the published implementation, the dominance comparison is made when a state is added to the queue. A second test, made when a state is taken from the queue, keeps whichever state with a given cleared set comes first, regardless of its HP. Because the queue orders states by the number of decisions before HP, a state with more HP that needs one more decision is discarded. On the complex floor, the published solver therefore reports an optimum of 22 HP. The exact solver, which also compares HP when a state is taken from the queue, finds 68 HP with ATK 16, DEF 20 and MDEF 20. The 21-decision route to 68 HP was executed action by action in the game engine; every action is legal and the boss is defeated with 68 HP remaining. The replay shows that the route is legal in the engine; that no route ends with more HP follows from the completeness of the search under the dominance test. On the simple floor and all fifteen generated floors, the exact

```
after the key at (5,1): HP 470, ATK 9, DEF 4, 1 yellow key
├─ fight Slime Lord (4,2)   → HP 356, 1 yellow key
│                              state value: none, the boss cannot be defeated
├─ fight Red Slime (5,3)    → HP 414, 1 yellow key
│                              state value: none
├─ fight Black Slime (6,4)  → HP 415, 1 yellow key
│                              state value: 13 HP                        ← optimal
└─ open yellow door (7,0)   → HP 470, no key
                              state value: none
```

Figure 4.3: The first decision of the simple floor in the solver's search tree. The state value of each branch comes from solving the resulting state exactly. Only the Black Slime keeps a win reachable. The Red Slime is not in the floor's only winning set: everything behind it is also reached past the Black Slime, so its 56 HP are wasted. The Slime Lord is in it, but costs 114 HP now against 22 HP where the optimal route fights it, far more than the floor's slack of 13 HP. The door spends the only key, which the optimal route needs at its third action for the door (4,7), behind which the next two keys lie; the observation tags this door as a dead end, and the engine replaces a choice of it (Listing 5.1, Section 5.6). The tag is recomputed at every decision; at turn 9, with two keys in hand, the same door is no longer a dead end.

solver and an independent Python reimplementation agree. All optima in this thesis come from the exact solver.

4.6.3 The Solver as Baseline and as Design Tool

The solver is not an agent that the language model competes with; it is the reference against which every episode is measured. It provides three things. First, the optimum HP_{opt} of each floor, against which the final HP of a successful episode is expressed (Section 5.9). Second, the number of distinct winning sets, which tells how many essentially different ways of winning a floor exist: the simple floor has one, the complex floor 32. Third, a per-decision analysis. From any state of an episode, each legal action is applied and the resulting state is written out as a new floor. The solver then determines whether that floor can still be won and with how much HP. Applied along the optimal route, this yields the critical and forced actions of Section 4.5.1. Applied to an agent's episode, it gives the *state value* after every decision, the highest final HP that can still be reached, and so locates the decision after which the boss can no longer be defeated (Section 6.4.5).

The same analysis was used to build the four hand-designed floors (Appendix B). Random layouts leave much slack: 13 of the 15 generated floors have an optimum of 150 to 250 HP, and the heuristic wins eight of them, so a model can win them without choosing well. The hand-designed floors were meant to be as tight as the simple floor. Each was built in a loop:

1. **Draft.** A layout is written as a text grid (walls, doors, keys, gems, potions,

monsters, boss, start), together with a monster table and the hero's starting attributes. Each draft is built around one difficulty mechanism observed on the simple and complex floors (the four mechanisms are listed in Appendix B). The drafts were written by the language model named in Section 4.5 from this description.

2. **Solve.** The draft is converted into a floor file and solved exactly. The solver reports the optimum, the states expanded, the number of winning sets and the number of optimal ones.
3. **Change the layout while there is more than one way to win.** If two different winning sets reach the same optimum, no change of the hero's attributes can separate them. The layout is changed instead, typically by removing an alternative entrance: on hand01, into the two wings behind the doors (4,8) and (6,8) (Figure B.2); on hand02, one of two identical monsters guarding the same room.
4. **Tune the start HP.** Once the layout admits a single winning set, the start HP decides how many sets are affordable. The solver is run for every start HP from the smallest winnable value upwards (Table 4.5). More HP pays for extra fights, so the number of winning sets grows quickly. The start HP is set to a value at which the winning set is still unique and a slack of a few HP remains at the optimum. On hand03 and hand04, the layout and the start HP were changed together.
5. **Check the reference agents.** The per-decision analysis gives the critical and forced actions, and the rule-based heuristic and random play (Section 4.7) are run. A floor is kept only if both fail.

Table 4.5: Start HP against the optimum and the number of distinct winning sets, from the exact solver. The first row of each floor is the smallest start HP at which the floor can be won; bold = the start HP used.

Floor	Start HP	HP_{opt}	Winning sets
hand01	395	9	1
	400	**14**	**1**
	405	19	1
	410	24	3
	420	34	5
	460	74	15
	700	314	48
hand02	50	1	1
	62	**13**	**1**
	75	26	4
	100	51	15

Floor	Start HP	HP_{opt}	Winning sets
	200	151	183
	400	351	702
hand03	641	1	1
	645	5	1
	649	**9**	**1**
	655	15	2
	670	30	10
	700	60	55
	800	160	711
hand04	92	1	1
	94	**3**	**1**
	142	51	82
	242	151	982
	492	401	2,684

The first versions of all four floors failed the uniqueness target, with 4, 2, 2 and 5 winning sets. The final versions have one winning set each, optima of 14, 13, 9 and 3 HP, and between 461 and 118,351 solver states. On each, the heuristic fails and random play never wins. The floors are therefore tight by construction and not representative of the game. Because they were built to defeat the rule-based heuristic, its failure on them is a design property, not a finding.

The solver's internals, the correction and the tooling are documented in Appendix B.

4.7 Reference Agents

Two simple agents bracket the LLM agents from below.

The **rule-based heuristic** chooses the first available action in a fixed priority order. It collects swords and shields, then gems, then keys, then potions. At a shop, it buys ATK when a reachable monster's DEF is at least the hero's ATK, and DEF otherwise. If no item or purchase is available, it takes the door or winnable fight whose cost, net of the items behind it, is lowest, preferring the cheapest of near-equal fights. When no such door or fight is left, it takes the stairs. It needs no model calls. It reaches MT8 of the tower but fails the simple floor (the boss survives, and the hero ends with 443 HP), the complex floor and all four hand-designed floors. Preferring the cheapest fight is the choice that the ordering structure of these floors punishes.

Random play chooses uniformly among the legal actions that do not kill the

hero. It wins none of the 500 games played on each hand-designed floor.

In the following chapter, the hybrid agent that acts in this environment is described: what the engine computes, what the language model sees, and how its choice is turned into an action.

5 Method

This chapter describes the hybrid agent and how it is evaluated. The agent divides the work between a deterministic engine and a language model (Figure 5.1). The engine computes everything that can be computed exactly: which targets are reachable, what each fight costs, and what each action unlocks. It writes these facts into the model's input as a numbered list of legal actions. The model reads the list, reasons, and answers with the number of one action, which the engine then executes. In the terms of Kambhampati et al. [36], the engine is a sound verifier. Unlike the verifiers of the LLM-Modulo framework, which check proposals afterwards, it acts before the model proposes anything, by restricting and annotating the options.

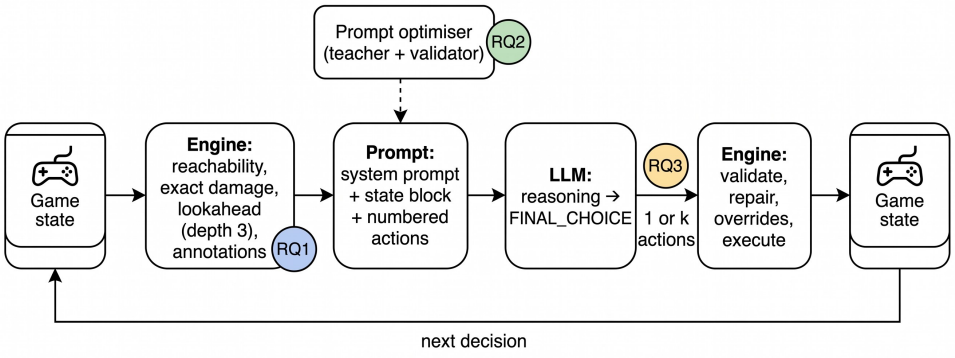


Figure 5.1: The hybrid agent (lookahead depth 3 on single floors, 2 on the tower). The research questions of Section 1.1 attach to the observation (RQ1), the system prompt (RQ2) and the planning horizon (RQ3).

Section 5.1 describes the engine and Section 5.2 the observation it produces. Section 5.3 describes how the model's answer becomes an action, and Section 5.4 the two system prompts. Section 5.5 describes the decision loop and the planning modes, and Section 5.6 the engine's assists and overrides. Section 5.7 presents the prompt optimisation loop, Section 5.8 the implementation, and Section 5.9 the measures used in Chapter 6.

5.1 The Engine

The engine is a Python reimplementation of the game rules of Chapter 4. It loads a floor from the game's data files or from a test map, simulates every action exactly, and exposes three services to the rest of the agent.

Reachability and action enumeration. A breadth-first search from the hero's position finds every tile the hero can walk to without passing a monster or a

closed door. Every monster, door, item, shop and staircase adjacent to that region becomes one candidate action. Fights that cannot be won ($a_h \leq d_m$) are not legal under Definition 4.1, but the engine lists them, marked as unwinnable, so that the model sees what blocks it. Choosing one triggers the mechanical override of Section 5.6. The agent therefore never chooses a path; it chooses a target, and the engine walks the shortest path to it.

Exact consequences. For every fight, the engine computes the HP cost with the full damage formula of Section 4.2, including the special abilities, and the HP that remains afterwards.

Lookahead. For every fight and door, the engine simulates the action and searches again to find what becomes reachable: items, keys, further doors, further monsters and the boss. If what becomes reachable is itself a monster or a door, the search continues behind it, up to a depth of three such *gateways* (two on the tower). The result is a tree of what the action unlocks, with the cost of every gateway on the way.

5.2 The Observation

The observation is the text the model receives at each decision. It consists of a state block and the numbered list of legal actions, each followed by its annotations. Listing 5.1 shows the observation at the first decision on the simple floor under the *final configuration* (temperature 0, lookahead depth 3, no conversation history, all annotations, mechanical assists only; Section 6.1.2). The engine has already collected the yellow key next to the start; the header counts completed turns, so the model's choice here is executed as turn 2, the numbering used in Chapter 6 (Section 5.5.1).

```
=== Floor: simple | Turn 1/300 ===
Hero: HP=470 (100.0% [OK]) ATK=9 DEF=4 MDEF=0 Gold=0 EXP=0
Keys: Y=1 B=0 R=0
Key budget: yellow 1 in hand + 4 on this floor = 5 for 4 closed yellow door(s)
Position: (5,1)
Enemies remaining: 14/14
Uncollected items still on floor: ATK+1@(1,0), Yellow Key@(0,1), ATK+1@(9,1),
↪  Red Potion (+75 HP)@(9,2)(+75HP), Blue Potion (+200 HP)@(10,2)(+200HP),
↪  [...]
DEF threshold tip: Collect +3 more DEF gems → Green Slime ×2 become FREE (saves
↪  48 HP; +4 DEF gems remain on map)
Goal: defeat Boss while keeping maximum HP. You do NOT need to kill all enemies
↪  — only fight enemies that grant stat gems, keys, or HP needed to reach and
↪  defeat the boss. There is NO shop on this floor. Boss at (5,10): ATK=20
↪  DEF=12 HP=50 | if fought now: UNWINNABLE — need ATK≥13 to deal damage (you
↪  have ATK=9) Use LOOKAHEAD TREE to find the cheapest path to the boss.
```

```
HP budget at max stats (all 4 ATK / 4 DEF gems collected): 1170 HP available (HP
↪  + 700 potions) vs 669 HP for the 2 monsters that block the way to the boss +
↪  the boss → margin +501 HP. 11 other monsters (258 HP at max stats) block
↪  nothing on the way to the boss; they are not in this budget — fight one only
↪  if its tree shows a stat gem or potion you need, and its cost then comes off
↪  this margin.
Recent actions this floor:
  T1: Collect Yellow Key at (5,1) (HP +0→470)

Available actions:
  (Note: lookahead = total item/stat value unlocked by this action
  ↪  [HP-equivalent]. [...])
  1. [fight] Slime Lord at (4,2) — monATK=10 monDEF=5 [FREE if DEF≥10, need +6
  ↪  more] — damage=114 (24.3%) hpAfter=356 [MODERATE]
  ↪  lookahead=0(raw=91-gw=114) effCost=None |
  ↪  min-cost=18@maxStats(ATK=13,DEF=8) save=96HP — collect stat gems first
 → next fight: ⚔ Slime Lord@(2,1) dmg=114(24.3%), FREE@DEF≥10(+6)
   → via [Slime Lord@(2,1) [dmg=114 (24%), FREE if DEF≥10 (+6 needed)]]: Red Gem
   ↪  (+1 ATK) [STAT], Yellow Key
 → stat gems behind it (+1 ATK): 228 HP of fighting to reach them, then they
 ↪  save 22 HP on the other fights listed now → net -206 HP
 → ORDER (this Slime Lord at (4,2)): take this fight BEFORE "Fight Red Slime at
 ↪  (5,3) dmg=56" — this fight then that route: 380 HP; that route then this
 ↪  fight: 392 HP (saves 12)
  2. [fight] Red Slime at (5,3) — monATK=11 monDEF=1 [FREE if DEF≥11, need +7
  ↪  more] — damage=56 (11.9%) hpAfter=414 [SAFE] lookahead=0(raw=220-gw=261)
  ↪  effCost=None | +75 HP potions behind it (collect any time) |
  ↪  min-cost=15@maxStats(ATK=13,DEF=8) save=41HP — collect stat gems first
  ↪  [ALTERNATIVE ROUTE — everything this opens is also opened by "Fight Black
  ↪  Slime at (6,4) dmg=55" (which opens more); taking both pays twice for the
  ↪  same access]
 → next fight: ⚔ Green Slime@(5,6) dmg=24(5.1%), FREE@DEF≥7(+3), ⚔ Black
 ↪  Slime@(3,7) dmg=55(11.7%), magic-atk, [...]
   [...]
 → stat gems behind it (+1 ATK +1 DEF): 317 HP of fighting to reach them, then
 ↪  they save 54 HP on the other fights listed now → net -263 HP
 → ORDER (this Red Slime at (5,3)): take this fight BEFORE "Fight Slime Lord at
 ↪  (4,2) dmg=114" — this fight then that route: 206 HP; that route then this
 ↪  fight: 277 HP (saves 71)
  3. [fight] Black Slime at (6,4) — monATK=5 monDEF=5 [magic-atk] [magic-atk:
  ↪  DEF useless — only ATK+MDEF reduce cost] — damage=55 (11.7%) hpAfter=415
  ↪  [SAFE] lookahead=29(raw=345-gw=237) effCost=26 | +225 HP potions behind it
  ↪  (collect any time)
 → next fight: ⚔ Green Slime@(5,6) dmg=24(5.1%), FREE@DEF≥7(+3)
 → unlocks: yellow door
   → via [Bat@(3,5) [...]]: Red Potion (+75 HP), Red Gem (+1 ATK) [STAT], Yellow
   ↪  Key
   → via [Green Slime@(5,6) [...]]: yellow door
   → via [yellowDoor@(7,5)]: Red Potion (+75 HP)
```

```
    → via [yellowDoor@(7,5) → Black Slime@(9,5) [...]]: Blue Gem (+1 DEF)
    ↪  [STAT], Red Potion (+75 HP)
      → via [Green Slime@(5,6) [...] → yellowDoor@(4,7) → Black Slime@(3,7)
      ↪  [...]]: Blue Gem (+1 DEF) [STAT], Yellow Key ×2
→ stat gems behind it (+1 ATK +2 DEF): 371 HP of fighting to reach them, then
↪  they save 75 HP on the other fights listed now → net -296 HP
→ ORDER (this Black Slime at (6,4)): take this fight BEFORE "Fight Slime Lord
↪  at (4,2) dmg=114" — this fight then that route: 175 HP; that route then
↪  this fight: 268 HP (saves 93)
→ keys: behind this fight none; it exposes yellow door (7,5) → none behind it
 4. [open_door] Yellow Door at (7,0) [DEAD END - DO NOT OPEN]

Choose the best action. After your analysis, end your response with:
FINAL_CHOICE: <number>
```

Listing 5.1: The observation at the first decision on the simple floor (final configuration). "[…]" marks shortened item lists, gateway annotations and the fixed explanatory note; the full text is reproduced in Appendix A.6 (Listing A.7). The static floor map (list of monsters, items and doors) is appended once to the system prompt. In `raw=345-gw=237`, the printed sum leaves out gateways in front of item names already listed; the lookahead value itself subtracts all four gateway fights (182 + 24 + 55 + 55 = 316 HP). `effCost=None` marks a fight whose lookahead value is 0 (Section 5.2.2).

The rest of this section explains each part of the observation. Each annotation is described by its purpose, its definition and the reason it is included. Table 5.1 summarises them.

Table 5.1: Annotations in the observation. Scope = where the annotation appears.

Annotation	Scope	Content
Damage and HP after	every fight	exact HP cost and remaining HP (Section 4.2)
Status tags	fights, doors	SAFE, MODERATE, RISKY, LETHAL, WINNABLE at ATK $\geq$ x, FREE if DEF $\geq$ x (a magic-attack tag instead on magic-attack fights), OPTIONAL, DEAD END, alternative route
Cost at maximum stats	winnable fights without magic attack	cost after every remaining stat item is collected
DEF threshold tip	state block	smallest DEF gain that makes a group of monsters free, and the HP it saves

Annotation	Scope	Content
Lookahead tree	fights, doors	what the action unlocks, up to three gateways deep, with each gateway's cost
Lookahead value and effective cost	fights, doors	HP-equivalent value of the unlocked stat items, keys and doors, net of gateway costs
Net stat-route line	fights, doors with stat items behind them	cost of reaching the stat items and what they save on the listed fights
Fight-order line	fights that a stat route makes cheaper	how much HP taking the fight now wastes and after which action, or which route to take it before
Key line	doors not tagged as dead ends, and fights that expose keys or doors	keys used, keys gained behind the action, doors exposed
Key budget	state block	keys held and on the floor versus closed doors, per colour
Boss line and HP budget	state block	whether the boss is winnable now, the ATK it needs, and the HP needed at maximum stats for the monsters that block the way

The annotations differ in kind. Damage, HP after, the status tags, the cost at maximum stats, the lookahead tree, the key lines and the key budget follow exactly from the rules in the current state. The net stat-route line, the fight-order line and the HP budget compare exact costs along routes that the engine specifies; the fight-order line also names an action to take first and is therefore advice. The lookahead value and the effective cost rest on hand-set weights (Section 5.2.2) and are heuristic estimates.

5.2.1 Exact Costs and Status Tags

Purpose: to remove every arithmetic step from the model's task. *Definition:* each fight lists its HP cost (`damage`), the share of current HP it takes and the HP that remains (`hpAfter`). A risk tag classifies a winnable fight by that share: `[SAFE]`

up to 15 %, `[MODERATE]` up to 30 % and `[RISKY]` above 30 %. `[LETHAL]` marks a fight that the hero cannot win or would not survive, and `[WINNABLE at ATK≥x]` gives the ATK at which such a fight becomes winnable. `FREE if DEF≥x` gives the DEF at which the fight costs nothing; magic-attack fights, whose cost DEF does not reduce, carry a tag saying so instead. The *cost at maximum stats* (`min-cost@maxStats`) is the fight's cost after every remaining stat item on the floor is collected; it is shown for fights without magic attack when it is at least 10 HP below the current cost. `[OPTIONAL]` marks a monster that blocks nothing needed for the goal, and `[DEAD END]` a door behind which nothing new can be reached with the keys held afterwards (Section 5.6). The *alternative-route tag* marks an action whose direct unlocks are all opened by another listed action that opens more; two actions that open exactly the same things are tagged as the same region. In the state block, the *DEF threshold tip* names the smallest DEF gain, within the DEF items left on the floor, that makes a group of monsters free, and the HP this saves. *Justification:* LLMs make planning errors that an exact computation does not [82]; the model's remaining task is to choose among options whose consequences are stated.

5.2.2 Lookahead Tree, Lookahead Value and Effective Cost

Purpose: to show what an action leads to, not only what it costs. *Definition:* the tree lists the items, keys, doors and monsters that become reachable, and, behind each gateway (`→ via [...]`), what becomes reachable after that gateway is also passed. Each unlocked item is valued in HP-equivalents: 70 per point of ATK, 90 per point of DEF, 45 per point of MDEF, 60, 90 and 120 for a yellow, blue and red key. Each unlocked door counts 80, 130 or 180 by colour, three quarters of that while more than one key of its colour is held; on the tower, a staircase counts 120 and a shop the stats it sells. The *net lookahead value* of an action is

$$L = \max\Big(0,\ \sum_i v_i\,\lambda_{j(i)} - \sum_g c_g\Big),$$

where v_i is the value of unlocked item i, $j(i)$ the number of gateways in front of it and c_g the HP cost of gateway g, each gateway counted once. The depth factor λ_j is 1 for items reached directly and 0.7, 0.5 and 0.35 behind one, two and three gateways. For a fight, the *effective cost* is its HP cost minus L; when L is 0 although gateways lie on the way, the line prints `effCost=None` instead, a placeholder that the models read as "not recommended yet" (Appendix E, incident 1). Potions are not included in L; they are listed separately ("+75 HP potions behind it"), because the HP of a potion is gained whenever it is collected and does not depend on the order of fights. *Justification:* without the tree, the model would have to

simulate reachability from coordinates, which the map-only condition indicates it does not do reliably (Qwen3.5-397B: precision 0.48, recall 0.68, $N = 2$; Table 6.4). The value lets the model compare a costly fight that unlocks a stat item with a cheap fight that unlocks nothing.

5.2.3 Net Stat-Route Line

Purpose: to state both the price and the benefit of reaching stat items. *Definition:* for an action with stat items behind it, the line gives the HP of fighting needed to reach them (C_{route}), the HP they would save on the other fights currently listed (S_{route}), and the difference $S_{\text{route}} - C_{\text{route}}$. In Listing 5.1, the Slime Lord (4,2) leads to one red gem for 228 HP of fighting, which saves 22 HP (net −206). The Black Slime (6,4) leads to one red and two blue gems for 371 HP, which save 75 HP (net −296). At this first decision every balance is negative, because the saving is counted only on the fights listed at the time. The line even ranks the Black Slime last, although it is the only action that keeps a win reachable (Figure 4.3); at later decisions, the line separates the routes. *Justification:* a saving shown without the cost of reaching it led the model to prefer expensive routes (Section 6.4.2); the line makes every such value two-sided (Figure 5.2).

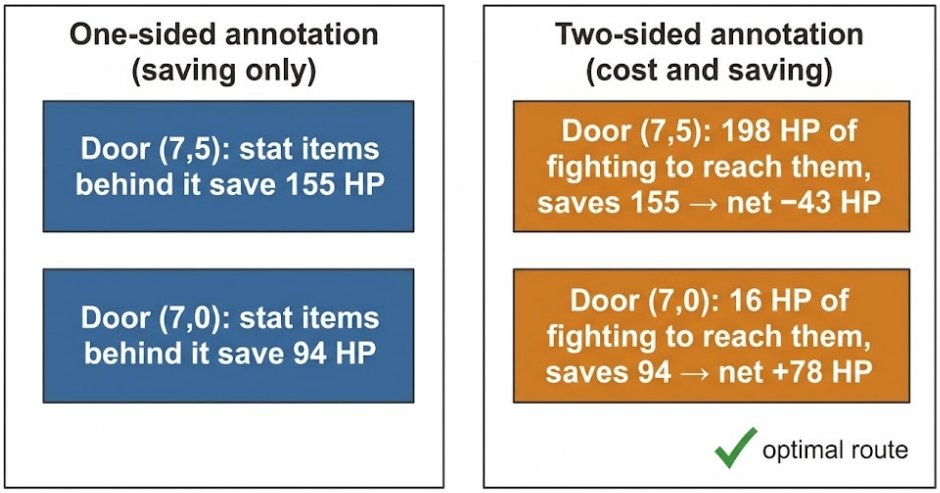


Figure 5.2: A one-sided and a two-sided annotation of the same two doors at turn 9 on the simple floor (Section 6.4.4). The hero stands at (1,10), at the far end of the lower-left room, with 336 HP, ATK 9, DEF 5 and two yellow keys. Shown only the saving, the larger number points to the door at (7,5); shown also the cost of reaching the stat items, the net balance points to the door at (7,0), which the optimal route takes next.

5.2.4 Fight-Order Line

Purpose: to make the effect of ordering visible. *Definition:* for a fight that a listed route would make cheaper, the line states how much HP taking the fight now wastes and which action to take first. It also states what that action costs and the combined cost of fight and route in both orders. A second, positive form ("take this fight before …") appears when fighting first is cheaper. In Listing 5.1, all three fights carry the positive form, and the lines of the Slime Lord and the Red Slime each recommend taking that fight before the other. At this decision, the order lines therefore do not separate the actions either. What singles out the Black Slime is its effective cost, the only numeric one in the list (26), together with the tags: the Red Slime is marked as an alternative route to the Black Slime, the door as a dead end, and the Slime Lord and the Red Slime with "collect stat gems first". In the re-run of Table 6.2, 19 of 20 episodes fought the Black Slime at this decision (Section 6.4). Neither form recommends a first action that costs more than 40 % of current HP (on the tower, the threshold of its rule 4, 60 %). *Justification:* on the simple floor, the order of one fight and one door at turn 9 separated wins from losses, and the model did not infer it from costs alone (Section 6.4.4).

5.2.5 Key Lines and Key Budget

Purpose: to make the key economy explicit. *Definition:* the key budget in the state block compares, per colour, the keys held and lying on the floor with the number of closed doors. Each door action not tagged as a dead end states which key it uses, how many keys the hero will hold after collecting what lies behind it, which keys lie behind it, and which further doors it exposes. If the door uses the last key of a colour without giving one back, a warning is added. A fight that directly exposes keys or doors carries a shorter line: the keys reachable after this fight alone and the doors it exposes. On the tower, where keys are shared by all floors, a second budget line counts the keys held and lying on all visited floors against all their closed doors. *Justification:* the lookahead value prices a key only by a fixed HP-equivalent and does not show which doors it will still be needed for. In the one failed episode of row 5 of Table 6.5, the model opened the door (7,5), which spends the only yellow key and gives none back, at turn 4 instead of the door (4,7) (Section 6.4.3).

5.2.6 Boss Line and HP Budget

Purpose: to tie each decision to the goal. *Definition:* the state block says whether the boss can be fought now and, if not, how much ATK is needed. The HP budget compares the HP available at maximum stats (current HP plus all potions) with

the HP needed at maximum stats for the monsters that block the way to the boss and for the boss itself. Monsters that block nothing are reported separately and are not included in the budget. The margin is therefore not the floor's slack: in Listing 5.1 it is 501 HP, but some of the monsters outside the budget guard stat items and keys that the win needs, and in the optimal order all fights together cost 1,157 HP, 488 more than the 669 HP that the budget counts, which leaves exactly the optimum of 13 HP. *Justification:* a budget over all monsters, including optional ones, declares a winnable floor unwinnable and licenses expensive fights (Section 6.4.2).

5.2.7 The Bare Observation

For the ablations of Chapter 6, the engine can also produce a *bare* observation. It contains the hero's attributes, the list of remaining items, the boss's attributes and the numbered action list with the exact cost of each fight, but no lookahead tree, no tags, no values and none of the lines of Sections 5.2.3 to 5.2.6. The engine still enumerates only legal, reachable actions. The bare observation isolates the contribution of the annotations from that of action enumeration and exact damage.

5.3 From Answer to Action

The model answers in free text, ending with the line `FINAL_CHOICE: <number>`. The engine extracts the number and checks that it denotes one of the listed actions. Because the model can only name a number from the list, it cannot refer to a target that does not exist or is unreachable; hallucinated targets cannot reach the game.

Two things can still go wrong: the answer can be truncated because the reasoning exhausted the token budget, or it can contain no valid number. An answer without a valid number goes to one repair call to the same model with reasoning disabled. This call uses constrained decoding [93]: the answer must be a JSON object with one integer field whose value is one of the listed action numbers. The repair call may only choose among actions that the mechanical overrides of Section 5.6 would not replace, so it cannot pick an unwinnable fight or a dead-end door. On single floors, a truncated answer is first sent again. On the tower, it goes to the repair call at once, because a truncated tower answer has used the whole reasoning budget and a repeated request would cost as much again. If neither the answer nor the repair yields a valid number, the request is sent again; after three consecutive failed requests at the same decision, the engine takes the rule-based heuristic's action. Every repair and fallback is logged and stored with the result (Appendix D); Chapter 6 states them where they occur.

5.4 The Two System Prompts

Two system prompts are used for all main results: one for single floors (2,526 characters) and one for the tower (6,035 characters). Both are reproduced verbatim in Appendix A.1 and A.2 (Listings A.1 and A.2). Appendix F lists the other system prompts used in Chapter 6. Each consists of three parts:

- **Combat rules.** The damage computation of Section 4.2 in five lines: who strikes first, the damage per strike, when a fight cannot be won, when it costs nothing, and how the special abilities change it. The prompt counts one monster strike per round. The engine counts one strike fewer per fight, because the hero strikes first and the monster dies before its last strike; the exact cost is printed with every fight.
- **How to read the observation.** The meaning of each annotation of Section 5.2, and the instruction that the lookahead tree is the only reliable map: if an action's tree does not list something, that action does not lead there.
- **Prompt rules.** The prompt rules of Table 4.2. Stat items come before the fights they make cheaper. Among fights that make progress, the one with the lowest effective cost comes first. No fight may cost more than 40 % of current HP (60 % on the tower) unless nothing else can make progress. No HP is spent on a fight that unlocks nothing needed. Keys are scarce, and a door is opened only when what lies behind it is needed. The tower prompt adds two rules. Rule 1: take the up-staircase to an unvisited floor before spending resources on the current one, unless a shop not yet used can be reached first. Rule 3 governs shops: never buy HP or level-ups; buy DEF before ATK, but enough ATK that the monsters on the way up cost at most 60 % of current HP; and check what an affordable purchase does to a fight before paying HP for it. Rule 3 also asks the model to spend EXP before gold, because gold also buys red keys, to buy where a point is cheapest, and to hand in the Cross only when nothing else can still raise ATK or DEF. The tower prompt also explains why ATK just above a monster's DEF is not enough. The fight becomes winnable, but the number of rounds, and with it the cost, falls only as ATK rises well above the monster's DEF.

The threshold of rule 4 is higher on the tower because, at 40 %, the rule kept the model from early fights towards weapons that the expert walkthrough takes; on single floors, 60 % lowered Qwen3.5-397B's result on the simple floor (Section 6.1.2). In the final configurations, the engine enforces neither threshold; the first and second tower observations enforced 40 % by a strategic override (Section 5.6).

The prompts deliberately contain no floor-specific knowledge: no coordinates, no lists of allowed or forbidden fights, no routes and no reference numbers from a walkthrough. The walkthrough principles S11 to S14 of Table 4.2 are therefore

never stated. This rule keeps the model's decisions attributable to its own reasoning over the observation, and for the prompt optimiser it is enforced by the validator of Section 5.7.

One further prompt, the *solver-route prompt*, is used as a labelled experimental condition on the complex floor (Appendix A.5, Listing A.5). It lists the route found by the published solver (the fights allowed and forbidden by position, the door order and the phases), together with a rule never to take a lethal fight. It is the plan-execution reference: it measures whether a model can carry out a given route, not whether it can find one. Because it contains compiled floor-specific knowledge, results obtained with it are reported separately as a compiled-knowledge condition (Chapter 6). The solver-route prompt and the exact-route prompt of Section 6.6 were run without the mechanical overrides of Section 5.6 and without the automatic 0-HP fights. The solver-route prompt was developed while the engine's overrides still included the strategic overrides of Section 5.6, which replaced steps of its listed route; its rule never to take a lethal fight took the place of the engine's lethal-fight override, and the exact-route prompt was run in the same configuration so that the two route conditions are comparable. Items and keys were still collected automatically, and an episode still ended as stuck when only lethal or unwinnable fights remained. A lethal fight chosen while another action existed was therefore executed; this is how the deaths in these conditions arise (Chapter 6). Results with the route prompts therefore differ from those with the single-floor prompt in the execution settings as well as in the prompt.

5.5 Decision Loop and Planning Modes

5.5.1 The Reactive Loop

In the reactive mode, the engine asks the model for one action at every decision. Each call contains the system prompt with the static floor map appended once, and a user message with the current observation. In the final configuration, the call contains no earlier turns. With the six previous observations and answers, which made up about three quarters of each request, Qwen3.6-35B won 1/5 on the simple floor, and without them 10/10 (Section 6.3). The engine asks the model only when there is a real choice: if exactly one action is legal, or if an assist of Section 5.6 applies, the engine acts without a model call. Every executed action, whether chosen by the model or taken by the engine, is one *turn*; turns are numbered from 1, and the header of the observation shows the number of turns completed. A turn at which the model is asked is a *decision*. On the simple floor, the engine collects the key at (5,1) at turn 1, so the first decision (Listing 5.1) is turn 2; on hand01, the first decision is turn 1. An episode ends when the goal is reached, when the

hero dies, when only lethal or unwinnable fights remain, or after 300 turns on a single floor; tower runs use a turn limit set per run (Appendix D).

5.5.2 Committed Planning Modes

To study the planning horizon (RQ3), the same loop can ask the model to commit to several actions per call. In the **commit-k** mode, the model answers with its choice for the current decision followed by a line `NEXT:` listing its next $k-1$ fights, doors or staircases by type and position (Appendix A.7; Listing A.8 shows the long-plan form). The engine executes the listed actions in order without calling the model, as long as each is still legal; items are collected in between. When the list is used up or an entry is no longer legal, the model is called again. In the **long-plan** mode, the model is asked to commit to as many further actions as it can, up to 59, and is called again in the same way. The long-plan instruction has two wordings, both reported in Chapter 6: the plan for the rest of the floor up to the goal (*whole-floor wording*) and as many actions as the model can (*max-steps wording*). In the **single-call** mode, the far end of the planning horizon (Section 2.4.1), the model is called once at the start of the episode (on the tower, once per floor) with the long-plan instruction. Turns not covered by this *single-call plan* are decided by the rule-based heuristic. These modes span the range from reactive control ($k = 1$) to a holistic plan for the whole floor, on the same engine, observation and model. Algorithm 5.1 shows the loop for all modes; the assists and overrides it refers to are described in Section 5.6.

5.5.3 The Tower Loop

On the tower, the loop is the same; what changes is the observation. The final tower observation, described below, is the result of several revisions, each made after tower runs had stopped at an obstacle that the observation did not state. Chapter 6 compares it with two earlier ones (Table 6.13). The *first tower observation* listed only the current floor's actions, the up-staircase and *return* actions to earlier floors, with each other floor summarised in one line. The *second tower observation* listed every reachable object on every visited floor, as the final one does, with a tower-wide key budget and the price of the way to a shop not yet reached; the planning modes were compared on the tower with this observation (Table 6.9). Both earlier observations ran with the 40 % override of Section 5.6, with rule 4 at 40 % and with the item values of another edition of the game (Section 4.5.2).

The rest of this section describes the final tower observation.

Action list. *Purpose:* to offer every object of the tower as one decision, without travel. *Definition:* the action list contains every reachable object on every visited

Algorithm 5.1: The decision loop. $k = 1$ is the reactive mode; in the single-call mode, the model is called only at the first decision (on the tower, at the first decision on each floor) and the heuristic replaces later calls. The two earlier tower observations (Section 5.5.3) also replaced fights costing more than 40 % of HP at line 17 (Section 5.6).

Require: floor F, system prompt P, model M, commitment k, turn limit T (300 on single floors)

```
1  s ← initial state of F; plan ← empty list
2  for t = 1 to T do
3      A ← ENUMERATE(s)                          ▷ legal actions with exact costs and lookahead (5.1)
4      if goal reached, hero dead, or A holds only lethal or unwinnable fights then
           break
5      if |A| = 1 then a ← the only action                                   ▷ no choice
6      else if an assist applies then a ← the assist action                  ▷ assisted turn (5.6)
7      else if plan is not empty and its next entry is legal in A then
8          a ← next entry of plan; remove it from plan
9      else
10         O ← OBSERVE(s, A)                     ▷ state block and annotated action list (5.2)
11         y ← M(P + static floor map, O)
12         (i, plan) ← PARSE(y)  ▷ FINAL_CHOICE and, if k > 1, NEXT: with k-1 entries
13         if i is not a listed number then   ▷ also a truncated y (sent again first on single
               floors, 5.3)
14             i ← REPAIR(M, A)     ▷ one constrained-decoding call without reasoning (5.3)
15         if no valid i after three failed requests then i ← HEURISTIC(s)   ▷ else the
               request is sent again
16         a ← action number i of A
17     if a came from the model or a plan and is a dead-end door, an unwinnable
           fight, or a lethal fight while a non-lethal action exists then
18         a ← HEURISTIC(s); plan ← empty list; log the override
19     s ← EXECUTE(s, a)                  ▷ walk the shortest path, apply exact consequences
20 return outcome of s
```

floor. The current floor's actions come first, then one block per other visited floor. Each block is formatted by that floor's own engine instance with the full observation of Section 5.2 (lookahead tree, effective cost, stat-route, order and key lines, status tags). Choosing an action on another floor walks there, which costs nothing in the game, and performs it in the same turn. A staircase appears only as the way up to a floor that has not been visited yet, and the engine takes it itself (Section 5.6). Purchases at every shop the hero has found are listed once and can be used from any floor. A purchase may carry a quantity: the answer may end with `QUANTITY: n` (at most 10), and the engine repeats the purchase without

asking again while it stays affordable. *Justification:* travel is not a decision in the game, and with return actions to earlier floors, models spent decisions shuttling between floors.

Cross-floor value. *Purpose:* to show what a gain in attributes is worth on all visited floors. *Definition:* stat items and shop purchases are annotated with their value across all visited floors: which fights they make winnable or free, and how much HP they save on the listed fights. For ATK purchases, the annotation also counts how many purchases make a fight winnable and how many make it cost at most the rule-4 share of current HP (60 %). *Justification:* a saving computed for one floor only would omit its scope, which makes it one-sided in the sense of Section 6.4.2.

Way-up line. *Purpose:* to tie each fight and purchase to the next floor. *Definition:* the *way-up* line gives the cheapest route on the highest visited floor to its up-staircase. A shortest-path search orders routes by unwinnable fights, then HP, doors and steps. The line gives the exact cost of every fight on the route at the hero's attributes and marks the fights that lie on every route. It states the total as a share of current HP and the ATK or DEF at which every fight on it costs at most the rule-4 share. It also states what one to three of each affordable ATK or DEF purchase does to these costs, with the purchases of one attribute combined across gold and experience shops. A fight that the hero can win but not survive is marked as lethal with its cost, not as unwinnable. A last line states whether the route is passable in path order. It assumes that before each paying fight, the gold and experience in hand, including what earlier fights on the route pay, are spent on the attributes that make that fight cheapest. It gives the HP left at the staircase or names the fight that cannot be survived. The action list tags the actions on the route; the line itself states computed costs and names no action to take. *Justification:* the strongest monster seen is the wrong reference for purchases, because early floors show monsters of later floors that are never on the way.

Purchase lines. *Purpose:* to price each purchase against the fights it changes. *Definition:* a fight that costs HP states what the purchases affordable now do to it: the fewest DEF purchases that make it free, or else those that make it cheapest. Each single-attribute purchase states its price per point. When another open shop sells the same attribute for less per point in the same currency, the line adds that shop's price and how much is still missing. For example, the gold shop on MT11 sells +20 for 100 gold (5 per point), against +4 for 25 gold on MT3. A shop that has not been reached yet is priced on the action that leads to it: what the hero's gold or experience would buy there, and the cost of every monster on the way, now and after one to three of the ATK purchases already available. *Justification:*

without these lines, a run paid HP for fights that a DEF purchase it could already afford would have made free.

Keys, treasures and scripted events. *Purpose:* to show the resources whose value is not an attribute. *Definition:* lines on scripted events (*event lines*) state what a character does, what it gives access to and the exact way there. If the item or guard of a scripted event lies behind doors, the line names those doors and the keys missing for them (for example, the Ice Pickaxe behind a blue door on MT12: "needs 1 blue key(s), you have 0"). The *seal line* of MT7, in front of the Cross, states the doors in front of the remaining guards and how many keys they need together. A *treasure* line in the state block lists up to five uncollected treasures on the visited floors (equipment, feathers, key boxes, the Cross, Holy Water and characters that hand such items over), each with the in-order verdict of the way there. A red door states how many red keys the hero holds and what the key seller charges for one. A door that is not on the way up and would leave fewer keys of its colour than the doors on the way up still need states so, with the coordinates of those doors (*key-reserve line*). A door or fight that makes 0-HP fights reachable states their number and the gold and experience they pay. The Cross states that its gain is a third of the attributes at the moment it is handed in. The MT15 trades become actions once the hero holds 500 experience or 500 gold. The engine never takes them, because they spend resources, and the treasure line lists them with their price and what the hero holds. *Justification:* tower runs stopped where a key spent dozens of turns earlier was needed (Section 6.7.3).

Vampire tag. *Purpose:* to state a damage rule that the DEF threshold does not describe. *Definition:* monsters whose damage is a share of the hero's current HP (the White Warriors take 25 %) carry the tag "[vampire: takes 25% of your current HP whatever your DEF, so it costs less when your HP is lower]" in place of the DEF threshold. The prompt's list of special abilities names the ability. *Justification:* the DEF-threshold form called these monsters "already immune", although each fight cost a quarter of the current HP.

Engine. The engine models the scripted events marked as modelled in Table B.4. The fight with the Red Demon King strengthens four kinds of monster for the rest of the game, as its script in the game data does.

5.5.4 Plan Memory

Without conversation history (Section 5.5.1), each request contains the system prompt and the current observation only. On the tower, this removes the reason for the previous decision along with the noise. A model that opened a door towards a weapon can decline the fight on the way to it at the next decision, because nothing in the request records the earlier intention.

Plan memory, part of the final tower observation, carries one line from one decision to the next. The request asks the model to end its answer with a line `PLAN:` before `FINAL_CHOICE`, written as a short to-do list: what it will do next and why, and what it is saving gold, experience or keys for. The engine stores the last such line and begins the next request with it, together with the turn and floor it was written on and the actions taken since then, including the assisted turns (Listing 5.2).

```
YOUR PLAN FROM YOUR PREVIOUS DECISION (turn 156, on MT5): Open yellow door at
↪  MT7 (8,11), fight two Skeleton
Captains to reach MT8, then scout MT8 before spending more resources.
Done since then (your choice first, then automatic steps): Open yellow door at
↪  (8,11).
Follow it if it still holds at the current state; change it if the state below
↪  shows a better option.
```

Listing 5.2: The plan-memory block at the start of a tower request.

If an answer contains no plan line, the previous plan is shown again with its own turn number, so its age is visible. Nothing in the plan is executed; it differs from the commitment of Section 5.5.2, where the engine carries out the listed actions without asking again.

The line is read from the visible answer and, if it is absent there, from the reasoning, because DeepSeek-V4-Flash and Qwen3.5-397B write it inside their reasoning. On single floors, plan memory lowered Qwen3.5-397B's result on the simple floor and is not used (Section 6.1.2).

5.6 Engine Assists and Overrides

Some actions need no judgement. The engine takes them itself and records each as an *assisted* turn with its reason:

- collecting a reachable stat item, key or potion, except, on the tower, the Holy Water of MT14, which doubles current HP and is left to the model;
- fighting a monster that costs 0 HP;
- on the tower, taking the up-staircase of a visited floor whose next floor has never been visited, from whichever floor the hero is on;
- on the tower, opening an ice gate, which needs no key and only frees the way.

The engine also checks every action the model chooses. A *mechanical override* replaces the choice with the rule-based heuristic's action when the chosen action is a dead-end door, a fight that cannot be won, or a lethal fight while a non-lethal action exists. If only lethal or unwinnable fights remain, the episode ends as stuck

instead of executing a fatal fight. A dead-end door is one behind which nothing new can be reached with the keys held afterwards, except monsters that cannot be won; a character that gives an item or triggers a scripted event counts as something new. Every override is logged with the model's choice, the executed action and the reason.

A *strategic override* replaces the model's choice by a judgement of the rule-based heuristic, for example whenever the heuristic judges another action much cheaper. In the development phase, the engine applied such overrides without logging them, and Chapter 6 shows that every early win on the simple floor contained one (Section 6.4.1). The two earlier tower observations (Section 5.5.3) also ran with an override that enforced the 40 % rule of the prompt. A fight costing more than 40 % of current HP whose items made no unwinnable fight winnable was replaced by the cheapest alternative that made progress; if the model chose it again at the same stats, it was executed. The final tower observation runs without this override, and rule 4 remains a rule of the prompt that the model may break. Strategic overrides are disabled in the final configuration. A single-floor result is attributed to the model only when no strategic override occurred; tower runs with the two earlier observations, which ran with the 40 % override, are reported as engine-assisted.

5.7 Prompt Optimisation

The prompt optimisation study (RQ2) uses a reflective loop derived from GEPA (Section 2.6). It keeps GEPA's central operator, the rewriting of a prompt by a *teacher*, a model that reads execution traces, and drops the parts that operate across many task instances. Each iteration has five steps:

1. **Evaluate.** The current prompt plays one episode on each of three training floors (the simple floor and two generated floors) at temperature 0. Its fitness is the mean goal score (Section 5.9.3).
2. **Build the reflection request.** A fixed template contains the scope of the task, a description of the observation the agent receives, the rules a new prompt must follow, the current prompt, one trace per training episode, and up to two earlier rewrites that scored worse. Each trace lists every decision with the chosen action, the hero's attributes afterwards, any override, and the end of the agent's reasoning, and ends with the outcome.
3. **Reflect.** The teacher writes a short diagnosis and one complete revised prompt.
4. **Validate.** The *validator*, a fixed check, rejects any candidate that contains coordinates, monster names, routes, lists of allowed fights or numbers of 100 or more. A rejected candidate is returned to the teacher once with the reasons.

5. **Select.** The candidate plays the same three floors. If its fitness is at least that of the best prompt so far, it becomes the new parent; otherwise the next reflection starts again from the best prompt.

If a candidate is rejected a second time, no candidate is produced in that iteration, and the parent is evaluated again. Algorithm 5.2 summarises the loop.

Algorithm 5.2: Reflective prompt optimisation with a validator (single lineage, elitist).

Require: seed prompt P0, training floors F1..F3, task model M, teacher T, validator V, 6 evaluations

P ← P0; P_{best} ← none; f_{best} ← $-\infty$; R ← empty list
for i = 1 to 6 **do**
  f ← mean goal score of M with P, one episode per floor, temperature 0
  if f ≥ f_{best} **then** P_{best}, f_{best} ← P, f
  else append (P, f) to R ▷ shown to the teacher as worse rewrites
  if i = 6 **then break**
  Tr ← traces of the episodes of P_{best} ▷ decisions, attributes, overrides, reasoning, outcome
  P' ← T(template, P_{best}, Tr, last two entries of R)
  if V(P') rejects **then** P' ← T(same request + the reasons for rejection)
  if V(P') accepts **then** P ← P' **else** P ← P_{best}
return P_{best}

A run consists of six evaluations, the seed prompt and five rewrites. The second study uses eight; the runs on the complex floor alone use two episodes per evaluation and six or fifteen evaluations (Section 6.5). In the terms of Section 2.6, the loop is a (1+1) evolutionary algorithm: the population is one prompt, mutation is the teacher's rewrite, fitness is the mean goal score, and selection is elitist. It is referred to as the *(1+1) loop*; it corresponds to Naive Prompt Optimization [7]. Table 5.2 lists which parts of GEPA it uses.

Table 5.2: The parts of GEPA (Section 2.6.1) and their use in the two loops of this thesis.

GEPA part	(1+1) loop	Pareto loop	Reason
Mutation by reflection on traces	yes	yes	the central operator; traces show the decisions that lost the game

GEPA part	(1+1) loop	Pareto loop	Reason
Acceptance on the same batch	yes (all training floors)	no (every valid candidate joins the population)	with three floors, the batch is the whole training set
Pool of candidates	no (one prompt)	yes	
Parent selection from the Pareto frontier over instances	no	yes	
Merge of two frontier candidates	no	once per run	
Separate validation set during the search	no	no	one episode costs up to several dozen model calls; the held-out floors are used only after the search
Number of evaluations	6 (8 in the second study)	8 candidates	one episode costs up to several dozen model calls

The *Pareto loop* adds GEPA's selection to the same teacher, validator, trace and harness. Each candidate is scored on every training floor separately. The parent of the next rewrite is drawn from the candidates that are best on at least one floor, with a weight equal to the number of such floors, and the teacher sees the parent's traces. At the last slot, the teacher instead merges the two frontier candidates with the highest mean score, given both prompts and their per-floor outcomes. The loop is implemented in the framework rather than taken from the GEPA library, because the library's instances are not the engine's floors and its reflection step does not apply the validator. The two loops therefore differ only in how parents are selected and accepted, plus one merge (Table 5.2), so that a difference between them can be attributed to selection.

One further run removes the validator. On the complex floor alone, the (1+1) loop starts from the single-floor prompt, and the teacher is told that the prompt will be used on that floor only and may contain positions, orders and conditions learned from the traces. The teacher sees the traces and the end state of each episode, never the solver's route. The result, the *evolved-route prompt*, is floor-specific knowledge like the solver-route prompt and is reported as a labelled condition; it tests whether evolution can compile from experience what the solver found by search.

From a methodological perspective, the restriction to small populations and few evaluations follows from the cost of the task. One episode takes up to several dozen model calls, so a large population evaluated on many instances is not affordable. The loop runs through the same engine, assists and measures as every other result, so that evolved and hand-written prompts are compared under identical conditions. The template is reproduced in Appendix A.8 (Listing A.9), and the study design in Section 6.5.

5.8 Implementation

The agent, the engine, the solver interface and the evaluation harness are implemented in Python 3.12. All models are accessed through the OpenAI-compatible interface of the GWDG SAIA service [15]. A repeated-episode harness executes each configuration N times, records the complete configuration with each result, and writes one log per episode with every observation, answer, reasoning text, override and assisted turn.

The code of the framework, the platform and the analysis scripts is available at `https://github.com/thetobysiu/mt-ai-lab`.

The main models were selected by four criteria: available on the service for the whole study, open weights, a reasoning mode, and a decision time that allows repeated episodes. Three models without a reasoning mode in these experiments, Mistral Medium 3.5, Llama-3.1-8B and DeepSeek-V4-Flash (no thinking), and the slow Qwen3.8-27B were added for specific contrasts (Section 6.1.1). From a methodological perspective, this thesis restricts itself to the models served by one inference service. This choice reflects practical constraints, as each episode needs up to several dozen calls to a large model. It also places the models' availability outside the author's control: three models used in early experiments were removed from the service during the study (Chapter 7). Section 6.1 lists the models and their settings.

The experiments are run and inspected through a web platform built for this thesis, the *Magic Tower AI Lab*. It shows every running episode live on the game map, drawn either as annotated cells or with the original game's tiles, and replays stored episodes with the model's reasoning at each decision (Figure 5.3). Every experiment of Chapter 6 is listed in a catalogue with its exact configuration and its reported result. A reader can enter an API key for an OpenAI-compatible service, which is kept in memory only, and re-run any configuration under a new label; stored results are never overwritten. The same catalogue is available from the command line. A further page shows every prompt optimisation run iteration by iteration, with the prompt changes, the teacher's diagnosis and the validator's

verdict. An automated test suite pins the engine's rules, the four baseline results of the heuristic and the exact text of the observation at a reference decision. Any change to the observation is therefore detected before it can affect an experiment. The platform is described in Appendix C.

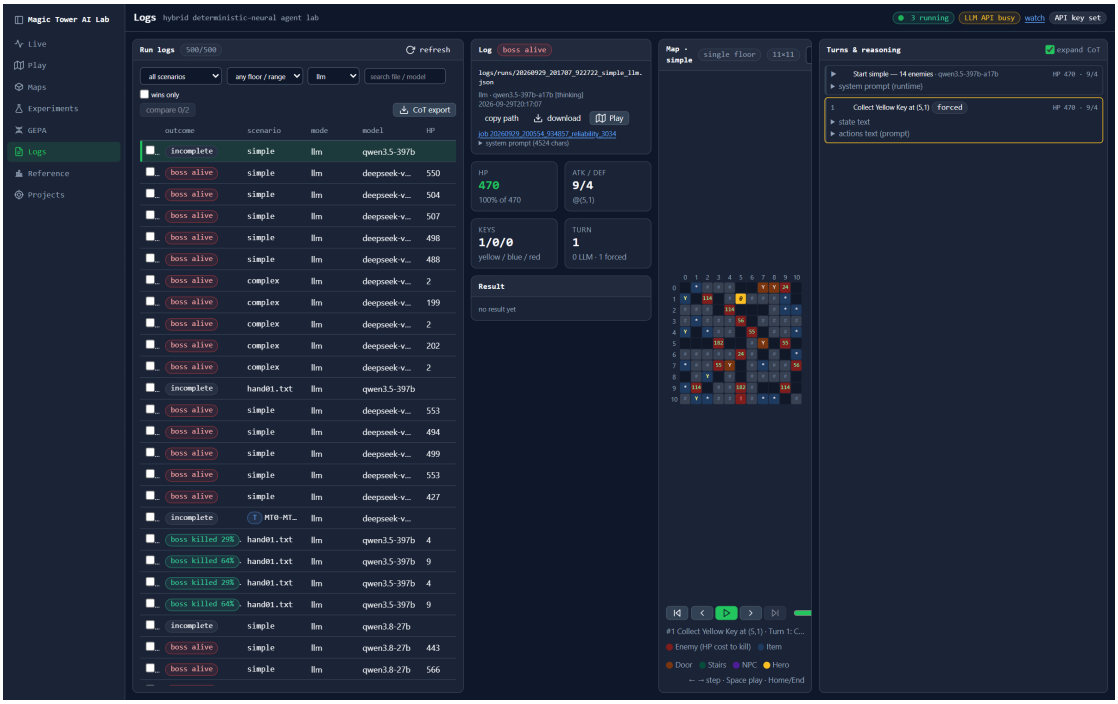


Figure 5.3: The platform's log page: stored run logs (left), the selected log's summary and hero state (centre left), the simple floor at turn 1 (centre right), and turn cards whose reasoning and action text expand on demand (right).

5.9 Measures

5.9.1 Success Rate

An episode is successful if the floor's goal is reached with the hero alive (Definition 4.1). For a configuration run N times with k successes, the success rate is $\hat{p} = k/N$, reported as `k/N` with the Wilson 95 % interval [94]:

$$\frac{\hat{p} + \frac{z^2}{2N} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}, \qquad z = 1.96.$$

The Wilson interval is preferred to the normal approximation for small N and for

rates near 0 or 1 [4]. At $N = 10$ its width is about ± 0.25, so differences in success rate smaller than about 0.3 are not resolved.

5.9.2 HP Relative to the Optimum and State Value

Among successful episodes, quality is measured as $\mathrm{HP}_{\mathrm{final}}/\mathrm{HP}_{\mathrm{opt}}$, the final HP relative to the exact optimum of the floor (Section 4.6). The ratio lies in $(0, 1]$, and 1 means the agent matched the solver. It is undefined for failed episodes and is reported as the mean over successes.

To locate where a failed episode was lost, the exact solver is also applied to the states along the episode. The *state value* of a state is the highest final HP that the exact solver can still reach from it; it is none when the boss can no longer be defeated. A decision lowers the state value by the final HP it costs, and the decision after which the value becomes none is the decision that lost the episode (Section 6.4.5, Figure 6.2).

5.9.3 Goal Score

The prompt optimiser needs a single number per episode. It uses

$$S_{\mathrm{goal}} = \begin{cases} 0.5 + 0.5 \cdot \min\bigl(1, \mathrm{HP}_{\mathrm{final}}/\mathrm{HP}_{\mathrm{opt}}\bigr) & \text{success,} \\ 0.35 \cdot C & \text{failure with the hero alive,} \\ 0.15 \cdot C & \text{hero dead,} \end{cases}$$

where C is the share of the floor's monsters defeated. The success range lies above both failure ranges, so any success scores higher than any failure. The two failure ranges, $[0, 0.35]$ and $[0, 0.15]$, overlap; a surviving failure outranks a death only at equal $C > 0$.

5.9.4 Paired Comparisons

When two prompts are evaluated on the same set of floors, they are compared with the exact sign test over the discordant pairs, the floor and repetition pairs in which exactly one of the two prompts wins. With n discordant pairs, of which b favour the first prompt, the one-sided p-value is $\sum_{i=b}^{n} \binom{n}{i} 2^{-n}$. The test is one-sided for the pre-registered confirmation (Section 6.5.2) and two-sided, twice the smaller tail, for the second study (Table 6.8).

5.9.5 Plan Check, Tower Progress and Cost

In the committed planning modes, every stored plan is checked by simulating it on the engine from the state in which it was written, with the assisted turns of the loop in between. A step *breaks* when its target is not reachable, needs a key the hero does not hold, or is a fight that cannot be won or survived at that point. *New decisions per plan* is the mean number of entries per plan that name neither an item nor an action already taken, such as a repeat of the chosen action. A *clean* plan is one in which no step breaks (Table 6.10).

Tower runs are reported by the highest floor reached, whether the hero is alive, and the final HP, ATK and DEF. A run that defeats the Nether Demon King on MT21 reaches the normal ending (Section 4.5.3) and counts as won.

The cost of a configuration is reported as model calls per episode (Table 6.9) and, where it matters, time per call. Steps executed from a committed plan are not calls.

6 Experiments

This chapter presents the empirical evaluation of the hybrid agent. The main objectives are to measure how often the agent wins each floor under the final configuration and which components of the observation are responsible. Further objectives are to test whether prompt optimisation adds to this result and how the planning horizon affects success and cost. Section 6.1 describes the setup shared by all experiments. Section 6.2 reports the main results, Section 6.3 the ablations and Section 6.4 the error analysis. Sections 6.5 and 6.6 address the second and third research questions. Section 6.7 reports results on the hand-designed floors, the generated floors and the tower. All experiments use the measures defined in Section 5.9.

6.1 Experimental Setup and Configurations

All experiments share one inference service, one evaluation harness and one fixed configuration. Most experiments change one factor of this configuration, and the tables mark those that change more.

6.1.1 Models and Inference Service

All models are accessed through the GWDG SAIA service [15]. They are named in this thesis by family, version and total size; the endpoint identifier is given in the second column of Table 6.1.

Table 6.1: Models used in the experiments. B = billion parameters; Reasoning cap = maximum number of output tokens per call, including the reasoning.

Name in this thesis	Endpoint model	Type	Reasoning mode	Reasoning cap (tokens)
Qwen3.5-397B	qwen3.5-397b-a17b	mixture of experts, 17 B active	thinking	32,768 (tower 8,192)
Qwen3.6-35B	qwen3.6-35b-a3b	mixture of experts, 3 B active	thinking	16,384 (map-only condition 32,768; two early rows of the repair sequence, Section 6.4.3: 8,192; tower 8,192)
Qwen3.8-27B	qwen3.8-27b	dense	thinking	32,768
DeepSeek-V4-Flash	deepseek-v4-flash-0731	mixture of experts, 284 B, 13 B active	thinking	16,384 or 32,768 (reactive, stated per cell), 32,768 (planning modes); tower 8,192 (reactive), 16,384 (planning modes)

Name in this thesis	Endpoint model	Type	Reasoning mode	Reasoning cap (tokens)
DeepSeek-V4-Flash (no thinking)	deepseek-v4-flash-0731	mixture of experts, 284 B, 13 B active	none	not applicable
Mistral Medium 3.5	mistral-medium-3.5-128b	128 B, architecture not stated by the service	none (the service accepts only no reasoning or a high-effort setting that took 430 s for a one-line question)	not applicable
Llama-3.1-8B	meta-llama-3.1-8b-instruct	dense	none	not applicable

DeepSeek-V4-Flash (no thinking) and Llama-3.1-8B are used only in the prompt optimisation study (Section 6.5), where weaker models leave room for improvement. Mistral Medium 3.5 was added for the re-run of Table 6.2 as a model of a different family without a reasoning channel. Qwen3.8-27B is not in the main table, because its episodes took 55 to 113 minutes each in preliminary episodes. It appears in the planning study (Section 6.6).

6.1.2 Final Configuration

Unless an experiment states otherwise, every episode uses the following settings:

- temperature 0 (Section 6.3 repeats the cells that carry the main conclusions at 0.6 and 1.0; the planning cells of Qwen3.8-27B use 0.6);
- lookahead depth 3 on single floors and 2 on the tower;
- no conversation history: each decision sees only the current observation;
- the single-floor prompt or the tower prompt of Section 5.4;
- all annotations of Section 5.2;
- answers ending with `FINAL_CHOICE: <number>`, with one constrained-decoding repair call for truncated or unparseable answers (Section 5.3);
- engine assists limited to automatic collection of items and keys, automatic fights that cost 0 HP and, on the tower, the climb to a floor not yet visited, each logged per turn;
- only the mechanical overrides of Section 5.6, each logged per turn.

Two settings of the tower configuration, the rule 4 threshold of 60 % and plan memory (Section 5.5.4), are not used on single floors. Both were tested on the simple floor and on the complex floor with both prompts. On the simple floor, each lowered Qwen3.5-397B from 5/5 to 2/5 [0.12, 0.77]; DeepSeek-V4-Flash went from 1/5 to 0/5 and 2/5. On the complex floor, the single-floor prompt stayed at 0/5 for both models, and with the solver-route prompt the results moved in both directions (Qwen3.5-397B from 3/5 to 5/5 at 60 % and to 2/5 with plan memory;

DeepSeek-V4-Flash from 4/5 to 3/5 and 4/5).

6.1.3 Episodes and Repetitions

Each configuration is run N times from the same initial state. The cells of the main results table use $N = 5$ (ten for Qwen3.6-35B with the solver-route prompt). The final-configuration cells of the ablations and of the repair sequence use $N = 10$, as do the intermediate repair rows and some reactive arms of the planning study. Most other comparisons use $N = 2$ to 5 (three for the slow Qwen3.8-27B), and tower cells have one to five runs. Success rates are reported as k/N with Wilson 95 % intervals at first mention. At most two configurations were run concurrently on one API key, because four parallel experiments had slowed every call of Qwen3.5-397B from about 20 s to over 300 s.

6.2 Main Results

Table 6.2 reports the success rate and the final HP relative to the exact optimum for each agent and floor.

Table 6.2: Success rate k/N and final HP relative to the exact optimum (HP/opt, mean over successful episodes, in parentheses) per floor. Final configuration unless marked. Simple = simple floor; Complex, single-floor prompt = complex floor with the single-floor prompt; Complex, solver-route prompt = complex floor with the solver-route prompt, which lists the published solver's route (compiled-knowledge condition and plan-execution reference, Section 5.4; note [d]); hand01–hand04 = hand-designed floors; Tower = highest floor reached from MT0 in the best of three runs with the final tower observation (Section 6.7.3; all runs in Table 6.13). DeepSeek-V4-Flash ran on the single floors with a reasoning cap of 32,768 tokens. Earlier episodes of the same configuration on the simple floor (Table 6.3, first row): Qwen3.5-397B 10/10, Qwen3.6-35B 7/10. Bold = best per column (equal rates: larger N); "—" = not run (note [b]).

Agent	Simple	Complex, single-floor prompt	Complex, solver-route prompt [d]	hand01	hand02	hand03	hand04	Tower
Exact optimum (HP)	13	68	68	14	13	9	3	— [c]
Rule-based heuristic	0/1	0/1	—	0/1	0/1	0/1	0/1	MT8
Random play	—	—	—	0/500	0/500	0/500	0/500	—
Qwen3.5-397B	**5/5** (0.15)	0/5	3/5 (0.25) †1	**5/5** (0.50)	4/5 (0.48)	0/5	0/5	**MT18**
DeepSeek-V4-Flash	1/5 (0.15)	0/5	**4/5** (0.26)	4/5 (0.66)	**5/5** (0.28)	0/5	0/5	MT17
Qwen3.6-35B	1/5 (0.15)	0/5	4/10 (0.20) †3	4/5 (0.64)	3/5 (0.31)	0/5	0/5	MT16
Mistral Medium 3.5 [a]	0/5	0/5	0/5 †4	**5/5** (0.64)	**5/5** (0.86)	0/5	0/5	MT8

[a] No reasoning channel (Section 6.1.1). [b] "—" = not run: random play was evaluated only on the hand-designed floors, and the heuristic reads no prompt. [c] No exact optimum is computed for the tower; the normal ending of the game is on MT21 (Section 4.5.3). [d] Run without the mechanical overrides and the automatic 0-HP fights (Section 5.4); a lethal fight chosen while another action existed was executed. † Number of episodes in which the hero died.

All LLM cells of Table 6.2 were re-run on the final version of the code. Each cell has five episodes (ten for Qwen3.6-35B with the solver-route prompt), and each tower cell three runs. The rule-based heuristic acts on the same engine output without a language model. It wins none of the six fixed floors of Table 6.2, against eight of the fifteen generated floors (Section 6.7.2). It ends the simple floor alive at 443 HP with the boss untouched, fails the complex floor and all four hand-designed floors, and reaches MT8 of the tower. Random play wins none of 500 games on any hand-designed floor.

Simple floor. Positions are given as (x, y) with the origin at the top left, as in Figure 4.1; Appendix B.5 gives all maps. Qwen3.5-397B wins 5/5, every episode at 2 HP (HP/opt 0.15). Every winning episode of the final configuration, of every model, follows the same sequence of actions, the *winning trajectory*. It is the optimal route of Table 4.4 with one change: it fights the Slime Lord (1,9) at turn 15, before the Bat (3,5), which costs 11 HP and leaves 2 (Section 6.4). The ablations of Table 6.3 were run on an earlier version of the code, and their final-configuration rows are called the earlier episodes below. Together with the earlier episodes of the same configuration (10/10, Table 6.3), Qwen3.5-397B wins 15 of 15 episodes on this floor [0.80, 1.00]. DeepSeek-V4-Flash and Qwen3.6-35B win 1/5 each [0.04, 0.62], and Mistral Medium 3.5 0/5. All five failures of Mistral Medium 3.5 and one of DeepSeek-V4-Flash leave the winning trajectory at turn 18 by fighting the Slime Lord (4,2). The other three failures of DeepSeek-V4-Flash fight the Slime Lord (4,2) at turn 2 or the Bat (3,5) at turn 4, or, in one episode that kept the state value at 13 HP up to turn 25, the two Slime Lords of the top-left corner before the Slime Lord (1,9) at turns 28 and 29. For Qwen3.6-35B, the earlier episodes of the same configuration won 7/10 [0.40, 0.89] (Table 6.3). In the re-run, all five episodes of Qwen3.6-35B are identical up to turn 22. At turn 22, on the same input, one episode opens the yellow door (7,5) and four fight the Slime Lord (4,2), as the three failures of the earlier episodes did (Section 6.4 explains why this loses). Section 7.2 discusses this difference. Every kill in Table 6.2 on this floor ends at 2 HP. Among the simple-floor episodes of Tables 6.3 and 6.5, two kills end higher. One ends at 10 HP (Table 6.5, row 7). One ends at 13 HP, the exact optimum (Qwen3.6-35B at lookahead depth 2, Table 6.3).

Complex floor. With the single-floor prompt, no model wins any of 20 episodes

(0/20 [0.00, 0.16]). All end alive with only lethal or unwinnable fights left. Section 6.4.5 locates the decision that lost each of them. With the solver-route prompt, three of four models win: DeepSeek-V4-Flash 4/5, Qwen3.5-397B 3/5 and Qwen3.6-35B 4/10 [0.17, 0.69]. Their kills reach 0.20 to 0.26 of the optimum on average, and every kill ends at 22 HP or below, the value of the listed route. Mistral Medium 3.5 wins 0/5 and dies in four of its five episodes. Among the conditions of Table 6.2, only this prompt has deaths, eight in total: four of Mistral Medium 3.5, three of Qwen3.6-35B and one of Qwen3.5-397B. They occur because this condition runs without the mechanical overrides, which would replace a lethal choice (Section 5.4). Six of the eight die in the boss fight, one at the Big Bat (9,1) and one at the Black Slime (0,4). The six failed episodes of Qwen3.6-35B leave the trajectory of its winning episodes at turn 12 (three episodes), turn 21 (one) or turn 37 (two); the decision that loses each of them comes at or after that turn (Section 6.4.5).

Hand-designed floors. The floors hand01 and hand02 are won by every model, and hand03 and hand04 by none in any of 20 episodes each (0/20 [0.00, 0.16]); every episode on hand03 and hand04 ends alive. On hand01, Qwen3.5-397B and Mistral Medium 3.5 win 5/5, DeepSeek-V4-Flash and Qwen3.6-35B 4/5. On hand02, DeepSeek-V4-Flash and Mistral Medium 3.5 win 5/5, Qwen3.5-397B 4/5 and Qwen3.6-35B 3/5. Mistral Medium 3.5 wins nothing on the simple and complex floors. On hand02, it ends four of its five kills at 13 HP, the exact optimum (HP/opt 0.86), the highest ratio in Table 6.2.

Tower. The tower has 24 floors, MT0 to MT23, and the game's normal ending is the defeat of the Nether Demon King on MT21. With the final tower observation, the highest floor reached in the best of three runs is MT18 for Qwen3.5-397B, MT17 for DeepSeek-V4-Flash and MT16 for Qwen3.6-35B. Mistral Medium 3.5 reaches MT8, the floor where the heuristic stops. In their other two runs, Qwen3.5-397B and Qwen3.6-35B stop at MT8, Mistral Medium 3.5 at MT4, and DeepSeek-V4-Flash at MT8 and MT17 (Table 6.13). Section 6.7.3 reports where and why each run stopped. A stored run of Qwen3.5-397B, resumed on MT18 with blue keys for sale as in the original game, reached the normal ending.

6.3 Ablations

Table 6.3 changes one factor of the final configuration at a time on the simple floor and repeats the main cells at temperatures 0.6 and 1.0.

Table 6.3: Ablations and temperature repetitions; k/N successes. Upper rows: earlier episodes of the final configuration on the simple floor (the re-run is in Table 6.2), one factor changed per row. Temperature rows: the cells of Table 6.2 at temperatures 0 · 0.6 · 1.0. "—" = not run.

Factor	Setting	Qwen3.5-397B	Qwen3.6-35B	DeepSeek-V4-Flash
Lookahead depth	3 (final)	10/10	7/10	—
	2	4/5	7/10	—
	1	0/3	0/5	—
	0	0/3	0/5	—
Observation	full annotations (final)	10/10	7/10	—
	bare observation [b]	0/3	0/3	—
	map only [b]	0/2	0/2	—
Conversation history	none, same version as the next row [a]	9/10	10/10	—
	six previous decisions [a]	10/10	1/5	—
Temperature 0 · 0.6 · 1.0	Simple	5/5 · 5/5 · 5/5	—	1/5 · 1/5 · 2/5
	Complex, single-floor prompt	0/5 · 0/5 · 0/5	—	0/5 · 0/5 · 0/5
	Complex, solver-route prompt [d]	3/5 †1 · 3/5 · 4/5	—	4/5 · 4/5 · 3/5 †1

[a] Measured on the annotation versions of Table 6.5, rows 6 and 7, before the two-sided order line; with the final annotations and no history, the models win 10/10 and 7/10 (first row). [b] Earlier version of the code: the map-only condition on the version of Table 6.5, row 7, the bare observation on that of row 9. [d] Run without the mechanical overrides and the automatic 0-HP fights (Section 5.4). † Number of episodes in which the hero died.

Lookahead depth. Removing one level of lookahead lowers Qwen3.5-397B from 10/10 to 4/5 [0.38, 0.96], and removing two or three levels to 0/3 [0.00, 0.56]. Qwen3.6-35B wins 7/10 at depths 3 and 2 and 0/5 [0.00, 0.43] at depths 1 and 0. The turn of the first departure from the winning trajectory (Section 6.4) depends

on the depth. At depth 1, all eight episodes depart by opening the yellow door (7,5), three at turn 3 and five at turn 9, which lowers the state value from 13 to 4 or 10 HP; the floor is lost at a later decision. At depth 0, five of eight episodes fight the Slime Lord at (1,9) at turn 9, before the stat item that makes it cheaper, and the other three fight the Bat (3,5) at turn 4; both choices make the boss unwinnable. At depth 2, the four failed episodes depart at turn 2 (the Red Slime (5,3)), at turn 4 (the Bat (3,5)) and, in two episodes, at turn 22 (the Slime Lord (4,2) instead of the door (7,5)). On hand01 and hand02, DeepSeek-V4-Flash wins 9 of 10 episodes at depth 1 [0.60, 0.98] (5/5 and 4/5), as at depth 3 (4/5 and 5/5). Its depth-1 kills end at 5 to 10 HP on hand01 and at 3 to 4 HP on hand02.

Observation. Without annotations, both models lose every episode (0/3 [0.00, 0.56] each), although the engine still enumerates only legal actions with exact costs. With the rules-only prompt of Section 6.5 instead of the single-floor prompt, Qwen3.5-397B also loses 0/3, ending alive at 219 to 481 HP. The same holds beyond the simple floor. With the bare observation, DeepSeek-V4-Flash loses all five episodes on hand01 and all five on hand02, alive at 114 to 153 and 42 to 69 HP. With the annotations, it wins 9 of 10 [0.60, 0.98]. Without the enumerated action list, Qwen3.5-397B must list its options from the grid before choosing (Table 6.4) and loses 0/2 [0.00, 0.66]. Half of the options it lists are not possible (66 of 128). Of these, 46 are real monsters, doors or items behind a closed door or an undefeated monster. It misses a third of the possible options, and a quarter of its choices are illegal. Qwen3.6-35B did not complete this analysis within 32,768 tokens on 22 of 30 decisions.

Table 6.4: Map-only condition on the simple floor ($N = 2$ per model). Options listed / legal = options listed by the model / legal options available, summed over all decisions; Precision = share of listed options that are legal; Recall = share of legal options that are listed; Impossible options = listed options that name no entity at that position / that name an entity the hero cannot reach; Illegal choices = share of decisions whose final choice is illegal or missing; Truncated decisions = answers cut off at the reasoning cap.

Model	Wins	Options listed / legal	Precision	Recall	Impossible options (no such entity / unreachable)	Illegal choices	Truncated decisions
Qwen3.5-397B	0/2	128 / 91	0.48	0.68	20 / 46	25 %	0 of 28
Qwen3.6-35B	0/2	15 / 120	0.20	0.03	11 / 1	93 %	22 of 30

Conversation history. The six previous decisions (observations and answers) made up about three quarters of every request. Removing them left Qwen3.5-397B at 9/10 [0.60, 0.98], against 10/10 with history on the same version; its one failure departed from the winning trajectory at turn 18, fought both Slime

Lords (4,2) and (2,1) at 60 HP each instead of 22, and ended 23 HP short of the boss. Removing them raised Qwen3.6-35B from 1/5 [0.04, 0.62] to 10/10 [0.72, 1.00]. On that version, Qwen3.6-35B without history wins all ten episodes along the winning trajectory, at 16 s per call. Qwen3.5-397B reasons about four times longer per decision without history (median 7,100 against 1,700 characters).

Temperature. All main cells use temperature 0, below the values that the model cards recommend for thinking mode (0.6 for Qwen3.5-397B, 1.0 for DeepSeek-V4-Flash). The cells that carry the main conclusions were therefore repeated at 0.6 and 1.0, with $N = 5$ (temperature rows of Table 6.3). No success rate at 0.6 or 1.0 differs from the rate at 0 by more than one episode in five.

6.4 Error Analysis

The error analysis concentrates on the simple floor (Sections 6.4.1 to 6.4.4); Section 6.4.5 locates the losing decisions on the complex floor with exact state values. The slack of the simple floor is small (Section 4.5.1). The hero starts with 470 HP and the potions add 700. In the optimal order, the monsters that must be fought cost 1,157 HP including the boss, so the optimal route ends with 13 HP. A single decision that wastes 13 HP or more loses the floor. Turns are numbered as in Section 5.5.1: the key at (5,1) is turn 1, and the first decision of Listing 5.1 is turn 2. At that decision, 19 of the 20 episodes of the re-run of Table 6.2 fought the Black Slime (6,4), the only choice that keeps the boss winnable (Figure 4.3); one episode of DeepSeek-V4-Flash fought the Slime Lord (4,2). The winning trajectory (Section 6.2) has seventeen decisions, on turns 2 to 5, 9 to 11, 15, 18, 22, 24, 27, 28, 31, 32, 35 and 36 (Table 4.4, last column); the turns in between are assisted. It follows the optimal route up to turn 11, the Green Slime (9,0). At turn 15 it fights the Slime Lord (1,9) before the Bat (3,5); the Slime Lord then costs 75 HP instead of 52 and the Bat 132 instead of 144, so the state value drops from 13 to 2 HP. This is why every win in Table 6.2 ends at 2 HP. From turn 15 on, the trajectory's slack is 2 HP, and any later decision that wastes 2 HP or more loses the floor. Two such decisions account for ten of the thirteen failed episodes in the re-run of Table 6.2. At turn 18, the hero stands at (0,7) with 720 HP, ATK 10 and DEF 6, after the blue gem behind the Slime Lord (1,9). Only the Bat (3,5), at 132 HP, keeps the boss winnable; the Slime Lord (4,2) costs 60 HP now against 22 at its place on the trajectory (turn 31). At turn 22, the hero stands at (2,4) in the left corridor with 663 HP, ATK 11, DEF 6 and one key. The door (7,5) keeps the state value at 2 HP; the Slime Lord (4,2) costs 52 HP now against 22 later, and the boss can no longer be defeated. The failed episodes discussed below were compared with the winning trajectory action by action to find the first action at which each departs, and the state value after that action shows whether the departure lost the floor or

only lowered the result.

6.4.1 Engine Overrides

In 94 turns of 68 stored simple-floor episodes, the engine had executed a different action from the one the model chose, and none of these replacements was logged. Every one of the 21 stored wins before the logging change contains such a replacement. In 17 of them it occurs at turn 9: the model chose the Slime Lord at (1,9), and the engine executed the door at (7,0). Reproducing that configuration deliberately gave 2/4 [0.15, 0.85], both wins again containing the replacement (Table 6.5, row 1). After the overrides were reduced to mechanical cases and logged, no strategic override occurred in any single-floor episode except those of row 1 of Table 6.5, which re-enables it deliberately. In the fifteen winning episodes of Qwen3.5-397B in the final configuration (Tables 6.2 and 6.3), no override of any kind occurred.

6.4.2 One-Sided Annotations

At the decisions whose reasoning was read, the model cited an annotation that stated one side of a trade-off and omitted the other. Five kinds were identified, on the simple floor and the complex floor.

1. **A saving without the cost of reaching it.** At turn 9 of the simple floor, the observation stated that the stat items behind door (7,5), in the right-hand rooms, "save 155 HP" and those behind door (7,0), in the top-right room, "save 94 HP". It did not state that reaching them costs 198 and 16 HP of fighting.
2. **A value without its cost: potions counted as fight value.** At the same turn, the Slime Lord at (1,9) was listed with an effective cost of −195. A 200-HP potion behind it was counted as its value, although the potion is collected in any order. On the complex floor with the single-floor prompt, the effective cost of a 235-HP fight was −218 for the same reason. The model reasoned that "the potion immediately restores me … a net gain of 18 HP".
3. **An efficiency without affordability.** On the simple floor, the positive form of the fight-order line ("take this fight before …") recommended a Bat costing 46.5 % of HP, without stating that this exceeds the rule 4 threshold (Table 6.5, row 8).
4. **A deficit without scope.** On the complex floor with the single-floor prompt, a floor budget summed all remaining monsters, including optional ones, and declared a winnable floor unwinnable ("CANNOT WIN even with all gems"). The model read it as a licence for expensive fights.

5. **A count without multiplicity.** On the simple floor, the lookahead tree listed identical items once, so two blue gems and two potions behind one door appeared as one each, and a profitable route appeared as a loss.

A sixth case was one-sided in a different sense. The fight-order line was printed only when the other order was cheaper. At turn 18, the Bat (3,5) therefore carried no line, and the model applied the line of the Bat (5,9) to it. Each defect was repaired in the engine by stating the missing side, not by changing the prompt.

6.4.3 Repair Sequence

Table 6.5 lists the changes cumulatively. Each row adds the change or changes named in it to the row above; from row 2 on, only mechanical assists are active.

Table 6.5: Cumulative repair sequence on the simple floor; k/N successes. Every success ends at HP 2 (HP/opt 0.15) unless noted; "—" = not run with Qwen3.6-35B.

#	Change	Qwen3.5-397B	Qwen3.6-35B
1	Earlier hand-written prompt with the strategic override (logged)	2/4 [d]	—
2	Single-floor prompt, mechanical assists only	0/5	—
3	+ alternative-route tag	0/5	—
4	+ net stat-route and fight-order lines	10/10	0/5
5	+ individual item counts, potion-free effective cost, affordability guard on the order line, required-monster budget	4/5	1/5 [a]
6	+ key budget and per-door key lines	10/10	1/5 [b]
7	+ no conversation history	9/10 [c]	10/10
8	+ two-sided fight-order line, key line on fights	7/10	1/10
9	+ 40 % guard on the positive order line (final configuration)	10/10	7/10

[a] Reasoning cap 8,192 tokens; truncated answers were repaired by constrained decoding, and in one episode the repair chose the unwinnable boss on 12 turns, before the repair was restricted to actions that the mechanical overrides would not replace (Appendix E, incident 9). [b] Reasoning cap 16,384 tokens from this row on (8,192 in rows 4 and 5). [c] One success at HP 10. [d] A fifth episode ended with a server error and is not counted.

The single largest step is row 4 (Figure 6.1): with model, prompt, temperature and depth fixed, adding two annotation lines raised Qwen3.5-397B from 0/5 [0.00, 0.43] to 10/10 [0.72, 1.00]. Rows 5 and 8 indicate that a repair can itself introduce

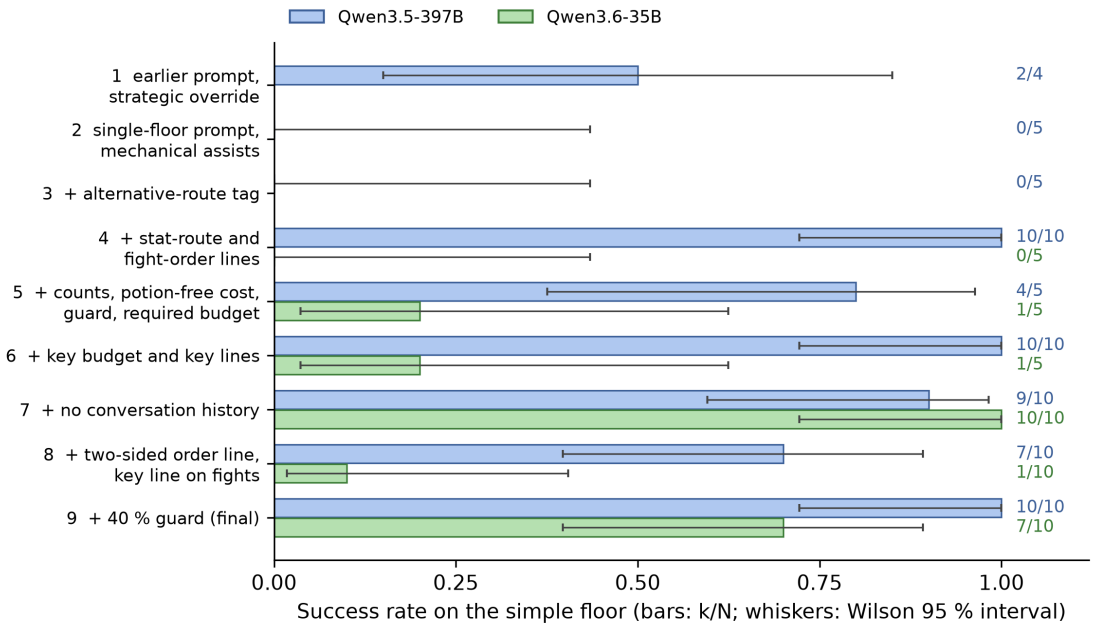


Figure 6.1: Success rate on the simple floor after each step of the repair sequence of Table 6.5. Bars show k/N, whiskers the Wilson 95 % interval; rows 1 to 3 were run with Qwen3.5-397B only.

a one-sided line. The changes of row 5 brought two doors within 9 HP of each other on the net stat-route balance. In the one failed episode of row 5, the model then took the door (7,5), which spends the only yellow key and gives none back, at turn 4 instead of the door (4,7); after that decision the boss can no longer be defeated. After the key lines of row 6, Qwen3.5-397B won 10/10. The positive order line of row 8 was followed as an instruction even when it recommended a fight above the rule 4 threshold of 40 %. With the guard of row 9, Qwen3.5-397B won 10/10 and Qwen3.6-35B 7/10. Qwen3.6-35B won 1/10 [0.02, 0.40] at row 8, against 7/10 for Qwen3.5-397B. With conversation history (row 6), it won 1/5, against 10/10 without it (row 7). Its row-5 result is confounded by the reasoning cap (note [a]).

6.4.4 Reasoning at the Decisive Turn

Listing 6.1 contrasts the reasoning of Qwen3.5-397B at turn 9 of the simple floor before and after row 4. The state is the same: the hero stands at (1,10) at the far end of the lower-left room with 336 HP, ATK 9, DEF 5 and two yellow keys.

```
Before (row 2, failure; 3,896 characters of reasoning, excerpt):
"6.  **Open Yellow Door at (7,0)**
```

```
    - Dead end / far away. I am at (1,10). ...
 Action 1 has effCost = -195. This is the best value on the board. ...
 Action 1 is clearly the winner. It is adjacent, negative effective cost (gain),
 and provides resources."

After (row 4, win; 1,329 characters of reasoning, excerpt):
"Action 6 (Open Yellow Door at 7,0) ... unlocks a path to +1 ATK gem with only
 16 HP fighting cost. This gives +78 HP net benefit ... Action 1 (Fight Slime
 ↪  Lord
 at 1,9) has lookahead=290 but costs 95 HP now. The note says it wastes 12 HP if
 done before getting the ATK from action 6. ... Action 7 ... would cost 198 HP
 ↪  to
 get +1 ATK +1 DEF - not as efficient as action 6."
```

Listing 6.1: Reasoning of Qwen3.5-397B at turn 9 of the simple floor, before and after the net stat-route and fight-order lines were added. By the exact solver, the Slime Lord (1,9) chosen in the failure makes the boss unwinnable, the door (7,0) keeps the state value at 13 HP and the door (7,5) lowers it to 10 HP. Appendix G gives the failing text with one omission (Listing G.1) and the winning text in full (Listing G.2).

Before the change, every premise of the reasoning came from the model's input. The effective cost of −195 and the unpriced door routes were in the observation. The "dead end" label had been on the door at turn 2 (Listing 5.1), where, with one key, it was correct; at turn 9, with two keys, it no longer applied but was still in the conversation history. After the change, the same model at the same state cites the new lines and chooses the door.

Listing 6.2 shows the effect of the unguarded positive order line of row 8 on Qwen3.6-35B at turn 4.

```
Observation (excerpt):
"take this fight BEFORE 'Open yellow door at (4,7)' - this fight then that
↪  route:
 312 HP; that route then this fight: 323 HP (saves 11)"   [Bat (3,5), 182 HP,
 ↪  46.5 %]

Reasoning (excerpt):
"This is a direct instruction from the engine's lookahead analysis. I should
↪  follow it."
```

Listing 6.2: Qwen3.6-35B reading a positive order line as an instruction. The comparison is locally exact but recommends a fight above the rule 4 threshold of 40 %; nine of ten episodes failed identically. A longer excerpt is given in Appendix G (Listing G.3).

6.4.5 The Complex Floor: Where the Episodes Are Lost

The complex floor holds three yellow keys for four yellow doors and two blue keys for five blue doors, and only the blue door (6,2) gates the boss. Its exact optimum (68 HP) opens five of the nine doors and fights 15 of the 24 monsters besides the boss, yet collects all 20 gems (Appendix B.5.2). Opening every door and fighting every monster is therefore not necessary. The solver's per-decision analysis (Appendix B.2.3) was applied to every state of 55 complex-floor episodes. They are the episodes of the re-run of Table 6.2 and those of Qwen3.5-397B with the single-floor prompt on the annotation versions of Table 6.5, rows 5 and 6. The state value (Section 5.9.2) is the highest final HP that can still be reached from a state. The drop in state value at a decision is the final HP that the decision costs. The decision after which no route defeats the boss is the one that loses the episode. Table B.3 lists the result per configuration, and Figure 6.2 plots the state value along every episode.

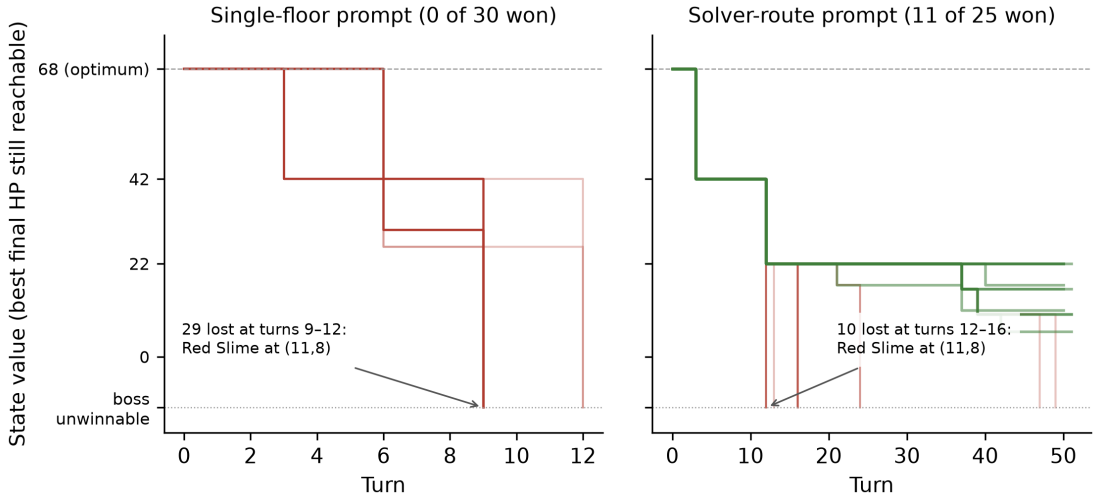


Figure 6.2: Exact state value after each turn of every complex-floor episode of Table B.3. The state value is the highest final HP still reachable; left = single-floor prompt (30 episodes), right = solver-route prompt (25 episodes). Red = failed, green = won; episodes that took the same decisions draw the same line, so darker lines stand for more episodes. A line ends when the boss becomes unwinnable or, in a won episode, at the boss fight. Dashed line = exact optimum (68 HP); the labels mark the losses at the Red Slime (11,8).

The single-floor prompt. All 30 episodes with the single-floor prompt fail, each at a single fight: 29 at the Red Slime (11,8) and one at the Red Bat (4,11), at turn 9 in 28 episodes and at turn 12 in two. Before that fight, 28 episodes had opened the yellow door (8,7) before (5,8), which lowers the value from 68 to 42 or

30. The other two had fought the Black Slime (12,9) at turn 6, which lowers it to 26. In the 28 episodes lost at turn 9, four fights are offered at that turn: the Red Slime (103 HP), the Black Slime (12,9) (156 HP), the Slime Lord (10,10) (235 HP) and the Red Bat (310 HP). Only the Black Slime keeps the boss winnable. The Red Slime is the cheapest fight at turn 9. Its cost falls to 0 at DEF 20, and the optimal route fights it as its twelfth decision, for 11 HP. The observation carried both sides. It showed an effective cost of 13, with the DEF gem behind the fight counted as value, and "min-cost=0@maxStats save=103HP — collect stat gems first". An order line compared the fight with a route that "costs 1217 HP (126 % of your HP)". In 29 of the 30 episodes the model chose the Red Slime; DeepSeek-V4-Flash, for example, reasoned that it was "the cheapest meaningful progress". Several door decisions come after the episode is lost. The yellow door (10,5) was opened in 17 of the 20 re-run episodes, at turns 16 to 24. Eighteen of the 20 spent a blue key on (12,2) (17 episodes) or (4,5) (one), and only one opened the boss door (6,2).

The solver-route prompt. The prompt's door order caps the result: opening (8,7) first and (8,4) second lowers the value from 68 to 42 and then to 22. Every win (11 of 25 [0.27, 0.63]) therefore ends at 22 HP or below, at most a third of the optimum. Each of the 14 failures is lost at a fight taken out of the prompt's order. Ten are lost at the Red Slime (11,8) at turns 12 to 16, including all five episodes of Mistral Medium 3.5. Two are lost at the Big Bat (9,1) at turn 24 and two at the Big Bat (0,6) at turns 47 to 49. Every episode of this prompt opened the boss door (6,2), as the route requires.

6.5 Prompt Optimisation (RQ2)

6.5.1 Study Design and Hypotheses

The study asks whether the reflective prompt optimisation of Section 5.7 adds anything once the engine annotates every action, and whether it adds something when the annotations are withheld. It uses the two weaker models of Table 6.1, Llama-3.1-8B and DeepSeek-V4-Flash (no thinking), because Qwen3.5-397B already wins the simple floor with the single-floor prompt (Table 6.2).

Each model is run with three prompts. The *rules-only* prompt states only the game rules, the goal and the answer format. The *evolved* prompt is what the (1+1) loop of Section 5.7 produces from it, and the *hand-written* prompt is the single-floor prompt. Listing A.3 gives the rules-only prompt, and Listing A.4 the one evolved prompt that passed the paired test of Section 6.5.2. Each prompt is run under the full and under the bare observation. The loop uses the simple floor and two generated floors for training, six evaluations per run, and Qwen3.5-

397B as the teacher, which is given no example prompt rules. The three prompts are then evaluated on thirteen held-out generated floors, one episode per floor. As a reference, Qwen3.5-397B and Qwen3.6-35B are run on the simple floor with the rules-only prompt under both observations and with the hand-written prompt under the bare observation ($N = 3$ each).

Five hypotheses were stated before the runs:

- **G1.** Under the full observation, the evolved prompt improves on the rules-only prompt by at most 0.05 mean goal score on the held-out floors, and the rules-only prompt performs about as well as the hand-written prompt.
- **G2.** The gain from optimisation is larger under the bare than under the full observation.
- **G3.** The rules-only prompt under the full observation beats the evolved prompt under the bare observation.
- **G4.** A gain on the training floors that does not carry over to the held-out floors is read as overfitting.
- **G5.** Qwen3.5-397B wins the simple floor with the rules-only prompt under the full observation in at least two of three episodes, and in none under the bare observation.

6.5.2 Results

Table 6.6: Held-out generated floors (13, one episode each): kills and mean goal score per prompt. Rules-only = game rules only; Evolved = output of the (1+1) loop from the rules-only prompt; Hand-written = single-floor prompt. Differences in the text are computed from unrounded scores.

Model	Observation	Rules-only	Evolved	Hand-written
Llama-3.1-8B	full	8 / 0.556	10 / 0.646 [a]	9 / 0.660
Llama-3.1-8B	bare	5 / 0.462	4 / 0.399	6 / 0.484
DeepSeek-V4-Flash (no thinking)	full	10 / 0.820	11 / 0.846	11 / 0.850
DeepSeek-V4-Flash (no thinking)	bare	9 / 0.678	= rules-only [b]	7 / 0.543

[a] The optimiser returned its seed; this prompt is byte-identical to the rules-only prompt. [b] The optimiser accepted no revision and returned the seed.

No episode ended in death (Table 6.6).

Noise floor. When the optimiser keeps its seed, the evolved arm is the rules-

only prompt played a second time. Four such pairs exist on the same thirteen floors: Llama-3.1-8B under the full observation in Table 6.6 (note [a]) and the three cells of the second configuration (below) that kept the seed. Their mean goal scores differed by +0.089 (Llama-3.1-8B, full observation, Table 6.6), +0.051 (the same cell in the second configuration), −0.043 (Llama-3.1-8B, bare) and −0.132 (DeepSeek-V4-Flash, full). On the training floors, the same starting prompt scored 0.463 and 0.602, 0.775 and 1.000, and 0.607 and 0.387 in two evaluations each. With one episode per floor, differences between candidates inside an optimisation run (0.01 to 0.24) are of the same size as differences between two evaluations of one prompt. The latter reach 0.225 on the training floors and 0.132 on the held-out floors.

Hypotheses. G1 is supported in part. Under the full observation, the one evolved prompt that differs from its seed (DeepSeek-V4-Flash) gains 0.025, within the pre-registered 0.05. The Llama-3.1-8B difference of +0.089 exceeds 0.05 but lies between two byte-identical prompts (note [a]). The hand-written prompt scores 0.104 (Llama-3.1-8B) and 0.030 (DeepSeek-V4-Flash) above the rules-only prompt, within the spread of repeated evaluations of one prompt (up to 0.132, above). The results do not support G2: under the bare observation, the evolved prompt for Llama-3.1-8B scores 0.063 below its seed on the held-out floors, and the optimiser kept the seed for DeepSeek-V4-Flash. G3 is supported for both models (Llama-3.1-8B 0.556 against 0.399; DeepSeek-V4-Flash 0.820 against 0.678, where the evolved prompt is the seed). G4: the one training gain under the full observation (DeepSeek-V4-Flash, from 0.775 for the seed to 0.859) did not survive paired re-evaluation. Played again on the training floors, the evolved prompt scored 0.775 and the rules-only prompt 0.859. G5 is supported: Qwen3.5-397B and Qwen3.6-35B each win the simple floor in 2/3 [0.21, 0.94] episodes with the rules-only prompt under the full observation. Under the bare observation, they win none with either prompt (0/3 [0.00, 0.56] each).

A second configuration. The teacher's trace was extended to show, at every decision, the options offered and a longer reasoning excerpt, and DeepSeek-V4-Flash replaced Qwen3.5-397B as teacher. On a stored request that had led the first teacher to write a harmful rule ("fight the boss even if lethal"), the new configuration produced a correct diagnosis. In three of four cells, the optimiser again kept the seed. The one accepted rewrite (DeepSeek-V4-Flash, bare observation) raised the training score from 0.387 to 0.624, about as much as two evaluations of the seed differ on those floors (0.387 and 0.607). On the held-out floors it equalled the rules-only prompt (9/13 [0.42, 0.87] each).

What the evolved prompts contain. Table 6.7 sets the prompt rules of the hand-written prompts against what evolution wrote on its own. Every rule of the

single-floor and tower prompts was written for this thesis from the sources of Table 4.2; none was copied from an evolved prompt. The table counts, for each rule, the candidates that state it: the forty candidates of the first study and the fourteen of the second (Section 6.5.3). All were written from the rules-only prompt by a teacher that was given no example prompt rules.

Table 6.7: Origin of the prompt rules of the hand-written prompts and how often prompt evolution rediscovered them. Origin = source of the strategy principle in Table 4.2 (S-number in parentheses). First study = forty candidates of the (1+1) loop (Section 6.5.2); second study = the seven new candidates of each of the (1+1) loop and the Pareto loop (eight per loop including the starting prompt), with DeepSeek-V4-Flash with thinking and the bare observation (Section 6.5.3). Counted by keyword patterns (Appendix D). — in the Origin column: not a rule of the hand-written prompts; — in the last column: no line of the full observation expresses the rule.

Rule	Origin	First study	Second study, (1+1)	Second study, Pareto	Expressed in the full observation by
collect stat items before the fights they make cheaper [a]	combat formula, exact solver (S3)	37/40	7/7	7/7	net stat-route line, fight-order line, cost at maximum stats
take the cheapest progress first (lowest effective cost)	design of this thesis (definition of the effective cost)	26/40	6/7	4/7	effective cost, fight-order line
never pay HP for a fight that unlocks nothing needed	exact solver (S5)	30/40	7/7	6/7	OPTIONAL tag, alternative-route tag
open doors only when what lies behind is needed	exact solver (S6)	14/40	7/7	6/7	DEAD END tag, key line
no fight above 40 % of current HP (tower: 60 %)	design of this thesis (S7)	3/40	2/7	0/7	—

Rule	Origin	First study	Second study, (1+1)	Second study, Pareto	Expressed in the full observation by
keys are scarce	exact solver (S6)	0/40	0/7	1/7	key budget
scout the next floor first; shop rules (tower only)	tower structure (S8); combat formula, expert walkthrough (S9)	not applicable (single floors)			way-up line, shop lines
Rules evolution added: trust the engine's annotations	—	32/40	2/7 [b]	3/7 [b]	—
Rules evolution added: fight the boss only when winnable and after the stats	—	18/40	4/7	1/7	boss line, WINNABLE tag
Rules evolution added: make the reasoning and the chosen action agree	—	8/40	0/7	0/7	—

[a] The rules-only prompt already contains a sentence that the pattern matches, so this row overstates rediscovery. [b] The second study used the bare observation, which shows no annotations.

Three of the first four rules appear in most candidates of both studies. These are stat items first (37/40, 7/7, 7/7, partly already in the rules-only prompt), no HP for unneeded fights (30/40, 7/7, 6/7) and cheapest progress first (26/40, 6/7, 4/7). Opening doors only when needed appears in 14 of 40 first-study and 13 of 14 second-study candidates. The rule 4 threshold (S7) and key scarcity (S6) were

almost never written (5 and 1 of 54 candidates).

The one rewrite that changed behaviour. The accepted rewrite for DeepSeek-V4-Flash under the bare observation (Listing A.4) states: "A fight is necessary only if that monster currently blocks your path to an uncollected gem/potion/key, or to the boss. … If the boss is the only remaining goal and the boss fight shows a positive HP-after, take the boss immediately." The rewrite was taken to confirmation after a trial with three episodes on each of four held-out floors (gen10, gen12, gen13 and gen14). These four floors had been chosen because the rewrite had won each of them in the single-episode evaluation. The pre-registered confirmation covered all thirteen held-out floors with three repetitions each, the trial episodes included. It passed its criterion of more kills and a one-sided sign test at $p < 0.05$ (Section 5.9.4). The rewrite won 30/39 [0.62, 0.87] episodes and the rules-only prompt 24/39 [0.46, 0.75], with discordant pairs $7 : 1$ ($p = 0.035$) and mean goal score 0.724 against 0.613. The effect is carried by two floors, gen08 and gen12, on which the rules-only prompt stops fighting with the boss alive and HP to spare. The four trial floors, chosen because the rewrite had won them, contribute $4 : 0$ of the discordant pairs, so the test is not independent of that choice. On the nine floors not used in the trial, the discordant pairs are $3 : 1$ and the kills 18 against 16 ($p = 0.31$; Figure 6.3), which does not separate the two prompts.

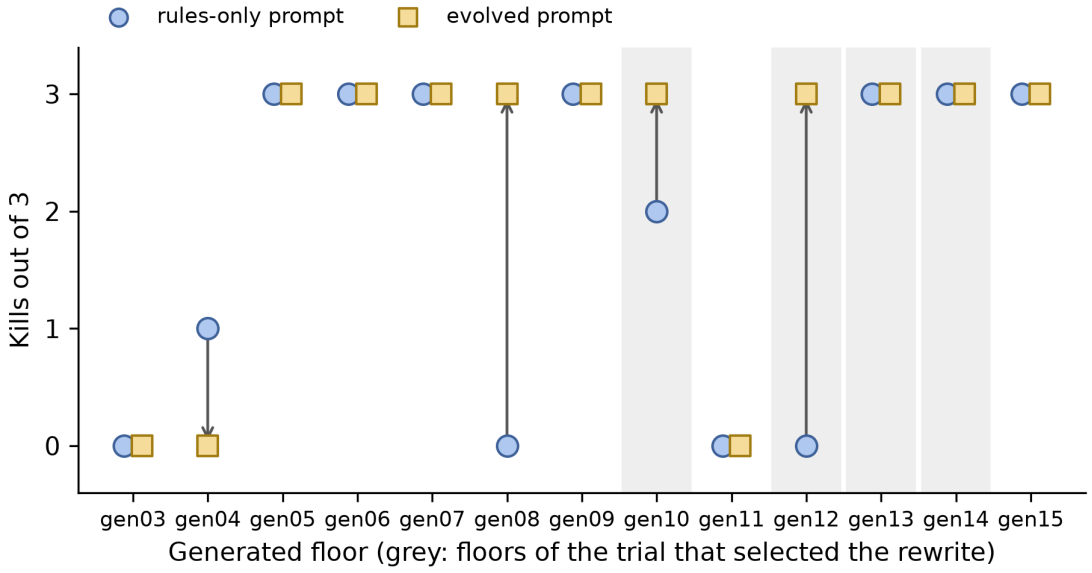


Figure 6.3: Paired test of the evolved prompt for DeepSeek-V4-Flash (no thinking) under the bare observation: kills out of three episodes per generated floor for the rules-only and the evolved prompt. Arrows mark floors on which the two differ; grey floors belong to the earlier trial on which the rewrite was selected and are part of the test.

The complex floor. A further pre-registered run started the optimiser from the hand-written single-floor prompt on the complex floor alone (Qwen3.6-35B, full observation, six evaluations of two episodes each). The prediction was that no evolved prompt would win, because the only prompts that ever won this floor encode a route (the solver-route prompt of Section 5.4, Listing A.5). The prediction held: no episode reached the boss during optimisation, and at $N = 5$ the single-floor prompt and its evolved version both won 0/5 [0.00, 0.43].

6.5.3 Second Study: Population Versus Single Lineage

The second study repeats the setting of the first study's only gain that passed its paired test, the bare observation, with three changes. The task model reasons (DeepSeek-V4-Flash with thinking). The same model is the teacher and sees the options offered at every decision. Two loops with the same budget of eight candidates are compared: the (1+1) loop and the Pareto loop of Section 5.7 (Table 5.2). Both start from the rules-only prompt and train on the simple floor, gen01 and gen02. The rules-only prompt and the best prompt of each loop are then played on the thirteen held-out floors three times each, paired by floor and repetition. The study was not pre-registered, so its sign tests are two-sided (Section 5.9.4).

Table 6.8: Second study: training score of the best candidate and held-out result (13 floors × 3, DeepSeek-V4-Flash with thinking, bare observation). Sign test on the episodes in which exactly one of the two prompts won, two-sided. Bold = best held-out result; — = not applicable.

Prompt	Training score	Held-out kills	Held-out goal score	Against rules-only (only rules-only wins : only this prompt wins)
Rules-only	0.597	**32/39**	**0.737**	—
(1+1) loop, best of 8	0.732	26/39	0.620	7 : 1, $p = 0.07$
Pareto loop, best of 8	0.678	24/39	0.602	8 : 0, $p = 0.008$

Both loops raised the training score, and both best prompts score below the rules-only prompt on the held-out floors (Table 6.8). The rules-only prompt wins 32/39 [0.67, 0.91] episodes, the best prompt of the (1+1) loop 26/39 [0.51, 0.79] ($p = 0.07$) and that of the Pareto loop 24/39 [0.46, 0.75] ($p = 0.008$). The two evolved prompts differ in 4 : 2 discordant pairs ($p = 0.69$). No episode ended in death; the evolved prompts stop with the boss alive where the rules-only prompt fights it. The gain on the training floors is therefore overfitting in the sense of hypothesis G4, now with a model that reasons and a teacher that sees every option.

The Pareto selection did not operate as intended. All eight candidates of the Pareto loop scored exactly the same on the simple floor (0.325) and on gen01 (0.762). On these two floors, no rewrite changed the outcome of the episode at temperature 0. Every candidate was therefore best on at least one floor, and the frontier contained the whole population. Parents and the reported best were chosen on gen02 alone, the one floor on which the candidates' scores differed (0.550 to 0.948). The merge of the two best candidates scored exactly as one of its parents.

6.5.4 Evolving a Route Without the Validator

The last run asks whether evolution can compile from its own traces what the solver found by search. On the complex floor alone, the (1+1) loop started from the single-floor prompt with the validator switched off. The teacher was told that the prompt is used on this floor only and may contain positions, orders and conditions. It saw every trace with the options offered and the end state, but never the solver's route. DeepSeek-V4-Flash with thinking was task model and teacher; the budget was 15 evaluations of two episodes. The best prompt, the *evolved-route prompt*, was then played five times. Three predictions were stated before the run: a win appears within the 15 evaluations (P1); the evolved-route prompt names the yellow door (10,5), which the solver-route prompt calls a trap and the exact optimum never opens (P2); and its success rate does not exceed the solver-route prompt's 4/5 for this model (P3).

No episode was won in training (scores 0.133 to 0.175 in all 15 evaluations) or in the five evaluation episodes (0/5 [0.00, 0.43]), all of which ended alive (2 to 155 HP). The results do not support P1 and P2, and P3 holds trivially. Although positions were allowed, no candidate named one. Every rewrite stayed a general rule set, and the evolved-route prompt sharpened the reading of the annotations instead ("If min-cost@maxStats is 0 or much lower, the fight MUST be deferred"). One of its rules reads "NEVER fight a monster with lookahead=0 or effCost=None — it is a dead end". It forbids exactly the Black Slime (12,9), the only fight that keeps the boss winnable at turn 9 (Section 6.4.5). The teacher diagnosed from the traces what the agent did, not which earlier decision lost the floor.

6.6 Planning Horizon (RQ3)

The planning study varies the planning horizon: it compares the reactive mode with the committed modes of Section 5.5.2. Each scenario uses the prompt that wins it, and each model is compared with its own reactive arm. Episodes and runs affected by four defects of the plan harness and by reasoning caps that truncated plan answers are excluded and described in Appendix D.6. The two long-plan modes were run with two instructions. The *whole-floor wording* asks for the plan

"for the rest of this floor … up to the goal". The *max-steps wording* (Listing A.8) asks for "as many of the actions you will take after it as you can", so that the model, not the instruction, sets how far it commits.

Table 6.9: Success k/N and model calls per episode (in parentheses) by planning mode. Commit-k = the model commits to k actions per call; long plan = the model commits to as many actions as it can (at most 59), re-planned when a step breaks or written in a single call for the whole episode (on the tower: one plan per floor). Long-plan cells give whole-floor wording / max-steps wording; DeepSeek-V4-Flash reactive cells give reasoning caps of 16,384 / 32,768 tokens. Complex route = complex floor with the solver-route prompt (Listing A.5). Tower = highest floor reached with the second tower observation (Section 6.7.3); reactive: three runs, other modes: one run each. Reactive single-floor cells are the earlier episodes of the final configuration, run on the same version of the code as the committed modes (Table D.4), and can therefore differ from Tables 6.2 and 6.11. Bold = highest success rate in the row (not marked where all modes tie); — = not run.

Floor	Model	Reactive	Commit-2	Commit-3	Commit-5	Long plan, re-planned	Long plan, single call
Simple	Qwen3.5-397B	**10/10** (17.0)	1/5 (9.6)	0/5 (7.6)	0/5 (5.2)	0/5 (3.6) / 0/5 (4.6)	0/5 (1.0) / 0/5 (1.0)
Simple	DeepSeek-V4-Flash	**3/5** (16.8) / 1/5 (17.0)	0/5 (9.6)	0/5 (7.8)	0/5 (5.8)	0/5 (5.4) / 0/5 (6.2)	0/5 (1.0) / 0/5 (1.0)
Simple	Qwen3.8-27B, temperature 0.6 [a]	**3/3** (17.0)	2/3 (9.3)	0/3 (7.0)	1/3 (5.0)	— / 0/3 (4.0)	— / no usable plan [b]
hand01	DeepSeek-V4-Flash	4/5 (13.8) / **5/5** (13.0)	—	0/5 (8.0)	0/5 (6.4)	— / 0/5 (5.2)	—
hand02	DeepSeek-V4-Flash	**5/5** (9.0) / 4/5 (9.8)	—	0/5 (7.0)	0/5 (5.8)	— / 0/5 (5.2)	—
Complex route [c]	DeepSeek-V4-Flash	3/5 (21.4) / 8/10 (20.8)	3/5 (15.6)	3/5 (9.6)	4/5 (5.8)	4/5 (3.0) / **5/5** (2.4)	0/5 (1.0) / 0/5 (1.0)
Complex route [c]	Qwen3.8-27B, temperature 0.6 [a]	3/3 (20.0)	3/3 (10.3)	3/3 (7.0)	3/3 (4.7)	— / 3/3 (2.0)	— / 0/3 (1.0)
Tower	DeepSeek-V4-Flash	MT8, MT8, MT8 (86)	MT8 (72)	MT8 (60)	MT8 (58)	— / MT8 (66)	— / MT3 (2)

[a] Reasoning cap 32,768 tokens. Earlier episodes of this model are excluded (Appendix D.6). [b] The single plan could not be parsed in any of the three episodes, so the heuristic played them (443 HP, boss untouched); not counted as a result. [c] Run without the mechanical overrides and the automatic 0-HP fights (Section 5.4).

On the simple floor, one of 70 committed-plan episodes of Qwen3.5-397B and DeepSeek-V4-Flash wins (1/70 [0.00, 0.08]; Qwen3.5-397B, commit-2), against

14/20 [0.48, 0.85] reactive episodes of the same two models. None of the 20 episodes with the max-steps wording wins. Qwen3.8-27B at temperature 0.6 loses less sharply but in the same direction. It wins 3/3 [0.44, 1.00] reactively, 3/9 [0.12, 0.65] with commit-2, -3 and -5 plans, and none of three with re-planned long plans (the single-call plan could not be parsed). Every failure ends alive, with 287 to 572 HP and the boss not killable; no committed episode dies. Even the smallest commitment wins less often: Qwen3.5-397B wins 1/5 at commit-2 against 10/10 reactively, and DeepSeek-V4-Flash 0/5 against 1/5 at the same reasoning cap. The hand-designed floors on which reactive play succeeds show the same pattern. On hand01 and hand02, DeepSeek-V4-Flash wins 18/20 [0.70, 0.97] reactive episodes over both reasoning caps and none of 30 committed episodes (0/30 [0.00, 0.11]; commit-3, commit-5 and re-planned max-steps plans), every one alive. Of the 87 broken plans on these floors, 50 break at a fight placed where it cannot yet be won.

The number of calls falls with k, but the number of actions actually committed grows much more slowly. A simulation of every stored plan (Table 6.10) shows that commit-5 plans commit about three new decisions per plan on average. Long re-planned plans commit about five on the simple and hand-designed floors (3.63 to 6.77). On the simple floor with the whole-floor wording, re-planned long plans are legal throughout in 6 of 40 cases and single-call plans in none of 10 (none of 15 with the complex floor). They name monsters that are not yet reachable and fights that cannot yet be won at the point where the plan puts them. With the max-steps wording, the plans become shorter and more often legal. A single-call plan commits 5.8 (Qwen3.5-397B) or 6.6 (DeepSeek-V4-Flash) new decisions instead of 11.0 and 15.6. Of the single-call plans, 3 of 5 and 4 of 5 are clean instead of none. Of the re-planned plans, 8 of 20 and 12 of 27 are clean instead of 1 of 13 and 5 of 27. Success stays at 0/5 [0.00, 0.43] for both models, also in the episodes whose plans are clean.

Table 6.10: Plan check. New / plan = mean number of new decisions per committed plan; Clean = plans with no broken step. WF = whole-floor wording, MS = max-steps wording.

Floor	Model	Mode	New / plan	Clean
Simple	Qwen3.5-397B	commit-2 / 3 / 5	1.64 / 1.71 / 3.23	35/44, 25/35, 14/22
Simple	Qwen3.5-397B	long re-planned / single call (WF)	6.77 / 11.0	1/13, 0/5

Floor	Model	Mode	New / plan	Clean
Simple	Qwen3.5-397B	long re-planned / single call (MS)	4.05 / 5.8	8/20, 3/5
Simple	Qwen3.8-27B, temperature 0.6	commit-2 / 3 / 5	1.59 / 1.94 / 3.46	23/27, 14/18, 9/13
Simple	Qwen3.8-27B, temperature 0.6	long re-planned (MS)	5.33	3/9
Simple	DeepSeek-V4-Flash	commit-2 / 3 / 5	1.84 / 1.78 / 2.86	33/44, 25/36, 15/28
Simple	DeepSeek-V4-Flash	long re-planned / single call (WF)	4.89 / 15.6	5/27, 0/5
Simple	DeepSeek-V4-Flash	long re-planned / single call (MS)	3.63 / 6.6	12/27, 4/5
hand01	DeepSeek-V4-Flash	commit-3 / 5	1.91 / 3.19	20/34, 9/31
hand01	DeepSeek-V4-Flash	long re-planned (MS)	5.92	6/26
hand02	DeepSeek-V4-Flash	commit-3 / 5	1.62 / 2.78	19/26, 9/23
hand02	DeepSeek-V4-Flash	long re-planned (MS)	4.43	11/21
Complex route	DeepSeek-V4-Flash	commit-2 / 3 / 5	1.19 / 1.50 / 2.86	70/77, 40/48, 25/29
Complex route	DeepSeek-V4-Flash	long re-planned (WF / MS)	8.60 / 7.83	5/15, 7/12
Complex route	DeepSeek-V4-Flash	single call (WF / MS)	18.0 / 14.0	0/5, 1/4
Complex route	Qwen3.8-27B, temperature 0.6	commit-2 / 3 / 5	1.63 / 2.00 / 3.93	29/30, 21/21, 12/14
Complex route	Qwen3.8-27B, temperature 0.6	long re-planned / single call (MS)	9.50 / 18.0	3/6, 0/3

On the complex floor with the route listed in the prompt, the results differ. With the same reasoning cap as its committed modes, DeepSeek-V4-Flash wins 8/10 [0.49, 0.94] reactively with 20.8 calls per episode. Commit-5 plans commit 2.86 new decisions each, 25 of 29 run clean, and the mode wins 4/5 [0.38, 0.96], three kills at 22 HP, with 5.8 calls per episode. Re-planned long plans commit about eight new decisions each and win 4/5 with the whole-floor wording (3.0 calls) and 5/5 [0.57, 1.00] with the max-steps wording (2.4 calls), every kill at 22 HP. At these sample sizes, the intervals of commit-5 and re-planned long plans (4/5 and 5/5) overlap with that of reactive play (8/10), with 5.8 to 2.4 calls per episode against 20.8. Qwen3.8-27B at temperature 0.6 wins 3/3 [0.44, 1.00] in every re-planned mode, down to 2.0 calls per episode with max-steps plans. Commit-2 is close to reactive play in practice for DeepSeek-V4-Flash, because in 68 of 77 plans the model repeated its chosen action as the first planned entry, even after the instruction that had invited this was corrected (Appendix D.6). Single-call plans, although the route is in the prompt, win none with either model or wording; 11 of 12 single-call plans contain a broken step (Table 6.10). Figure 6.4 sets the two floors side by side.

On the tower, the planning modes were compared with the second tower observation (Section 6.7.3). With it, DeepSeek-V4-Flash reaches MT8 in all three reactive runs (86 calls on average). It also reaches MT8 with commit-2, commit-3 and commit-5 plans (72, 60 and 58 calls) and with re-planned max-steps plans (66 calls). A single plan per floor stops on MT3 at 5 HP after two calls. In each commit-k run, the engine itself takes about 120 of the roughly 205 turns (collection, 0-HP fights, scouting). Of the remaining decisions, the commit-3 and commit-5 runs take 29 and 31 from their plans.

Executing the exact optimum. The exact-route prompt (Listing A.6), a second compiled-knowledge condition, lists the 21 decisions of the exact optimal route with the expected HP after each; it is played reactively. Qwen3.6-35B wins 2/5 [0.12, 0.77] with it, both at 58 HP (0.85 of the optimum), the highest HP of any LLM kill on the complex floor. DeepSeek-V4-Flash wins 1/5 [0.04, 0.62], at 40 HP. Both winning episodes of Qwen3.6-35B take the 21 listed decisions in order; one of its three failures ends with the hero dead, because this condition, like the solver-route prompt, runs without the mechanical overrides (Section 5.4). The two wins also take two unlisted fights that cost 0 HP, against the Red Slimes (9,3) and (11,0). The solver takes such fights automatically, so they are not decisions of the route and are not listed, while this condition runs without the engine's automatic 0-HP fights. Before each of the first 18 listed decisions, the hero's attributes equal those on the solver's route at the same point. Both episodes then leave the Red Slime (11,0), which guards a gem of +5 MDEF, until after the 20th decision; the reasoning of one of them cites the prompt's rule against fights that are not in the

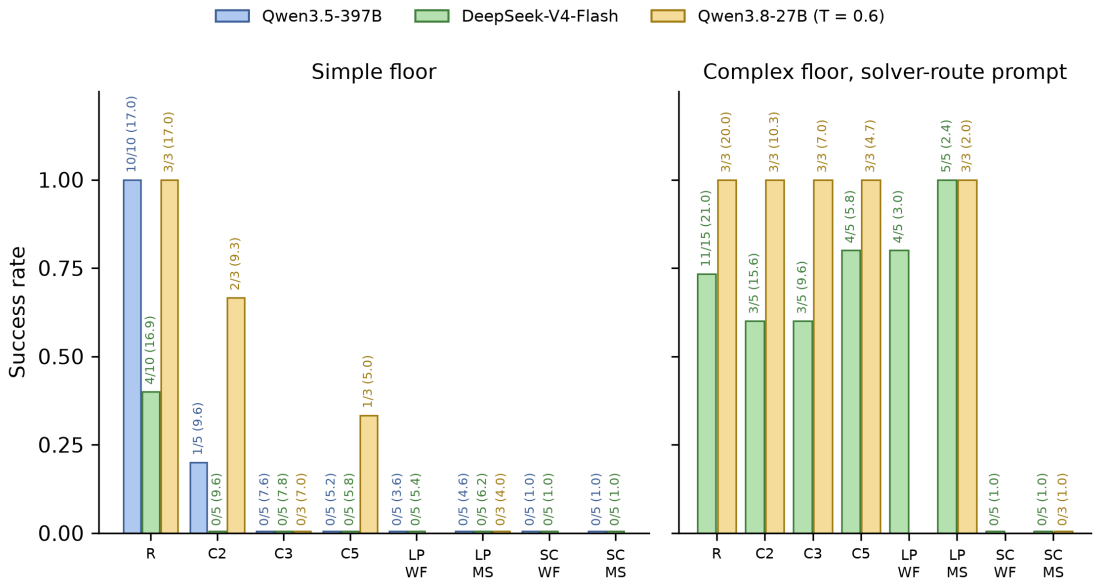


Figure 6.4: Success rate by planning mode on the simple floor and on the complex floor with the solver-route prompt. The labels give k/N and, in parentheses, the mean number of model calls per episode. R = reactive (DeepSeek-V4-Flash: both reasoning caps pooled); C2, C3, C5 = commit-2, -3, -5; LP = long plan, re-planned; SC = single call; WF = whole-floor wording; MS = max-steps wording. Qwen3.8-27B was run at temperature 0.6 and with the max-steps wording only; its single-call cell on the simple floor is not shown (no usable plan).

route. The Bat (11,4) and the Big Bat (12,12) are therefore fought with MDEF 15 instead of 20 and cost 5 HP more each, which accounts for the 10 HP between 58 HP and the optimum. Every failure leaves the listed route by the model's own choice. Three of five DeepSeek-V4-Flash episodes leave it at the same decision, taking a fight shown at 43 HP instead of the listed one shown at 185.

6.7 Hand-Designed Floors, Generated Floors and the Tower

6.7.1 Hand-Designed Floors

Table 6.11 combines the design mechanisms of Table B.2 and the per-decision analysis of Section 4.6.3 with the results of Table 6.2.

Table 6.11: Hand-designed floors: design, exact optimum and results of the final configuration, the episodes of Table 6.2 (five episodes per cell; k/N with HP/opt in parentheses; reasoning caps as in Table 6.1, DeepSeek-V4-Flash 32,768 tokens). Critical = actions on the optimal route at which at least one alternative makes the boss unwinnable; Forced = actions at which every alternative does.

Floor	Design	HP_{opt}	Critical / forced	Qwen3.5-397B	DeepSeek-V4-Flash	Qwen3.6-35B	Mistral Medium 3.5
hand01	four yellow doors and one blue door; rooms reached by a door or past a monster	14	13 / 3	5/5 (0.50)	4/5 (0.66)	4/5 (0.64)	5/5 (0.64)
hand02	defence thresholds; two keys for four doors	13	8 / 3	4/5 (0.48)	5/5 (0.28)	3/5 (0.31)	5/5 (0.86)
hand03	the complex floor's special-ability monsters on a 13 × 13 layout	9	8 / 8	0/5	0/5	0/5	0/5
hand04	six yellow doors for two keys, a costly shortcut, a blue door before the boss	3	12 / 11	0/5	0/5	0/5	0/5

The four models agree floor by floor. Each wins hand01 and hand02 in most episodes, and none wins hand03 or hand04 (0/20 on each floor). Which floors are won does not follow the number of critical actions. The floor hand01 has 13 and is won; hand03 has 8 and is not. The number of forced actions is 3 on hand01 and hand02, and 8 and 11 on hand03 and hand04; the simple floor, with 17 critical and 10 forced actions, is won by Qwen3.5-397B in 15 of 15 episodes. On hand03, the models repeatedly choose fights they cannot win. The engine replaced 57 such choices in the 20 episodes with a mechanical override, 9 to 20 per model. On hand04, they repeatedly choose doors marked as dead ends (48 mechanical overrides, 10 to 18 per model). Every episode on these two floors ends alive, with only unwinnable fights or dead ends left.

On hand01 (Figure B.2), the hero starts at (5,10) with one yellow key, between two yellow doors: (4,8) opens the left wing and (6,8) the right wing. Within the lookahead, the observation shows +2 ATK, +1 DEF and two yellow keys behind the left door for 112 HP of fighting, and +1 ATK, +1 DEF, a yellow key and the blue key behind the right door for 48 HP; the blue key opens the blue door (5,2) on the way to the boss. Every episode opens one of the two doors at its first decision. The exact solver gives an optimum of 14 HP after the left door at (4,8) and at

most 10 HP after the right door at (6,8). In 18 of the 20 episodes, the model opened the right door first. These are all five episodes of Qwen3.5-397B, Qwen3.6-35B and Mistral Medium 3.5 and three of DeepSeek-V4-Flash. All 17 wins among these 18 episodes end at 4 to 9 HP. In the other two episodes, DeepSeek-V4-Flash opened the left door first; one won at 10 HP, and the other ended alive at 153 HP with the boss not defeated. The observation at this decision (Appendix G, Listing G.4) rates the right door higher on both one-step measures. Its net stat-route balance is −33 against −92 HP, and its net lookahead value 170 against 121. In all five episodes of Qwen3.5-397B, the reasoning quotes the fight-order line of the Black Slime (5,4), which stands in the central room and blocks nothing. That line names the right door as the route that makes the fight cheaper. Appendix G gives the reasoning of DeepSeek-V4-Flash (left door, Listing G.5) and of Qwen3.5-397B (right door, quoting the same line, Listing G.6) at this decision, from episodes of the same configuration on an earlier version of the code.

6.7.2 Generated Floors

Most of the fifteen generated floors have more HP slack than the simple and complex floors: on 13 of them, the optimum ends at 150 to 250 HP. Pooled over all stored episodes of all models, prompts and observations, the kill rate per floor ranges from 18 of 18 to 0 of 18 (Table 6.12). The heuristic wins eight of the fifteen floors. With the final configuration, Qwen3.5-397B wins gen01 and gen02 in 3/3 [0.44, 1.00] episodes each, at the exact optimum (168 and 232 HP). It also wins gen03, the only generated floor with an optimum as low as the simple floor's (13 HP), in the one episode that was run, at the exact optimum. DeepSeek-V4-Flash was run with the final configuration and a reasoning cap of 16,384 tokens, three episodes on each of the thirteen floors gen03 to gen15. These floors were not used to develop the single-floor prompt or the annotations. It wins 31/39 [0.64, 0.89], at 0.96 of the optimum on average and in 11 episodes exactly at it. It wins 18/18 [0.82, 1.00] on the six of these floors that the heuristic wins, and 13/21 [0.41, 0.79] on the seven that the heuristic loses. The floors gen11 and gen12 are lost in all three episodes, alive. On the same thirteen floors, three episodes each, the same model won 32/39 with the bare observation and the rules-only prompt (Section 6.5.3, Table 6.8). With this much HP slack, the generated floors therefore do not separate the observation conditions. The evidence for the annotations comes from the simple floor and from hand01 and hand02 (Sections 6.3 and 6.4.3).

Table 6.12: Generated floors: exact optimum, heuristic result and pooled LLM kills over all stored episodes except the paired confirmation episodes of Section 6.5.2 (a difficulty signal, not a comparison of conditions).

Floor	HP_{opt}	Heuristic	LLM kills	Floor	HP_{opt}	Heuristic	LLM kills
gen01	168	win	18/18	gen09	204	win	18/18
gen02	232	win	15/18	gen10	88	fail	6/18
gen03	13	fail	5/19	gen11	194	fail	0/18
gen04	225	win	14/18	gen12	182	fail	8/18
gen05	172	fail	13/18	gen13	207	win	14/18
gen06	242	win	18/18	gen14	151	fail	11/18
gen07	222	win	18/18	gen15	216	win	16/18
gen08	250	fail	11/18				

6.7.3 The Tower

The tower runs use the tower prompt (Section 5.4) and the tower loop of Section 5.5.3. Table 6.13 lists the runs from MT0 with the final tower observation and, for comparison, with two earlier tower observations, together with the continuations of one stored run from MT18. The first observation offered only the actions of the current floor, the up-staircase and returns to earlier floors. The second, also used in the planning study (Section 6.6), offered every reachable object on every visited floor, a tower-wide key budget and the price of the way to an unreached shop. Both earlier observations ran with the item values of another edition of the game (Section 4.5.2), with the 40 % override of Section 5.6 and with rule 4 at 40 %; their runs are therefore engine-assisted (Section 5.6). The comparison is therefore one between whole configurations, not between observations alone.

Table 6.13: Tower runs from MT0 with three tower observations. Highest floor reached per run; ATK / DEF at the end of the best run. First and second = the two earlier observations described above, with the item values of another edition, the 40 % override, rule 4 at 40 % and the shop rule "ATK above the highest DEF in scope". Final = the observation of Section 5.5.3 (the game's item values, no override, rule 4 at 60 %, plan memory, level-ups offered, no yellow or blue keys for sale). Resumed = a stored run continued from MT18 with the final observation and blue keys for sale (see text).

Agent (observation)	Runs	Highest floor	ATK / DEF	Stopped because
heuristic, depth 3 [a]	1	MT8	113 / 82	Red Bats lethal
Qwen3.5-397B (first)	1	MT7	36 / 35	stuck; 131 EXP unspent

Agent (observation)	Runs	Highest floor	ATK / DEF	Stopped because
DeepSeek-V4-Flash (first)	5	MT7 (4), MT3	33 / 38	stuck; one run at 25 HP on MT3
Qwen3.5-397B (second)	1	MT8	81 / 117	Red Bats unaffordable
DeepSeek-V4-Flash (second)	3	MT8 (3)	92 / 106	Red Bats unaffordable
Qwen3.5-397B (final)	3	MT18, MT8 (2)	1,257 / 1,996	MT18: no blue key left; MT8: only lethal fights
DeepSeek-V4-Flash (final)	3	MT17 (2), MT8	1,346 / 1,066	MT17: upgraded monsters; MT8: Senior Mage lethal
Qwen3.6-35B (final)	3	MT16, MT8 (2)	552 / 1,133	MT16: Red Demon King unwinnable; MT8: Red Bats lethal
Mistral Medium 3.5 (final)	3	MT8, MT4 (2)	98 / 100	MT4: Skeletons lethal at 5 and 30 HP
Qwen3.5-397B (resumed)	1	**MT21, normal ending**	2,264 / 2,558	won
DeepSeek-V4-Flash (resumed)	3	MT21 (2), MT20	2,294 / 3,633	turn limit (860)

[a] The rule-based heuristic reads no prompt; it reaches MT8 with both sets of item values.

First and second observation: MT7 and MT8. With the first observation, every run stopped at or below MT7. Qwen3.5-397B bought four DEF and one ATK at the MT3 shop, DEF first as the prompt requires, and ended with 131 experience points unspent. The experience shop on MT5 lies behind a Big Bat that cost 55 % of the hero's HP. With the second observation, all four runs passed MT7, spent their experience and stopped on MT8 with ATK between 79 and 92. Only the run with ATK 92 met the shop rule of those observations, ATK above the highest DEF in scope, here the Red Bats' DEF of 90. Even at ATK 92, the Red Bats cost about 15,000 HP.

Final observation: the Red Demon King and the scripted events. With the final observation, the runs that pass MT8 meet the obstacles that the expert walkthrough describes as the second half of the game. On MT16, the Red Demon King (ATK 1,000) guards the way up behind a red door. The key-reserve line of

the final observation warns when a door off the way up would leave no red key for it, and the red door states the price of one (Section 5.5.3). The three runs that passed MT16 defeated the King. The Qwen3.6-35B run that reached MT16 could not win the fight at ATK 552 and ended stuck on MT9. Killing the King triggers a scripted upgrade of the later monsters (Section 4.5.3). After it, the monsters on MT17 need ATK above about 1,100 and cost 2,000 HP and more. Both DeepSeek-V4-Flash runs that reached MT17 stopped there, with ATK 1,048 and 1,346. One of the two had never used the first scripted event of the thief, which opens an MT2 room with eight gems, the Steel Sword and the Silver Shield. The run of Qwen3.5-397B that passed MT17 reached MT18 with ATK 1,257 and DEF 1,996. Eight of the twelve runs stop on MT8 or below, the floor where the heuristic also stops.

MT18 and the princess. In the game, the up-staircase of MT18 appears only after the hero talks to the princess. Her cell is opened by the thief of MT4 once the hero brings him the Ice Pickaxe from MT12, which lies behind a blue door. The staircase of MT20 appears when the Cross is handed in. In the final configuration, no shop sells blue keys (Section 4.5.2). The run of Qwen3.5-397B reached MT18 with no blue key left, although the event line named the blue door and the missing key. The rest of the tower was therefore tested from a stored position on MT18, with blue keys offered for sale as the original game does (50 g at the Key Seller on MT5). The position is the arrival on MT18 (turn 708, ATK 1,346, DEF 1,314) of a run of Qwen3.5-397B with a predecessor of the final observation, which lacked the purchase quantities, the vampire tag and the event lines and had used its last blue keys on three doors of MT7 about 100 turns earlier. It was continued four times with the final observation and blue keys for sale. Qwen3.5-397B bought one blue key, fetched the pickaxe, gave it to the thief and freed the princess. It then climbed to MT19, defeated the last White Warrior on MT7 for the Cross and handed the Cross in. At turn 832 it defeated the Nether Demon King on MT21, the normal ending of the game. The three DeepSeek-V4-Flash continuations reached MT20, MT21 and MT21 before the turn limit of 860, 152 turns after the resumption. From this position, one of four continuations reached the normal ending (1/4 [0.05, 0.70]).

7 Discussion and Conclusion

This chapter interprets the results of Chapter 6 in light of the research questions. Section 7.1 discusses the results one comparison at a time. Section 7.2 examines the three rival explanations stated in Chapter 1, and Section 7.3 the methodological lessons of the study. Section 7.4 states the limitations. Section 7.5 answers the research questions, Section 7.6 proposes how the work should be continued, and Section 7.7 concludes.

7.1 Discussion

7.1.1 The Observation Versus the Model

The strongest effect in this thesis is that of the observation. With model, prompt, temperature and lookahead depth fixed, two annotation lines raised Qwen3.5-397B on the simple floor from 0/5 to 10/10 [0.72, 1.00] (Table 6.5, rows 3 and 4). The two lines state the cost of reaching stat items and the effect of fight order. Every observation with fewer than two levels of lookahead or without annotations loses all its episodes: a lookahead of one step or none (0/3 each, Table 6.3), the bare observation (0/3) and the map only (0/2), the last two measured on an earlier version of the code. The rule-based heuristic, which acts on the same engine output, fails the floor as well. The language model therefore contributes something the heuristic lacks, but only once the engine has made the relevant quantities legible. The effect does not appear to be specific to the simple floor, on which the observation was developed. DeepSeek-V4-Flash wins 9 of 10 episodes [0.60, 0.98] on hand01 and hand02 and loses all ten with the bare observation (Section 6.3). On thirteen held-out generated floors, which leave more HP slack, it wins 31 of 39 [0.64, 0.89] with the annotations and 32 of 39 with the bare observation and the rules-only prompt (Sections 6.5.3 and 6.7.2). These floors therefore do not separate the two observations. The depth of lookahead that is needed appears to depend on the floor. On hand01 and hand02, DeepSeek-V4-Flash wins as often with one step of lookahead as with three (5/5 and 4/5 against 4/5 and 5/5). On the simple floor, by contrast, the stat item that decides the door choice lies beyond a one-step tree.

The error analysis shows what "legible" means. At the decisions whose reasoning was read, the model cited an annotation that stated one side of a trade-off and omitted the other (Section 6.4.2). Five kinds of this defect were identified, on the simple floor and the complex floor. They are a saving without the cost of reaching it, a value without its cost (potions counted as fight value), an efficiency without its affordability, a deficit without its scope, and a count without its multiplicity. In

each of these cases, the written reasoning cited the number it was given and argued from it. The text alone does not show that it reflects how the answer was produced [41, 81]. The stronger evidence is the intervention: with model and prompt fixed, repairing the annotation changed the choice (Table 6.5). A defect of this kind lies in the input and can be repaired there. This suggests a design rule for engines that annotate an agent's observation. A quantity should enter the observation only together with the quantity it trades against. Each new annotation should then be tested at a state where the two disagree. The repair sequence indicates why the test is needed: two of its steps introduced a new one-sided line, which the next step had to repair (Table 6.5, rows 5 and 8).

The first decision on hand01, a floor not used during development, indicates that the rule is not yet fully met. In 18 of the 20 episodes of the re-run, the model opened the right door first, after which at most 10 HP is reachable (Section 6.7.1). These are all five episodes of Qwen3.5-397B, Qwen3.6-35B and Mistral Medium 3.5 and three of DeepSeek-V4-Flash; the other two DeepSeek-V4-Flash episodes opened the left door, which keeps the optimum of 14 HP reachable. Both one-step measures in the observation rate the right door higher (Appendix G, Listing G.4). Qwen3.5-397B also reads a fight-order line as advice for that door, although the line belongs to an optional monster and names the door only as the route that makes that fight cheaper. Every model still wins the floor in most episodes, but none reaches the optimum: the wins after the right door end at 4 to 9 HP, the one win after the left door at 10 HP. With at most five episodes per model and a single decision, this should be interpreted cautiously. It suggests that the annotations were refined on the simple floor and carry some of that floor's structure.

7.1.2 The Engine Versus the Model

The second finding concerns what the engine does instead of the model. Every simple-floor win recorded before the overrides were logged, 21 of 21, contained a silent strategic override (Section 6.4.1). In 17 of them, the replacement came at turn 9, where the engine executed the heuristic's door instead of the model's choice of the Slime Lord. The early success rate was therefore the rate at which the model's error and the heuristic's replacement coincided. Reproducing that configuration gave 2/4, and both wins again contained the replacement (Table 6.5, row 1). With the overrides reduced to mechanical cases and logged, Qwen3.5-397B won 0/5 with the single-floor prompt (Table 6.5, row 2). Those early wins measured a hybrid of model and heuristic in unrecorded proportions, not the model.

The lesson is likely to apply to other agent evaluations. The agent under test is the model together with every override between the model and the environment. An override that is not logged can change the measured success rate as much

as a change of model, and it is invisible in the aggregate score. In the final configuration, all remaining overrides are mechanical (dead-end doors, unwinnable and lethal fights), each one is logged, and none of its wins contains a strategic override. The tower runs with the two earlier observations ran with a strategic override as well and are therefore reported as engine-assisted (Section 5.6).

7.1.3 Finding Versus Following an Order

The complex floor with the single-floor prompt marks the boundary of the approach. None of the four models wins any of its five episodes there. Qwen3.5-397B and DeepSeek-V4-Flash also fail it at temperatures 0.6 and 1.0 (Table 6.3). No model wins hand03 or hand04 in 20 episodes each (Table 6.2). The boundary does not follow the number of critical actions: hand01 has 13 and is won, hand03 has 8 and is not (Table 6.11). On the complex floor, the decisive choice is an order of doors and fights that pays off only over many steps. Exact state values locate the loss. Every one of the 30 failed episodes with the single-floor prompt (the 20 of the re-run and ten earlier episodes of Qwen3.5-397B, Table B.3) is lost at one fight, 29 of them at the Red Slime (11,8) (Section 6.4.5). In 28 of them, an earlier door order had already lowered the state value from 68 to 42 or 30, so little slack was left. The Red Slime costs 103 HP when it is offered at turn 9 and 11 HP where the optimal route takes it. The only fight that keeps the boss winnable at that turn, the Black Slime (12,9), costs more now (156 HP). The trap door (10,5) and the blue doors that look like the error come later. The observation carried both sides of the decisive fight, but the side that favours the right order is spread over a route that the annotation prices as unaffordable. In 29 of the 30 episodes, the models followed the low effective cost and discarded the line stating that the fight becomes free after the stat items. No per-decision analysis of the episodes was made for hand03 and hand04, where the models repeatedly chose unwinnable fights and dead-end doors (Section 6.7.1).

When the order is supplied, three of the four models win with it. With the solver-route prompt, DeepSeek-V4-Flash wins 4/5, Qwen3.5-397B 3/5 and Qwen3.6-35B 4/10 in the re-run of Table 6.2, while Mistral Medium 3.5 wins none and dies in four of five episodes; this condition runs without the mechanical overrides (Section 5.4). In the reactive arm of the planning study, DeepSeek-V4-Flash won the same condition 8/10 (Table 6.9). Every win of the re-run ends at 22 HP or below, a third of the optimum, because the prompt's door order caps the result. With the exact optimal route in the prompt, Qwen3.6-35B reaches 58 of 68 HP in 2 of 5 episodes; the missing 10 HP come from one unlisted 0-HP fight taken late (Section 6.6). Where the models leave a given route, they leave it towards the fight that is cheaper now. Ten of the 14 failures with the solver-route prompt are lost at

the same Red Slime, taken out of the prompt's order (Section 6.4.5). Three of five DeepSeek-V4-Flash episodes with the exact route take a fight shown at 43 HP instead of the listed one shown at 185 (Section 6.6). This is the same pull as with the single-floor prompt, which suggests that the models weight a cost shown now above a saving that the observation states for later. The models can follow an order of this length but do not find one. Finding it is a search problem, which the exact solver solves by expanding 729,953 states (Table 4.3).

7.1.4 Observation Design on the Tower

On the tower, the best runs with the final observation reached MT16, MT17 and MT18, against MT7 and MT8 with the first and second observations (Table 6.13). The comparison is one between whole configurations. Besides the observation, the item values of the engine, the 40 % override, the threshold of rule 4, the shop rule, plan memory, the key purchases and the level-up offer changed as well, and the contribution of each was not separated. With the second observation, every run stopped on MT8, where the heuristic also stops, with too little attack for the Red Bats. With the final observation, eight of the twelve runs still stopped on MT8 or below (Section 6.7.3). The best runs stopped at failures of resource foresight: a key or attack must be kept or built for a use that the observation shows only much later. Qwen3.5-397B reached MT18 without the blue key needed on the way to the princess who reveals the staircase, although a line of the observation named the missing key. Continued from a stored MT18 position of an earlier run of the same model with a predecessor of the final observation (Section 6.7.3), with blue keys for sale as the original game sells them, Qwen3.5-397B reached the normal ending, and three continuations by DeepSeek-V4-Flash reached MT20, MT21 and MT21 before the turn limit. A line naming the missing key therefore did not prevent the failure it describes. This resembles the pull towards the option that is cheaper now, seen on the complex floor (Section 7.1.3).

The threshold of rule 4 differs between the two prompts. At 60 % and, separately, with plan memory, Qwen3.5-397B won the simple floor in 2 of 5 episodes each, against 5 of 5 (Section 6.1.2). On a floor with a single winning order, the stricter limit and the absence of plan memory appear to be part of what keeps that order stable. That one threshold does not serve both settings suggests that a fixed share of HP is a poor proxy for "do not take a fight you cannot recover from"; ideally, the prompt would need no tuned threshold at all.

7.1.5 Prompt Optimisation Versus Annotations

The prompt optimisation study was designed to separate two explanations of why the agent succeeds: that the prompt carries the strategy, or that the annotations

do. The results favour the second explanation, with one qualification.

Starting from a prompt that states only the rules, reflection over game traces rediscovers the strategy principles that follow from the combat formula, the floor structure and the definition of the effective cost (Table 6.7). Of the forty candidates of the first study, 30 state that no HP should be paid for fights that unlock nothing, and 26 that the cheapest progress comes first. The stat-item rule appears in 37 of 40, but the rules-only prompt already contains a matching sentence. The two rules written least often are the HP limit, a design choice of this thesis, and key scarcity, which the exact solver shows; they appear in only 5 and 1 of 54 candidates. Each rediscovered principle corresponds to an annotation that the engine already prints. With the full observation, the rules-only, evolved and hand-written prompts are within the variation between repeated evaluations of one another. For Llama-3.1-8B, a byte-identical copy of the rules-only prompt scored 10 held-out kills where the original scored 8, which sets the noise floor of the comparison (Table 6.6). Without the annotations, neither the evolved nor the hand-written prompt restores what was lost. The one gain that passed a paired test was found for DeepSeek-V4-Flash (no thinking) under the bare observation (30 against 24 kills, one-sided sign test, $p = 0.035$). That test included the trial floors on which the rewrite had been selected; on the nine other floors, it does not separate the two prompts ($p = 0.31$). It came from a rewrite that describes in prose the computation behind the engine's OPTIONAL tag. This suggests that, where the observation lacks a fact, the optimiser can partly supply it as a procedure.

The limiting factor is measurement. With one episode per floor, the acceptance step compares candidates whose differences are no larger than the spread of a single prompt, so it selects on noise (Section 6.5.2). Reflection may be sample-efficient, as the authors of GEPA report [1], but evaluation in this setting is not. An episode is a long chain of decisions whose outcome often turns on one choice between options of similar annotated value (Section 6.4.3). The results suggest that, in an agent whose engine can compute the consequences of actions, strategic knowledge is better placed in the observation, where it is exact and specific to the state, than in the prompt. The optimiser remains useful as a diagnostic: its rewrites name the facts the model needed and did not see.

The second study, with a reasoning task model, points in the same direction. With a teacher that saw every option, both the (1+1) loop and the Pareto loop raised the training score (Table 6.8). Both best prompts then won fewer held-out episodes than the rules-only prompt: 26 and 24 against 32 of 39, significant only for the Pareto loop (two-sided sign test, $p = 0.008$; (1+1) loop $p = 0.07$). The population did not help, for a reason specific to deterministic episodes. At temperature 0, a rewrite that did not change a decision on a floor reproduced its episode, so all

eight candidates of the Pareto loop tied on two of the three training floors. A frontier over floors then reduces to selection on the one floor whose result varies, which is the floor most exposed to overfitting. GEPA's selection presupposes training instances on which candidates differ. With three long, deterministic training episodes, this difference has to be designed into the training set, for example with more floors or with several decision positions per floor instead of whole episodes.

Even without the validator, the (1+1) loop did not compile the complex floor's route (Section 6.5.4). The teacher saw every trace and never wrote a position, and one of its general rules forbade the one fight that keeps the floor winnable at turn 9. A trace shows what the agent did, not which decision lost the game. The solver's per-decision values (Section 6.4.5) carry the information that the teacher lacked, which makes them a candidate input for reflection on long horizons (Section 7.6).

7.1.6 Reactive Control Versus Committed Planning

On the simple floor, committing to several actions per call loses almost every episode that reactive control wins. Qwen3.5-397B wins 10/10 reactively, 1/5 when it commits to two actions and none with any longer commitment. DeepSeek-V4-Flash wins no committed-plan episode. Qwen3.8-27B at temperature 0.6 wins 3/3 reactively against 3 of 9 with commit-k plans and none with long plans (Table 6.9). The failures end alive with 287 to 572 HP left, which indicates a wrong order rather than recklessness. The two hand-designed floors that reactive play wins behave the same way: DeepSeek-V4-Flash wins 18 of 20 reactive episodes there and none of 30 committed ones.

The results are consistent with an explanation by the timing of information. The annotations that decide these floors are computed for the state in which they are printed. A step committed at one decision is chosen before the engine has printed the facts of the states it leads to. Longer plans also do not become holistic. With the whole-floor wording, re-planned plans were legal throughout in 6 of 40 cases and single-call plans in none (Table 6.10). They name monsters that are not yet reachable and fights that are not yet winnable. Asking instead for as many actions as the model can give produced shorter plans, and the single-call plans were mostly clean (3 of 5 and 4 of 5), yet none won. The failure lies in the order committed to, not in illegal steps.

On the complex floor with the solver-route prompt, the picture differs. DeepSeek-V4-Flash wins 8/10 reactively with 20.8 calls per episode and 4/5 with commit-5 plans at 5.8 calls. With re-planned plans of maximal length, it wins 5/5 at 2.4 calls. Qwen3.8-27B at temperature 0.6 wins 3/3 in every re-planned mode, and single-call plans win none for either model. At these sample sizes this is parity at about a quarter to a ninth of the cost, not an advantage. The two floors separate

two activities that "planning" conflates. Deriving an order from facts that appear state by state cannot be committed ahead. Transcribing the next few steps of an order that is already given can, as long as each plan is re-anchored to a fresh state after a few steps.

The tower adds a third case. With the second tower observation, DeepSeek-V4-Flash reaches MT8 in all three reactive runs with 86 calls on average. Its commit-2, commit-3 and commit-5 runs reach the same floor with 72, 60 and 58 calls (one run each, Table 6.9). An imperfect order inside a floor appears to cost little there, and the highest floor reached, MT8, was set by the attack needed further up, not by the planning mode. A single plan per floor is the exception: it stops on MT3 after two calls, at 5 HP. Committing to a few actions per call thus lowered the cost without changing the outcome where the order was given or the run stopped for another reason. Where the order had to be derived state by state, it changed the outcome: on the simple floor and on hand01 and hand02.

7.1.7 Model Scale and Family

Model size alone did not order the results. On the simple floor, the largest model, Qwen3.5-397B, wins 5/5 in the re-run of Table 6.2 (15 of 15 episodes of this configuration overall), against 1/5 for DeepSeek-V4-Flash and Qwen3.6-35B (whose earlier episodes won 7/10, Section 7.2) and none for Mistral Medium 3.5. On hand02, Mistral Medium 3.5, which has no reasoning channel, ends four of five episodes at the exact optimum, the highest ratio in Table 6.2. On the tower with the final observation, its best run reached MT8 against MT16 for Qwen3.6-35B, a smaller model with a reasoning channel (Table 6.13).

Size appears to affect something narrower: the tolerance for defective annotations. On the same observation, Qwen3.5-397B won 10/10 where Qwen3.6-35B won 0/5 at a reasoning cap of 8,192 tokens (Table 6.5, row 4). The unguarded order line of row 8 cost Qwen3.6-35B more than Qwen3.5-397B (1/10 against 7/10). Qwen3.6-35B also reacted to the length of the request: with six turns of conversation history it won 1/5, without history 10/10 (Table 6.5, rows 6 and 7). Whether the larger model re-derives the quantities from the lookahead tree was not tested, and the two models differ in generation as well as size. Qwen3.6-35B may thus be a more sensitive instrument for the observation's mistakes, which would make it well suited for developing annotations and less suited for reporting headline rates. The first door on hand01 did not separate the models either: 18 of 20 episodes of four models of three families opened the same door (Section 7.1.1). Section 7.2 asks whether these readings withstand the rival explanations stated in Chapter 1.

7.2 Rival Explanations

Three alternative explanations were stated in Chapter 1.

R1: Apparent effects are caused by the non-determinism of the inference service. The service is not deterministic at temperature 0: identical configurations produced both wins and losses. Three observations bound this explanation. First, every headline comparison was repeated, and for the main effects the Wilson intervals of the two conditions do not overlap (for example, 10/10 [0.72, 1.00] against 0/5 [0.00, 0.43]). Second, when an annotation made one option dominant, all fifteen episodes of Qwen3.5-397B in the final configuration followed the same trajectory (Section 6.4). Third, at temperatures 0.6 and 1.0, no repeated cell of Table 6.2 moved by more than one episode in five, and neither the order of the conditions nor the boundary changed (Section 6.3). Non-determinism appears to select among options that the model already finds close; it magnifies ambiguity in the observation rather than acting as an independent source of failure. A second, slower source of variation is the served model itself. On the same observation at turn 22, the five re-run episodes of Qwen3.6-35B split one to four (Section 6.2), and the earlier episodes of the same configuration, on an earlier version of the code, won 7/10 against 1/5 in the re-run. The failures of both sets of episodes leave the winning trajectory at the same decision, turn 22. The difference appears to lie in how often the served model takes one of two close options at that decision, not in the agent. R1 is therefore rejected for the headline effects; it cannot be excluded for differences of one or two episodes in five.

R2: Apparent gains of prompt optimisation are the luck of single episodes. This explanation holds for every gain measured on training floors. The one training gain under the full observation did not survive paired re-evaluation, and the training gains of the second study reversed on the held-out floors (Section 6.5). In the first study, the noise floor was measured directly, and the one gain that exceeded it passed a pre-registered paired test (Section 7.1.5). That gain is narrow: it rests largely on two floors that share one failure. The test is also not independent of the selection, because it includes the twelve trial episodes on the four floors chosen for the rewrite's wins, which contribute $4 : 0$ of the discordant pairs. On the nine floors not used to select the rewrite, the discordant pairs split $3 : 1$ ($p = 0.31$). R2 therefore cannot be excluded for this gain either; it is a result on a selected sample, not an independent confirmation.

R3: The engine's overrides, not the model, produce the wins. For the early simple-floor results, this explanation is correct (Section 7.1.2). For the final configuration, it is excluded: none of its wins contains a strategic override, and the mechanical overrides that remain are logged with every episode. On the tower, it applies to the runs with the two earlier observations, which ran with the 40 %

override; the final tower observation ran without it (Section 5.6).

The rival explanations concern the results; the next section turns to threats to the measurements themselves.

7.3 Methodological Lessons

The study encountered six threats to the validity of its own measurements. They were detected in different ways: some because a previously measured result failed to reproduce, others in code review, in a check of the service's model list, in a tile-by-tile check of the game data, while building new floors, or by applying the exact solver to stored episodes. Three of them changed, or would have changed, headline results: the silent strategic override, contained in every early win (Section 7.1.2), the placeholder described below, and the wrong reference optimum. Appendix E gives the dated record of eighteen incidents, including several that changed no result.

A machine-generated annotation is a parameter with discontinuous effects. Replacing a placeholder value in one annotation with the corresponding number made the observation more informative to a human reader. It lowered the success rate from 5 of 10 episodes per model to 0 of 20 episodes across two models (Appendix E, incident 1). The stored reasoning had read the placeholder as "not recommended yet". Prompt-sensitivity research reports comparable effects for wording and layout [69]; here the effect arose in the structured, machine-generated part of the prompt. The exact observation text is therefore pinned by an automated test at a reference decision.

Silent overrides and silent configuration. Two code paths, both labelled "reactive", applied different overrides to the model's choices, and all early wins came from one of them. The strategic override of Section 7.1.2 also acted unseen, as did a default that mapped a floor to the wrong prompt. A switch that turned the overrides off also hid the annotations of fights, and a stale process continued an obsolete configuration alongside the corrected one. Each of these produced results that were internally consistent and wrong (Appendix E, incidents 2, 3, 4, 7 and 8). Each result now records its loop, prompt, overrides, depth and configuration.

The reference can be wrong. The published solver reported 22 HP as the optimum of the complex floor; the exact optimum is 68 HP (Section 4.6.2). Wins at 22 HP, which matched the published value, reach a third of the exact optimum. The defect came to light during an exact re-solve while the hand-designed floors were built; a match with a reference is only as good as the reference.

The visible error is not always the decisive one. Inspection of the complex-floor traces points to the trap door at (10,5), which 17 of the 20 episodes of the

re-run opened and which the solver-route prompt warns against. The exact state values place all 30 losses earlier, at turn 9 or 12, at an innocuous-looking fight taken before the door was opened (Section 7.1.3). An error analysis by inspection finds the decision a human would question; only an exact reference finds the decision that mattered.

The engine can be more generous than the game. An earlier version of the code showed two staircases that the game hides until a scripted event. On it, one run's highest floor was overstated by three floors, MT21 instead of MT18 (Appendix E, incident 17). The engine models the game, and every unmodelled event is a place where it may be easier or harder than the original.

Models disappear. Three models used in early experiments were removed from the inference service during the study. The successor of one of them, Qwen3.6-35B, won 0 of 10 episodes on a prompt with which its predecessor had won 2 of 18 (Appendix E, incident 6). A prompt tuned to one model is a model-specific artefact.

Taken together, these incidents suggest that the main function of the evaluation platform is to make experiments falsifiable. It does so through complete episode logs, recorded configurations, pinned observation text, logged overrides and an exact reference.

7.4 Limitations

Several methodological aspects constrain the scope of the conclusions that can be drawn.

First, the study uses one game. Magic Tower isolates the ordering of irreversible decisions under exact rules, which is what makes attribution possible. Real planning problems add uncertainty, partial observability and much larger state spaces. The results should not be extrapolated to such settings without further study.

Second, sample sizes are small. The cells of the re-run of Table 6.2 use five episodes and earlier headline cells ten, a size dictated by the cost of reasoning models. Repeated episodes of large reasoning models take minutes each, and four parallel configurations slowed every call of Qwen3.5-397B from about 20 s to over 300 s (Section 6.1.3). At $N = 10$, the Wilson interval spans about ± 0.25, so differences in success rate smaller than about 0.3 are not resolved (Section 5.9.1), and several statements about model families rest on three to five episodes. The bare-observation and map-only contrasts on the simple floor use three and two episodes on an earlier version of the code (Table 6.3). The second prompt optimisation study was not pre-registered; for the evolved-route run, only three predictions

were stated in advance. The prompt rules in the evolved prompts were counted by keyword patterns, which overstate the stat-item rule because the rules-only prompt already contains a matching sentence.

Third, the observation was developed on the simple floor, and the error analysis was carried out on the same floor. The hand-designed floors and the held-out generated floors are checks beyond the development floor, and the first decision on hand01 suggests that some of the annotations' behaviour is specific to the floor on which they were refined (Section 7.1.1).

Fourth, the layouts of the four hand-designed floors were drafted by a language model (Claude Opus 5.5) and accepted by the author only after exact solving and tuning (Section 4.6.3). They were built to have a unique winning set and to defeat the heuristic, not to be representative of the game. The heuristic's failure on them is therefore a design property, not evidence for the agent.

Fifth, the only reference agents are the rule-based heuristic and random play. No stronger deterministic controller that uses the same annotations, such as a bounded search over the listed actions, was run. The results therefore show an advantage of the hybrid agent over the implemented heuristic on the floors tested, not that a language model is needed to choose among the engine's annotations.

Sixth, all models were accessed through one inference service, with its own serving stack and availability, and the served models changed during the study (Section 7.2). Results of a hosted model are a property of the model version served at the time, and exact reproduction depends on continued access to the same versions.

Seventh, decoding used temperature 0 throughout, so that all cells share one condition and a failure can be traced along a single trajectory. This is below the values that the model cards recommend for thinking mode (0.6 for Qwen3.5-397B, 1.0 for DeepSeek-V4-Flash). The DeepSeek-R1 report found that greedy decoding of long reasoning outputs raises repetition rates [14], and the truncated answers that had to be repaired (Section 5.3) are consistent with that. The repetitions at 0.6 and 1.0 changed no conclusion (Section 7.2). Nucleus and top-k sampling were not varied, and cells that were not repeated might still look different under the recommended settings.

Last, the final tower observation was developed on the same tower, each revision in response to how earlier runs had stopped. The 60 % threshold of rule 4 was chosen after earlier tower runs had stopped (Section 5.4), so no tower result is held out. Each configuration was run one to five times per model. The scripted events that affect play up to MT21 are modelled, including the hidden staircases of MT18 and MT20 and the normal ending; the dialogue-only events and those above MT21 are not (Section 4.5.3). The final tower observation withholds the

yellow and blue keys that the original game's shops sell. The normal ending was reached in a run continued from a stored position with blue keys for sale, not in a run from MT0, and the three continuations by DeepSeek-V4-Flash ended at the turn limit, 152 turns after the resumption. The comparison with the two earlier observations also spans a change in the values of nine items, which came from another edition of the game (Section 4.5.2); the single floors are not affected. The tower therefore measures how far the agent climbs under the modelled rules, not how far it could climb in the original game. Within these limits, the next section answers the research questions.

7.5 Answers to the Research Questions

This thesis investigated a hybrid agent in which a deterministic engine computes and annotates the legal actions and a language model chooses among them. It asked how the agent performs on a long-horizon game with irreversible decisions, and whether evolutionary prompt optimisation and committed planning improve it. The experimental results provide partial answers to the research questions formulated in Section 1.1.

Regarding the first research question, the engine's observation carries most of the effect. With exact annotations, the engine's assists and only mechanical overrides, Qwen3.5-397B wins the simple floor in every one of 15 episodes [0.80, 1.00]; in the re-run, the other three models win it in at most one of five (Qwen3.6-35B won 7/10 in its earlier episodes). Every one of the four models wins hand01 and hand02 in most episodes. On thirteen held-out generated floors, DeepSeek-V4-Flash wins 31 of 39 episodes [0.64, 0.89] with the annotations but also 32 of 39 without them and with the rules-only prompt, so these floors do not test the observation. The rule-based heuristic fails the simple floor, and without annotations Qwen3.5-397B fails it too (0/3), as does DeepSeek-V4-Flash on hand01 and hand02 (0/10). The step from 0/5 to 10/10 came from repairing annotations that stated one side of a trade-off, with model and prompt unchanged. On the 24-floor tower, the final observation, together with the game's item values, no 40 % override, a rule 4 threshold of 60 %, plan memory, the revised shop rule, no yellow or blue keys for sale and the level-up offer, raised the highest floor reached from MT7 with the first observation to MT18. A run continued from MT18 with blue keys for sale reached the normal ending. The boundary of the approach lies at floors whose decisive choice is a multi-step order of doors and fights. On the complex floor, every failed episode is lost at a single fight, in 29 of 30 the same cheap fight taken before the stats that make it nearly free. Three of the four models win with such an order when it is given, but no model finds it. Hypothesis HA is supported for Qwen3.5-397B on the simple floor and for DeepSeek-V4-Flash on hand01 and

hand02. Two caveats apply: the bare-observation contrast on the simple floor rests on three episodes on an earlier version of the code, and the heuristic's failure on the hand-designed floors holds by construction. For DeepSeek-V4-Flash and Mistral Medium 3.5, the evidence on the simple floor does not support HA; for Qwen3.6-35B it is undetermined (1/5 in the re-run, 7/10 in its earlier episodes).

With respect to the second research question, reflective prompt optimisation rediscovers, from a rules-only prompt, the strategy principles that follow from the combat formula and the floor structure, but rarely the HP limit and key scarcity. On an agent whose observation already carries these principles, the evolved prompts are within the variation between repeated evaluations of the hand-written and rules-only prompts. The one gain that passed a paired test arises without annotations, restates an annotation as a procedure, and is not independent of the selection of its trial floors. With a reasoning model, both the (1+1) loop and the Pareto loop overfit their training floors and win fewer held-out episodes than the rules-only prompt, significantly so for the Pareto loop (two-sided sign test, $p = 0.008$). Without the validator, evolution did not learn the complex floor's route. Across the experiments, the observation constrained success more than the prompt did. Hypothesis HB is supported, with the caveat that its test uses one episode per floor on thirteen floors and two models, so a small gain of the evolved prompts cannot be excluded.

Regarding the third research question on the planning horizon, reactive control outperforms every committed planning mode on the floors whose decisive facts appear only as states are reached: the simple floor, hand01 and hand02. For Qwen3.5-397B and DeepSeek-V4-Flash, reactive control wins 14 of 20 simple-floor episodes against 1 of 70 for committed plans, and DeepSeek-V4-Flash wins 18 of 20 against 0 of 30 on the two hand-designed floors. On single floors, committed planning reaches parity with about a quarter to a ninth of the model calls only where the order is given in the prompt. On the tower with the second observation, commit-k plans reach the same floor as reactive control with 58 to 72 calls against 86, while a single plan per floor stops on MT3. Hypothesis HC is supported where the order must be derived; where the order is given, committed planning lowered the cost without lowering success. These answers point to the experiments that a continuation should run, which the next section proposes.

7.6 Future Work: A Proposal for the Next Study

A continuation of this work should be set up as follows.

Floors and measures. Use the four hand-designed floors and the thirteen held-out generated floors as the evaluation set, keep the simple floor, gen01 and gen02

for development, and add floors that were not built to defeat the heuristic. Report success with Wilson intervals and HP relative to the exact optimum, with at least ten episodes per cell, and log every override. Add the solver's per-decision analysis (Section 4.6.3) to every failed episode. Each failure is then located at the decision after which the state value becomes none. This was done for the complex floor (Section 6.4.5) and should become part of every report.

A search component for multi-step orders. On the complex floor with the single-floor prompt, hand03 and hand04, no model found the winning order from the per-state annotations. Keep the division of labour and move the order into the engine. A bounded search from the current state proposes a small set of candidate orders of fights and doors with their exact HP cost. The model chooses among them as it now chooses among actions. The question is whether the model can choose well among legible plans, which it did not produce in this study. Run on its own, the same search is also the deterministic comparator that this study lacks (Section 7.4).

Planning that can consult the engine. The committed plans of this study are consistent with failure by timing: each step was chosen before the engine had computed the facts of the states it leads to. A planner that proposes a step, receives that state's annotated action list from the engine, and continues would test whether holistic planning fails for lack of information or for lack of ability.

Prompt optimisation with repeated evaluation. Run the Pareto loop and the (1+1) loop at the same budget with three to five episodes per candidate and floor [7]. Use several decision positions per floor as training instances instead of whole episodes (Section 7.1.5). The present study indicates that without repetition, the acceptance step cannot see gains of the size the rewrites produce. Optimise where the observation is known to be incomplete: under the bare observation, where the study found its only gain that passed a paired test (Section 6.5.2), and on hand03 and hand04, where the full observation does not suffice. Give the teacher the solver's state values for each failed episode, which the traces lack (Section 6.5.4).

The observation itself as the object of optimisation. Every gain on the simple floor came from reading failed episodes and repairing the observation. That loop can be automated: a teacher reads failed traces and proposes a new annotation, which is accepted only if the engine computes it exactly and it improves success on held-out floors. The first decision on hand01 is a ready test case.

The rest of the tower. Model the remaining scripted events of the upper floors. Run the final observation from MT0 with the yellow and blue keys for sale that the original game offers, and test it on a tower that was not used to develop it. Repeat

these runs often enough to report a distribution of the highest floor reached rather than single values.

7.7 Conclusion

Restated against the results, the three contributions of Section 1.2 are as follows. First, the thesis presents an engine observation layer for language-model agents and identifies the one-sided annotation as a class of defects in such layers. Repairing these defects took Qwen3.5-397B on the simple floor from 0/5 to 10/10 without a strategic override. Second, it provides a repeated-episode evaluation protocol with exact optima and logged overrides, documents six threats to the validity of its own measurements, and releases the platform that implements it. Third, it reports negative and boundary results, part of them stated in advance. Prompt optimisation stays within the noise floor when annotations are present. Reactive control wins 14 of 20 simple-floor episodes against 1 of 70 for committed plans. No model finds the decisive order of the complex floor from a general prompt.

The title's trajectory "from reactive control to holistic planning" describes the information that a planner would need rather than a capability that planning delivered. In this environment, the agent's decisions were rarely better than what the engine stated at the moment of decision. On the complex floor they were worse: the models discarded the line that favoured the right order. When the engine stated both sides of every trade-off, Qwen3.5-397B won the simple floor in 15 of 15 episodes without a strategic override, and the prompt mattered little. When it did not, neither a better prompt nor a longer plan recovered the loss.

References

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. arXiv preprint arXiv:2507.19457, 2025.

[2] Berk Atil, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J. Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, et al. Non-determinism of "deterministic" LLM settings. arXiv preprint arXiv:2408.04667, 2025.

[3] Nikolas Belle, Dakota Barnes, Alfonso Amayuelas, Ivan Bercovich, Xin Eric Wang, and William Wang. Agents of Change: Self-evolving LLM agents for strategic planning. arXiv preprint arXiv:2506.04651, 2025.

[4] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001. doi: 10.1214/ss/1009213286.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2005.14165.

[6] Roger Creus Castanyer, Pablo Samuel Castro, and Glen Berseth. Agentick: A unified benchmark for general sequential decision-making agents. arXiv preprint arXiv:2605.06869, 2026.

[7] Yuan Chang and Xiaoqi Chen. Naive Prompt Optimization: Rethinking the need for complex prompt search. arXiv preprint arXiv:2608.27266, 2026.

[8] Zhenpeng Chen, Chong Wang, Weisong Sun, Xuanzhe Liu, Jie M. Zhang, and Yang Liu. Promptware engineering: Software engineering for LLM prompt development. arXiv preprint arXiv:2503.02400, 2025.

[9] Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next AutoDiff: Generative optimization with rich feedback, execution traces, and LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.16218.

[10] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[11] ckcz123. magic-tower-AI: AI for the Magic Tower game. GitHub repository, 2017. URL `https://github.com/ckcz123/magic-tower-AI`.

[12] ckcz123. mota-js: HTML5 Magic Tower template. GitHub repository, version 2.10.3, 2023. URL `https://github.com/ckcz123/mota-js`.

[13] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, et al. TextWorld: A learning environment for text-based games. In *Computer Games Workshop*, 2019.

[14] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

[15] Ali Doosthosseini, Jonathan Decker, Hendrik Nolte, and Julian Kunkel. SAIA: a seamless Slurm-native solution for HPC-based services. *The Journal of Supercomputing*, 82(7):403, 2026. doi: 10.1007/s11227-026-08508-3.

[16] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, 2nd edition, 2015. doi: 10.1007/978-3-662-44874-8.

[17] Federico Errica, Giuseppe Siracusano, Davide Sanvito, and Roberto Bifulco. What did I do wrong? Quantifying LLMs' sensitivity and consistency to prompt engineering. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025. arXiv:2406.12334.

[18] Expert walkthrough. Walkthrough of the 24-floor Magic Tower. Community text in Chinese, author unknown; archived with the repository (grey literature), n.d.

[19] Michael Feffer, Ronald Xu, Yuekai Sun, and Mikhail Yurochkin. Prompt exploration with prompt regression. arXiv preprint arXiv:2405.11083, 2024.

[20] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[21] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2 (3–4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5.

[22] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[23] Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. TextArena. arXiv preprint arXiv:2504.11442, 2025.

[24] Atharva Gundawar, Karthik Valmeekam, Mudit Verma, and Subbarao Kambhampati. Robust planning with compound LLM architectures: An LLM-Modulo approach. arXiv preprint arXiv:2411.14484, 2024.

[25] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.08532.

[26] Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations (ICLR)*, 2022.

[27] Yi Han, Yan Wang, Lingfei Qian, Haohang Li, Yupeng Cao, Yueru He, Xueqing Peng, Nanhan Shen, Yitao Xu, Yankai Chen, et al. Can LLM agents be CFOs? Benchmarking long-horizon resource allocation in an uncertain enterprise environment. arXiv preprint arXiv:2603.23638, 2026.

[28] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[29] John Hartley, Conor Hamill, Devesh Batra, Dale Seddon, Ramin Okhrati, and Raad Khraishi. How personality traits shape LLM risk-taking behaviour. arXiv preprint arXiv:2503.04735, 2025.

[30] Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. Thinking Machines Lab: Connectionism, 2025. doi: 10.64434/tml.20250910.

[31] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1904.09751.

[32] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner Monologue: Embodied reasoning through planning with lan-

guage models. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2023.

[33] Dominik Jeurissen, Diego Perez-Liebana, Jeremy Gow, Duygu Cakmak, and James Kwan. Playing NetHack with LLMs: Potential and limitations as zero-shot agents. arXiv preprint arXiv:2403.00690, 2024.

[34] Jingru Jia, Zehua Yuan, Junhao Pan, Paul E. McNamara, and Deming Chen. Decision-making behavior evaluation framework for LLMs under uncertain context. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.05972.

[35] Daniel Jurafsky and James H. Martin. Speech and Language Processing. Online manuscript, 3rd edition draft, 2025. URL `https://web.stanford.edu/~jurafsky/slp3/`.

[36] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. LLMs can't plan, but can help planning in LLM-Modulo frameworks. In *Proceedings of the International Conference on Machine Learning (ICML), Position Paper Track*, 2024. arXiv:2402.01817.

[37] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations (ICLR)*, 2024.

[38] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[39] Mingze Kong, Zhiyong Wang, Yao Shu, and Zhongxiang Dai. Meta-prompt optimization for LLM-based sequential decision making. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026. arXiv:2502.00728.

[40] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack Learning Environment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[41] Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jack-

son Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. arXiv preprint arXiv:2307.13702, 2023.

[42] Yuxiang Lin, Zihan Wang, Mengyang Liu, Yuxuan Shan, Longju Bai, Junyao Zhang, Xing Jin, Boshan Chen, Jinyan Su, Xingyao Wang, Jiaxin Pei, and Manling Li. BAGEN: Are LLM agents budget-aware? arXiv preprint arXiv:2606.00198, 2026.

[43] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. arXiv preprint arXiv:2304.11477, 2023.

[44] Lihao Liu, Peng Tang, Kunwar Yashraj Singh, and Shabnam Ghadar. ESPO: Error-structured prompt optimization via diagnose, diversify, and stabilize. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026. arXiv:2609.04197.

[45] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024.

[46] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9), 2023. arXiv:2107.13586.

[47] Shiyan Liu, Qifeng Xia, Qiyun Xia, Yisheng Liu, Xinyu Yu, and Rui Qu. Reflection in the dark: Exposing and escaping the black box in reflective prompt optimization. In *Proceedings of the ACL Student Research Workshop*, 2026. arXiv:2603.18388.

[48] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

[49] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. arXiv:2104.08786.

[50] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An

analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2401.13178.

[51] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

[53] Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of LLM agents: A survey. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2025. arXiv:2507.21504.

[54] Lilian Ngweta, Kiran Kate, Jason Tsay, and Yara Rizk. Towards LLMs robustness to changes in prompt format styles. In *Proceedings of the NAACL Student Research Workshop*, 2025. arXiv:2504.06969.

[55] Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. arXiv:2406.11695.

[56] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[57] Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *International Conference on Learning Representations (ICLR)*, 2025.

[58] Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn,

et al. Orak: A foundational benchmark for training and evaluating LLM agents on diverse video games. arXiv preprint arXiv:2506.03610, 2025.

[59] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. arXiv preprint arXiv:2305.15334, 2023.

[60] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL*, 2024. arXiv:2311.05772.

[61] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[62] Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, et al. A systematic survey of automatic prompt optimization techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 33066–33098, 2025. arXiv:2502.16923.

[63] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding large language models using 3D scene graphs for scalable task planning. arXiv preprint arXiv:2307.06135, 2023.

[64] Rishav Rishav, Pushpak Pujari, and Pushpendre Rastogi. ContraPrompt: Contrastive prompt optimization via dyadic reasoning trace analysis. arXiv preprint arXiv:2604.17937, 2026.

[65] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2021.

[66] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. MiniHack the planet: A sandbox for open-ended reinforcement learning research. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[67] Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. The Prompt Report: A systematic survey of prompt engineering techniques. arXiv preprint arXiv:2406.06608, 2024.

[68] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[69] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design, or: How I learned to start worrying about prompt formatting. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.11324.

[70] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016. arXiv:1508.07909.

[71] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024.

[72] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. arXiv:1701.06538.

[73] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[74] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

[75] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270.

[76] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models.

In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[77] Mayank Singh, Vikas Yadav, and Eduardo Blanco. Error taxonomy-guided prompt optimization. arXiv preprint arXiv:2602.00997, 2026.

[78] Yifan Song, Guoyin Wang, Sujian Li, and Bill Yuchen Lin. The good, the bad, and the greedy: Evaluation of LLMs should not ignore non-determinism. arXiv preprint arXiv:2407.10457, 2024.

[79] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. arXiv:2309.02427.

[80] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[81] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.04388.

[82] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models. arXiv preprint arXiv:2305.15771, 2023.

[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1706.03762.

[84] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291, 2023.

[85] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. arXiv:2305.04091.

[86] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization.

In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.16427.

[87] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2203.11171.

[88] Yufeng Wang. Doing what they say, not what they reason: Locating the faithfulness gap in LLM agents. arXiv preprint arXiv:2606.00476, 2026.

[89] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2302.01560.

[90] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2109.01652.

[91] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[92] Xinyu Weng. Magic Tower: A novel benchmark for evaluating large language models in strategic game playing. Undergraduate research report, The Chinese University of Hong Kong, 2025. Code: `https://github.com/xx-Weng/Magic_Tower`.

[93] Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. arXiv preprint arXiv:2307.09702, 2023.

[94] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. doi: 10.1080/01621459.1927.10502953.

[95] Anna Wimbauer, Jonas Möller, Erik Imgrund, and Konrad Rieck. Fingerprinting inference systems of large language models. arXiv preprint arXiv:2605.29979, 2026.

[96] Yue Wu, Xuan Tang, Tom M. Mitchell, and Yuanzhi Li. SmartPlay: A benchmark for LLMs as intelligent agents. arXiv preprint arXiv:2310.01557, 2023.

[97] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling reasoning from observations for efficient augmented language models. arXiv preprint arXiv:2305.18323, 2023.

[98] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.

[99] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

[100] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[101] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[102] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. arXiv preprint arXiv:2406.12045, 2024.

[103] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. arXiv preprint arXiv:2406.07496, 2024.

[104] Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan, and Ofir Press. VideoGameBench: Can vision-language models complete popular video games? arXiv preprint arXiv:2505.18134, 2025.

[105] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language Agent Tree Search unifies reasoning, acting, and planning in language models. arXiv preprint arXiv:2310.04406, 2023.

[106] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

[107] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu

Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR)*, 2023.

A Prompts and Observation

This appendix reproduces the system prompts of the main results of Chapter 6, the rules-only prompt and the evolved prompt of the paired test of the prompt optimisation study, the compiled-knowledge prompts for the complex floor, the complete observation of Listing 5.1, the planning instruction, and the reflection template and validator of the prompt optimiser. The other evolved prompts (Tables 6.6 and 6.8, Sections 6.5.2 and 6.5.4) are stored with their runs (Appendix D). All texts are verbatim. The character counts are those of the text sent to the model. Appendix F lists the system prompts behind the conditions of Chapter 6 with their character counts.

A.1 The Single-Floor Prompt

Listing A.1 is the prompt of every single-floor result in Chapter 6 that names no other prompt (2,526 characters). Its combat section simplifies the rule of Section 4.2: it counts one monster strike per round. In the engine, the monster strikes one time fewer than the hero, so the prompt's formula overstates each cost by one monster hit. The exact cost is printed with every fight, so the model never needs the formula to compute a cost.

Listing A.1: The single-floor system prompt.

```
You play Magic Tower (MOTA): a deterministic, turn-based dungeon game on ONE
↪  floor. Each turn the engine lists every legal action with exact numbers; you
↪  answer with one action number.

COMBAT (exact, deterministic, turn-based):
- You and the monster trade hits in rounds; you strike first.
- Your hit = heroATK - monDEF. If heroATK <= monDEF you deal no damage and
↪  cannot win.
- Monster hit = monATK - heroDEF. If heroDEF >= monATK the fight costs 0 HP.
- rounds = ceil(monHP / your hit); total cost = rounds x monster hit. The list
↪  shows it as damage= and hpAfter=.
- Special abilities: [magic-atk] ignores your DEF; [Nx-combo] multiplies the
↪  monster's hit; [first-strike] adds one extra monster hit.

READING THE ACTION LIST:
- damage / hpAfter are exact for your current stats. [FREE if DEF>=X] tells you
↪  how much DEF makes that fight free; min-cost@maxStats tells you its cost
↪  after collecting every remaining stat item.
- The indented tree under a fight or door shows what it unlocks (items, keys,
↪  doors, further enemies). lookahead = value of what it unlocks; effCost =
↪  damage - lookahead.
- The tree is the only reliable map. Do not infer routes from coordinates ("the
↪  boss is above me, so I must go straight up") — walls and doors make that
↪  wrong; if an action does not list something in its tree, that action does
↪  not lead there.
- [DEAD END] doors unlock nothing useful. [OPTIONAL] enemies block nothing.
↪  [LETHAL] means hpAfter <= 0.

PRIORITIES:
1. Stat items first. Every +1 DEF lowers every future fight; every +1 ATK lowers
↪  the rounds. Reach stat items before paying for fights that get cheaper
↪  afterwards; the order of fights decides how much HP you keep.
2. When several fights all make progress, take the one with the lowest effCost
↪  (it already accounts for what the fight unlocks); a fight with lookahead=0
↪  that unlocks nothing new is almost never the right first move.
3. Never fight a monster that costs more than 40% of your current HP unless
↪  nothing else on the floor can still make progress.
4. HP is the only currency here. Never pay HP for a fight that unlocks nothing
↪  needed for the goal.
5. Keys are scarce: open a door only when what it unlocks is needed.

GOAL: reach the floor's objective (kill the boss if there is one) alive with as
↪  much HP as possible. Think about the whole floor before each move: which
↪  stat items are still out there, what the boss costs now versus after
↪  collecting them, and which fights are truly on the way.
```

```
Give a short reasoning, then end with: FINAL_CHOICE: <number>
```

A.2 The Tower Prompt

Listing A.2 is the prompt of the final tower observation (Section 5.5.3; 6,035 characters). The prompt of the first tower observation is kept in the repository; that of the second is stored in each run log (Table D.5). Their rule 4 states 40 % instead of 60 %, their rule 3 asks only that ATK stay above the highest DEF of a needed monster, and they lack the bullets on the way-up, treasure and event lines.

Listing A.2: The tower system prompt.

```
You play Magic Tower (MOTA): a deterministic, turn-based dungeon game across
↪  many floors. Each turn the engine lists every legal action with exact
↪  numbers; you answer with one action number. You do not have to kill every
↪  monster — the tower is designed so that some monsters are only worth
↪  fighting later, after stat upgrades.

COMBAT (exact, deterministic, turn-based):
- You and the monster trade hits in rounds; you strike first.
- Your hit = heroATK - monDEF. If heroATK <= monDEF you deal no damage and
↪  cannot win — only more ATK helps.
- Monster hit = monATK - heroDEF. If heroDEF >= monATK the fight costs 0 HP.
- rounds = ceil(monHP / your hit); total cost = rounds x monster hit. The list
↪  shows it as damage= and hpAfter=.
- Special abilities: [magic-atk] ignores your DEF; [Nx-combo] multiplies the
↪  monster's hit; [first-strike] adds one extra monster hit; [vampire] takes a
↪  fixed share of your current HP whatever your DEF, so it costs less when your
↪  HP is lower: take such fights before potions or other HP gains, not after.

READING THE ACTION LIST:
- damage / hpAfter are exact for your current stats. [FREE if DEF>=X] tells you
↪  how much DEF makes a fight free; [WINNABLE at ATK>=X] tells you how much ATK
↪  makes a [LETHAL] fight winnable.
- The indented tree under a fight or door shows what it unlocks (items, keys,
↪  doors, shops, stairs, further enemies). lookahead = value of what it
↪  unlocks; effCost = damage - lookahead.
- Every reachable object on every visited floor is listed, each floor in its own
↪  block: choosing an action on another floor walks there (stairs cost nothing)
↪  and performs it. Stairs appear only as the way up to a floor you have not
↪  visited yet. Stat items and shop upgrades show their cross-floor value
↪  (which fights elsewhere become winnable or free); the tower key budget
↪  counts keys and closed doors on all visited floors.
- [shop_buy] lists what a discovered shop sells and the price in gold or EXP; a
↪  shop found once is usable from any floor.
- The Way up line shows the cheapest route to the next unvisited floor with the
↪  exact cost of each fight on it, what each ATK or DEF purchase you can afford
↪  does to those costs, and whether the route is passable in order when the
↪  gold/EXP each fight pays is spent on ATK/DEF before the next fight. Actions
↪  on that route are marked "on the way up".
- The Treasures line lists equipment, feathers, key boxes and other one-off
↪  items still lying on visited floors, with the way there. Door and fight
↪  lines also state the gold and EXP of the 0-HP fights behind them; red doors
↪  state what a red key costs.
- Event lines show what a scripted character or item does (a character that
↪  opens a sealed area on another floor, the Cross that the fairy on MT0 turns
↪  into a stat multiplier), what it gives access to and the exact way there.
```

```
- The tree is the only reliable map. Do not infer routes from coordinates —
↪  walls and doors make that wrong; if an action does not list something in its
↪  tree, that action does not lead there.
- [DEAD END] doors unlock nothing useful. [LETHAL] means hpAfter <= 0.

PRIORITIES:
1. Scout first (except: reaching a shop you have not used yet comes before
↪  scouting, because every later purchase depends on it). If an up-staircase is
↪  available and the next floor has not been visited yet, take it before
↪  spending HP or resources on the current floor: seeing the next floor tells
↪  you which stats you need and where the cheap fights are. Come back down for
↪  specific fights and items once your stats make them cheap.
2. Stat items and stat upgrades first. Every +1 DEF lowers every future fight;
↪  every +1 ATK lowers the rounds. Prefer routes to stat items over fights that
↪  get cheaper afterwards.
3. Shops: NEVER buy HP. NEVER buy a level-up from the EXP shop. Buy DEF before
↪  ATK — except that your ATK must be high enough that the monsters you still
↪  need, at least those on the way up, cost at most 60% of your current HP
↪  (rule 4); the Way up line shows how many purchases that takes. ATK just
↪  above a monster's DEF makes it winnable but still very expensive, because
↪  the cost is rounds x monster hit and the rounds fall only as ATK rises well
↪  above the monster's DEF. When ATK is too low by that measure, buy ATK until
↪  it is not. Spend gold and EXP as soon as a useful DEF/ATK purchase is
↪  affordable. Before paying HP for a fight, check what the DEF or ATK you can
↪  buy now does to it (a fight line shows it: "costs 0 HP" or "costs X instead
↪  of Y after purchases you can make now"); if purchases make it free or
↪  cheaper, buy them first instead of paying the HP. Spend EXP before gold,
↪  because gold also buys red keys: when a red door leads to something you
↪  need, keep the gold for its key. Buy where a point of ATK/DEF is cheapest:
↪  once a shop with a better price per point is open (a shop line shows it),
↪  use the dearer shop only when you cannot afford the better one and need the
↪  upgrade now; otherwise save for the better one. Hand in the Cross only when
↪  no purchase or item you can still reach raises ATK or DEF: its gain is a
↪  third of your stats at that moment.
4. Never fight a monster that costs more than 60% of your current HP unless
↪  nothing else across all floors can still make progress and all gold/EXP is
↪  already spent.
5. Never pay HP for a fight that unlocks nothing you need (no item, key, door,
↪  shop or stairs behind it). Free fights (0 HP) are always fine — they pay
↪  gold and EXP. When several fights all make progress, take the one with the
↪  lowest effCost.
6. Keys are scarce and doors outnumber keys: open a door only when what it
↪  unlocks is needed now.

GOAL: climb as high as possible without dying. Think across floors before each
↪  move: what does the next floor need, which earlier floors hold cheap fights
↪  or items now, and what should the gold/EXP buy.

Give a short reasoning, then end with: FINAL_CHOICE: <number>
```

```
If that choice is a shop purchase you want to make several times in a row, add a
↪  line QUANTITY: <n> (at most 10); it is repeated n times while it stays
↪  affordable, without asking you again.
```

A.3 The Rules-Only Prompt

Listing A.3 is the starting prompt of the prompt optimisation study (Section 6.5): the game rules, the goal and the answer format, without strategy (826 characters).

Listing A.3: The rules-only prompt.

```
You play Magic Tower, a turn-based dungeon game on one floor. Each turn you get
↪  your state and a numbered list of the actions you can take now; you answer
↪  with one action number.

GAME RULES:
- A fight is a sequence of rounds; you hit first. Your hit = your ATK - the
↪  monster's DEF. If your ATK <= the monster's DEF you cannot damage it at all.
- The monster's hit = its ATK - your DEF; if your DEF >= its ATK it cannot hurt
↪  you. The fight ends when one side reaches 0 HP. If you reach 0 HP you lose
↪  the game.
- A door needs a key of the same colour; each key opens exactly one door and is
↪  used up.
- Walking onto an item collects it: gems raise ATK or DEF permanently, potions
↪  restore HP, keys open doors.
- HP is not restored between fights.

GOAL: kill the boss and stay alive.

End your answer with: FINAL_CHOICE: <number>
```

A.4 The Evolved Prompt of the Paired Test

Listing A.4 is the one evolved prompt that changed behaviour in the prompt optimisation study (Section 6.5.2; 3,152 characters). In the second configuration of the optimiser, the teacher wrote it for DeepSeek-V4-Flash (no thinking) under the bare observation, and it was accepted at the first iteration.

Listing A.4: The evolved prompt of the paired test of Section 6.5.2.

```
You play Magic Tower, a turn-based dungeon game on a single floor. Each turn you
↪  receive your hero state and a numbered list of legal actions (fights, door
↪  opens). You answer with one action number.

COMBAT:
- You always hit first. Your damage to a monster = your ATK - its DEF; if ATK ≤
↪  DEF you deal 0 and cannot win (the action will say "you cannot win").
- Monster damage to you = its ATK - your DEF. If your DEF ≥ its ATK, it deals 0.
- Magic attacks ignore your DEF: the displayed damage is fixed (given as
↪  [magic-atk]).
- A fight goes until one side reaches 0 HP. If you reach 0, you lose the game.
- HP never regenerates; every HP spent reduces your final score.

ITEMS & DOORS:
- Gems permanently raise ATK or DEF; potions restore HP; keys open one door each
↪  and are consumed.
- Opening a door costs no HP and may lead to items or dead ends. Only open doors
↪  that give access to an uncollected gem/potion/key or to the boss.
- Collecting an item is automatic when you step on it (shown as engine action).
↪  You never need to spend a turn to "collect".

YOUR ONLY GOAL: kill the boss and end with as much HP as possible.

CORE DECISION RULE:
- A fight is **necessary** ONLY if that monster currently blocks your path to an
↪  uncollected gem/potion/key, or to the boss.
- A door is **worth opening** ONLY if it gives access to an uncollected
↪  gem/potion/key or to the boss.
- Every other fight or door open is pure waste: it costs HP or a key and gains
↪  nothing. Do NOT take it.
- Before choosing any action, check each listed option: does it lead
↪  (immediately or after that monster/door) to a remaining useful item or the
↪  boss? If not, skip it.

WHEN MULTIPLE ACTIONS ARE OFFERED:
- If any action leads to an uncollected gem/potion/key at low HP cost and
↪  enables future survival, prefer the one that unlocks the most items for the
↪  least HP.
- If the boss is the only remaining goal AND the boss fight shows a positive
↪  HP-after, take the boss immediately — do not waste HP on any other monster
↪  first.
- If the boss is listed "you cannot win", you must improve ATK (or DEF) via
↪  remaining gems first.
```

```
- NEVER take a fight "because you have to pick something." If every offered
↪  fight is unnecessary, the least-bad choice is the one with the smallest HP
↪  loss, but only as a last resort — never when the boss is winnable or a gem
↪  is reachable.
- If offered a fight that blocks nothing and a door that leads nowhere, still
↪  avoid both; pick the option with the least HP cost only if forced.

PLANNING:
- Check what items remain on the floor (the system prompt lists their
↪  positions). Before fighting, verify the monster is on the only path to one
↪  of them; many monsters guard nothing.
- Track your ATK vs boss DEF: if your ATK ≤ boss DEF you deal 0 — you MUST find
↪  ATK gems before attempting the boss, even if it costs HP to reach them.
- After you have collected all gems/potions that are reachable without excessive
↪  HP loss, the boss becomes the only goal. Fight it as soon as you can survive
↪  with positive HP — every extra fight before the boss reduces your score.

OUTPUT:
- Give a short reasoning (2-4 sentences) then a single line: FINAL_CHOICE:
↪  <number>
```

A.5 Compiled-Knowledge Prompts for the Complex Floor

These prompts contain floor-specific knowledge and are reported only as labelled conditions (Section 5.4). Both were run without the mechanical overrides and the automatic 0-HP fights, so a lethal fight that the model chose was executed; the survival rule of the solver-route prompt addresses this. The first, the solver-route prompt of Listing A.5, lists the route of the published solver with a survival rule (5,473 characters). Its results form the "Complex, solver-route prompt" column of Table 6.2 and the complex-floor rows of Tables 6.9 and 6.10. The prompt contains statements that the combat rules and the exact solver refute. The yellow door (5,8) does not gate the boss; only the blue door (6,2) does (Appendix B.5.2). The Bat (11,4) and the Big Bat (0,6) are not dead ends; the exact optimum fights both. The prompt states that the cost of a Slime Lord never decreases; it falls as ATK and MDEF rise, and the exact optimum fights the Slime Lord (3,10). The prompt is reproduced as it was used. The second, the exact-route prompt of Listing A.6, lists the 21 decisions of the exact optimal route with the expected HP after each (2,101 characters). It does not list the two fights at 0 HP that the solver takes automatically, against the Red Slimes (9,3) and (11,0), because they are not decisions of the route (Section 6.6). Its results are reported under "Executing the exact optimum" in Section 6.6.

Listing A.5: The solver-route prompt (with the survival rule).

```
You are playing Magic Tower (MOTA) on the COMPLEX floor. Goal: defeat Boss at
↪  (6,0) with maximum HP.

CRITICAL OVERRIDE: The action list will show [SAFE], lookahead values, and
↪  effCost for ALL enemies — IGNORE these labels for enemies NOT in the
↪  allowlist below. A fight can be [SAFE] with high lookahead and STILL be
↪  FORBIDDEN because it wastes HP needed for the boss. The ONLY metric that
↪  matters is: IS THE ENEMY IN THE ALLOWLIST? If NO → DO NOT FIGHT IT
↪  regardless of how cheap or safe it appears.

=== ABSOLUTE ALLOWLIST — ONLY THESE FIGHTS ARE PERMITTED ===

You may ONLY fight enemies at these EXACT positions (no others, ever):
  (8,12) Green Slime [3x-hit] — Phase 1
  (12,9) Black Slime [armor-break] — Phase 1
  (8,2) Bat [2x-hit] — Phase 2
  (4,2) Red Bat [first-strike] — Phase 2
  (2,4) Bat [2x-hit] — Phase 3
  (11,8) Red Slime — Phase 3
  (9,1) Big Bat — Phase 4
  (11,0) Red Slime — Phase 4
  (0,2) Big Bat — Phase 4
  (6,4) Green Slime — Phase 5 ONLY (costs 3 HP at ATK≥14)
  (6,6) Green Slime — Phase 5 ONLY (costs 3 HP at ATK≥14)
  (12,12) Big Bat — Phase 5 ONLY (for Blue Potion before boss)
  (3,8) Green Slime — Phase 6 (boss-gate path, costs 3 HP)
  (6,0) Boss — LAST fight only

ANY enemy not listed above = DO NOT FIGHT. Specifically FORBIDDEN:
- ALL Slime Lords at (10,10), (3,10), (4,7) — magic-atk, HP wasted permanently
- Red Bat at (4,11) — TRAP! There are TWO Red Bats. ONLY fight the one at (4,2).
↪  Position (4,11) is in the BOTTOM-LEFT, (4,2) is in the TOP-LEFT. Fight
↪  TOP-LEFT only.
- Red Bat at (0,9) — not required
- Black Slime at (2,9) — FORBIDDEN even though it has lookahead. You do NOT need
↪  it.
- Black Slime at (0,4) — not required
- Green Slime at (10,6) — not required (even if cheap/free)
- Red Slime at (9,3) — ABSOLUTELY FORBIDDEN. This wastes 25+ HP for nothing.
- Bat at (11,4) — FORBIDDEN (dead-end, wastes 116 HP)
- Big Bat at (0,6) — FORBIDDEN (dead-end, wastes 124 HP)
- ANY enemy not in the allowlist above — even if marked "SAFE" or costs 1 HP

=== DOOR SEQUENCE (open in this EXACT order) ===
1. Yellow Door at (8,7) — FIRST (use starting yellow key)
2. Yellow Door at (8,4) — SECOND (use key from (12,10))
```

```
3. Blue Door at (6,2) — THIRD (critical for ATK gems in top-left)
4. Blue Door at (12,2) — FOURTH (DEF+1 and HP potion)
5. Yellow Door at (5,8) — FIFTH and LAST (boss-gate! use key from (0,0))

CRITICAL: There are only 3 Yellow Keys on the map. Do NOT open Yellow Door at
↪  (10,5) — it is a TRAP that wastes your last key. Save it for (5,8) which
↪  gates the boss path.

=== PHASE TRANSITIONS ===
Phase 1: Fight Green Slime@(8,12) + Black Slime@(12,9)
Phase 2: Open Yellow Door(8,4), fight Bat@(8,2), open Blue Door(6,2), fight Red
↪  Bat@(4,2)
Phase 3: Fight Bat@(2,4) — then GO EAST (through 4,3→4,2→6,2→8,4→east side). DO
↪  NOT fight any Phase 5 enemies yet!
Phase 4 STRICT ORDER (do NOT skip ahead):
  4a. Red Slime@(11,8) — go east first, cheap at ATK=13 DEF=14
  4b. Big Bat@(9,1) — then go north
  4c. Red Slime@(11,0) — cheap, unlocks potions
  4d. Big Bat@(0,2) — LAST of Phase 4 (return west; cheapest at DEF=16)
  4e. Open Blue Door@(12,2) — unlocks final DEF gem
Phase 5 (ONLY after ATK≥14 AND DEF≥16! If you fight Green Slime@(6,4) at ATK=13
↪  it costs 44 HP instead of 3 HP — DO NOT DO THIS):
  5a. Green Slime@(6,4) [costs 3 HP at ATK=14; costs 44 HP at ATK=13 — NEVER
  ↪  FIGHT BEFORE ATK=14!]
  5b. Green Slime@(6,6) [costs 3 HP at ATK=14]
  5c. Big Bat@(12,12) [costs 124 HP at ATK=14,DEF=17; costs 170 at lower stats —
  ↪  WAIT!]
Phase 6 (boss run):
  6a. Open Yellow Door@(5,8) — boss gate
  6b. Fight Green Slime@(3,8) [costs 3 HP, on boss path]
  6c. Fight Boss@(6,0) — FINAL

CRITICAL SEQUENCING: After Bat@(2,4), do NOT fight Big Bat@(0,2) immediately
↪  even though it's nearby. Go EAST first (4a-4c) to collect stat gems +
↪  potions between fights, THEN return for Big Bat@(0,2). This saves ~25 HP on
↪  the Big Bat fight and preserves HP budget for boss.

=== DECISION RULES (apply in order) ===
0. SURVIVAL GATE (overrides every other rule, including the allowlist): NEVER
↪  choose an action whose line shows [LETHAL] or hpAfter < 0. If the Boss is
↪  the only allowlisted enemy left and it is [LETHAL], you are NOT ready: first
↪  take fights that cost 0–5 HP and expose a Red/Blue Potion in their lookahead
↪  (e.g. Red Slime@(9,3) 1 HP → potion, Green Slime@(10,6) 0 HP), collect every
↪  potion still reachable, and only then re-check the Boss line. hpAfter must
↪  be > 0 before you ever pick the Boss. The allowlist tells you which fights
↪  are worth HP; it does not authorise dying.
1. Complete ALL fights in the current phase BEFORE moving to the next phase
2. Is there a door from the sequence that hasn't been opened? → OPEN IT
3. Is there an allowlist fight for the current phase? → FIGHT IT (check
↪  coordinates!)
```

```
4. If multiple allowlist enemies are available, fight the one from the EARLIEST
↪  incomplete phase
5. NEVER open Yellow Door at (10,5) — waste of your only boss-gate key
6. GATE CHECK: Before fighting Green Slime@(6,4), @(6,6), or Big Bat@(12,12) —
↪  verify your ATK≥14. If ATK<14, DO SOMETHING ELSE (go east for Phase 4
↪  fights). This single rule saves 80+ HP.
7. When two enemies share the same name (e.g. two Red Bats), ALWAYS verify the
↪  (x,y) coordinates match the allowlist

=== WHY SLIME LORDS ARE FORBIDDEN ===
Slime Lord has magic-atk: DEF IGNORED. Cost 160-235 HP, NEVER decreases.
Blue Potion behind Slime Lord@(10,10) is reachable via Big Bat@(12,12) later.

RESPONSE FORMAT:
FINAL_CHOICE: <number>
```

Listing A.6: The exact-route prompt.

```
You are playing Magic Tower (MOTA) on the COMPLEX floor. Goal: defeat the Boss
↪  at (6,0) with maximum HP.

This prompt was compiled from the EXACT solver route (final HP 68, ATK 16, DEF
↪  20, MDEF 20). It is an experiment condition, not a general strategy: it
↪  lists the fight and door decisions of the optimal route in order. Items
↪  (gems, potions, keys) are not listed: whenever an item is available, collect
↪  it before taking the next route step.

=== THE ROUTE: 21 decisions, in this exact order ===
 1. Open yellow door (5,8)
 2. Fight Green Slime (8,12)
 3. Fight Black Slime (2,9)      [armour-break]
 4. Fight Red Bat (0,9)          [first strike]
 5. Open blue door (0,10)
 6. Fight Slime Lord (3,10)      [magic attack]
 7. Fight Black Slime (12,9)     [armour-break]
 8. Open yellow door (8,7)
 9. Fight Big Bat (0,6)
10. Fight Bat (2,4)
11. Fight Red Bat (4,2)          [first strike]
12. Fight Red Slime (11,8)
13. Fight Green Slime (6,4)
14. Fight Green Slime (6,6)
15. Fight Big Bat (0,2)
16. Open blue door (6,2)
17. Open yellow door (8,4)
18. Fight Big Bat (9,1)
19. Fight Bat (11,4)
20. Fight Big Bat (12,12)
21. Fight Boss (6,0)             — LAST

=== HOW TO USE IT ===
- Each turn, find the EARLIEST step of the route that is not done yet and whose
↪  action is in the list of available actions; take it. Check the (x,y)
↪  coordinates: several monsters share a name.
- Do not fight any monster and do not open any door that is not in the route.
↪  Every fight not listed wastes HP the boss fight needs; every door not listed
↪  wastes a key the route needs.
- The expected HP after the steps is approximately: 1000 → 880 → 795 → 485 → 685
↪  → 510 → 471 → 546 → 361 → 206 → 86 → 150 → 223 → 221 → 101 → 101 → 176 → 221
↪  → 196 → 312 → 68. HP rises where potions behind a step are collected
↪  automatically.

=== SURVIVAL GATE (overrides the route) ===
```

```
NEVER choose an action whose line shows [LETHAL] or hpAfter < 0. If the next
↪  route step is lethal, you have deviated: take the cheapest available fight
↪  or door that exposes a potion or a stat gem, then re-check.

RESPONSE FORMAT:
FINAL_CHOICE: <number>
```

A.6 The Complete Observation of Listing 5.1

Listing 5.1 shortens the observation at the first decision on the simple floor. Listing A.7 gives it in full: first the static floor map, which is appended once to the system prompt, then the user message of the first decision. The header counts completed turns; the engine's key pickup is turn 1, so this decision is turn 2 (Section 5.5.1).

Listing A.7: The static floor map and the complete observation at the first decision on the simple floor (final configuration).

```
=== Floor Layout: simple (static reference) ===
Enemies:
  Bat at (3,5) | ATK=17 DEF=2 HP=100 | fight_cost=182
  Bat at (5,9) | ATK=17 DEF=2 HP=100 | fight_cost=182
  Black Slime at (3,7) | ATK=5 DEF=5 HP=45 [magic-atk] | fight_cost=55
  Black Slime at (6,4) | ATK=5 DEF=5 HP=45 [magic-atk] | fight_cost=55
  Black Slime at (9,5) | ATK=5 DEF=5 HP=45 [magic-atk] | fight_cost=55
  Boss at (5,10) | ATK=20 DEF=12 HP=50 | fight_cost=999999999
  Green Slime at (5,6) | ATK=7 DEF=3 HP=50 | fight_cost=24
  Green Slime at (9,0) | ATK=7 DEF=3 HP=50 | fight_cost=24
  Red Slime at (10,7) | ATK=11 DEF=1 HP=70 | fight_cost=56
  Red Slime at (5,3) | ATK=11 DEF=1 HP=70 | fight_cost=56
  Slime Lord at (1,9) | ATK=10 DEF=5 HP=80 | fight_cost=114
  Slime Lord at (2,1) | ATK=10 DEF=5 HP=80 | fight_cost=114
  Slime Lord at (4,2) | ATK=10 DEF=5 HP=80 | fight_cost=114
  Slime Lord at (9,9) | ATK=10 DEF=5 HP=80 | fight_cost=114
Items:
  Blue Gem (+1 DEF) (def+1) at (0,7) [tier=core_stat]
  Blue Gem (+1 DEF) (def+1) at (10,4) [tier=core_stat]
  Blue Gem (+1 DEF) (def+1) at (2,10) [tier=core_stat]
  Blue Gem (+1 DEF) (def+1) at (8,10) [tier=core_stat]
  Blue Potion (+200 HP) (hp+200) at (0,9) [tier=hp]
  Blue Potion (+200 HP) (hp+200) at (10,2) [tier=hp]
  Red Gem (+1 ATK) (atk+1) at (1,0) [tier=core_stat]
  Red Gem (+1 ATK) (atk+1) at (1,3) [tier=core_stat]
  Red Gem (+1 ATK) (atk+1) at (7,10) [tier=core_stat]
  Red Gem (+1 ATK) (atk+1) at (9,1) [tier=core_stat]
  Red Potion (+75 HP) (hp+75) at (10,6) [tier=hp]
  Red Potion (+75 HP) (hp+75) at (2,4) [tier=hp]
  Red Potion (+75 HP) (hp+75) at (7,7) [tier=hp]
  Red Potion (+75 HP) (hp+75) at (9,2) [tier=hp]
```

```
  Yellow Key (yellow_key+1) at (0,1) [tier=key]
  Yellow Key (yellow_key+1) at (0,4) [tier=key]
  Yellow Key (yellow_key+1) at (1,10) [tier=key]
  Yellow Key (yellow_key+1) at (2,8) [tier=key]
  Yellow Key (yellow_key+1) at (5,1) [tier=key]
Doors:
  yellow door at (4,7)
  yellow door at (7,0)
  yellow door at (7,5)
  yellow door at (8,0)

=== Floor: simple | Turn 1/300 ===
Hero: HP=470 (100.0% [OK]) ATK=9 DEF=4 MDEF=0 Gold=0 EXP=0
Keys: Y=1 B=0 R=0
Key budget: yellow 1 in hand + 4 on this floor = 5 for 4 closed yellow door(s)
Position: (5,1)
Enemies remaining: 14/14
Uncollected items still on floor: ATK+1@(1,0), Yellow Key@(0,1), ATK+1@(9,1),
↪  Red Potion (+75 HP)@(9,2)(+75HP), Blue Potion (+200 HP)@(10,2)(+200HP),
↪  ATK+1@(1,3), Yellow Key@(0,4), Red Potion (+75 HP)@(2,4)(+75HP),
↪  DEF+1@(10,4), Red Potion (+75 HP)@(10,6)(+75HP), DEF+1@(0,7), Red Potion
↪  (+75 HP)@(7,7)(+75HP), Yellow Key@(2,8), Blue Potion (+200
↪  HP)@(0,9)(+200HP), Yellow Key@(1,10), DEF+1@(2,10), ATK+1@(7,10),
↪  DEF+1@(8,10)
DEF threshold tip: Collect +3 more DEF gems → Green Slime ×2 become FREE (saves
↪  48 HP; +4 DEF gems remain on map)
Goal: defeat Boss while keeping maximum HP. You do NOT need to kill all enemies
↪  — only fight enemies that grant stat gems, keys, or HP needed to reach and
↪  defeat the boss. There is NO shop on this floor. Boss at (5,10): ATK=20
↪  DEF=12 HP=50 | if fought now: UNWINNABLE — need ATK≥13 to deal damage (you
↪  have ATK=9) Use LOOKAHEAD TREE to find the cheapest path to the boss.
HP budget at max stats (all 4 ATK / 4 DEF gems collected): 1170 HP available (HP
↪  + 700 potions) vs 669 HP for the 2 monsters that block the way to the boss +
↪  the boss → margin +501 HP. 11 other monsters (258 HP at max stats) block
↪  nothing on the way to the boss; they are not in this budget — fight one only
↪  if its tree shows a stat gem or potion you need, and its cost then comes off
↪  this margin.
Recent actions this floor:
  T1: Collect Yellow Key at (5,1) (HP +0→470)

Available actions:
  (Note: lookahead = total item/stat value unlocked by this action
  ↪  [HP-equivalent].
   effCost = raw damage cost minus lookahead. A 200-damage fight with
   ↪  lookahead=180 costs only 20
   effective HP. For via-chain enemies: fight cost shown with DEF threshold hint
   ↪  — consider
   scouting upstairs/getting more DEF before fighting if the cost is too high.
   min-cost@maxStats = fight cost AFTER collecting all remaining DEF/ATK gems —
   ↪  shows how much HP
```

```
  you save by collecting stat gems BEFORE this fight. Large savings = defer
   ↪  this fight.)
 1. [fight] Slime Lord at (4,2) — monATK=10 monDEF=5 [FREE if DEF≥10, need +6
  ↪  more] — damage=114 (24.3%) hpAfter=356 [MODERATE]
  ↪  lookahead=0(raw=91-gw=114) effCost=None |
  ↪  min-cost=18@maxStats(ATK=13,DEF=8) save=96HP — collect stat gems first
→ next fight: ⚔ Slime Lord@(2,1) dmg=114(24.3%), FREE@DEF≥10(+6)
  → via [Slime Lord@(2,1) [dmg=114 (24%), FREE if DEF≥10 (+6 needed)]]: Red Gem
   ↪  (+1 ATK) [STAT], Yellow Key
→ stat gems behind it (+1 ATK): 228 HP of fighting to reach them, then they
 ↪  save 22 HP on the other fights listed now → net -206 HP
→ ORDER (this Slime Lord at (4,2)): take this fight BEFORE "Fight Red Slime at
 ↪  (5,3) dmg=56" — this fight then that route: 380 HP; that route then this
 ↪  fight: 392 HP (saves 12)
 2. [fight] Red Slime at (5,3) — monATK=11 monDEF=1 [FREE if DEF≥11, need +7
  ↪  more] — damage=56 (11.9%) hpAfter=414 [SAFE] lookahead=0(raw=220-gw=261)
  ↪  effCost=None | +75 HP potions behind it (collect any time) |
  ↪  min-cost=15@maxStats(ATK=13,DEF=8) save=41HP — collect stat gems first
  ↪  [ALTERNATIVE ROUTE — everything this opens is also opened by "Fight Black
  ↪  Slime at (6,4) dmg=55" (which opens more); taking both pays twice for the
  ↪  same access]
→ next fight: ⚔ Green Slime@(5,6) dmg=24(5.1%), FREE@DEF≥7(+3), ⚔ Black
 ↪  Slime@(3,7) dmg=55(11.7%), magic-atk, ⚔ Slime Lord@(1,9) dmg=114(24.3%),
 ↪  FREE@DEF≥10(+6), ⚔ Bat@(3,5) dmg=182(38.7%), FREE@DEF≥17(+13), ⚔ Bat@(5,9)
 ↪  dmg=182(38.7%), FREE@DEF≥17(+13)
  → via [Bat@(3,5) [dmg=182 (39%), FREE if DEF≥17 (+13 needed)]]: Red Potion
   ↪  (+75 HP), Red Gem (+1 ATK) [STAT], Yellow Key
  → via [Green Slime@(5,6) [dmg=24 (5%), FREE if DEF≥7 (+3 needed)]]: yellow
   ↪  door
      → via [Green Slime@(5,6) [dmg=24 (5%), FREE if DEF≥7 (+3 needed)] →
       ↪  yellowDoor@(4,7) → Black Slime@(3,7) [dmg=55 (12%), magic-atk (DEF
       ↪  irrelevant)]]: Blue Gem (+1 DEF) [STAT], Yellow Key ×2
→ stat gems behind it (+1 ATK +1 DEF): 317 HP of fighting to reach them, then
 ↪  they save 54 HP on the other fights listed now → net -263 HP
→ ORDER (this Red Slime at (5,3)): take this fight BEFORE "Fight Slime Lord at
 ↪  (4,2) dmg=114" — this fight then that route: 206 HP; that route then this
 ↪  fight: 277 HP (saves 71)
 3. [fight] Black Slime at (6,4) — monATK=5 monDEF=5 [magic-atk] [magic-atk:
  ↪  DEF useless — only ATK+MDEF reduce cost] — damage=55 (11.7%) hpAfter=415
  ↪  [SAFE] lookahead=29(raw=345-gw=237) effCost=26 | +225 HP potions behind it
  ↪  (collect any time)
→ next fight: ⚔ Green Slime@(5,6) dmg=24(5.1%), FREE@DEF≥7(+3)
→ unlocks: yellow door
  → via [Bat@(3,5) [dmg=182 (39%), FREE if DEF≥17 (+13 needed)]]: Red Potion
   ↪  (+75 HP), Red Gem (+1 ATK) [STAT], Yellow Key
  → via [Green Slime@(5,6) [dmg=24 (5%), FREE if DEF≥7 (+3 needed)]]: yellow
   ↪  door
  → via [yellowDoor@(7,5)]: Red Potion (+75 HP)
```

```
    → via [yellowDoor@(7,5) → Black Slime@(9,5) [dmg=55 (12%), magic-atk (DEF
    ↪  irrelevant)]]: Blue Gem (+1 DEF) [STAT], Red Potion (+75 HP)
      → via [Green Slime@(5,6) [dmg=24 (5%), FREE if DEF≥7 (+3 needed)] →
      ↪  yellowDoor@(4,7) → Black Slime@(3,7) [dmg=55 (12%), magic-atk (DEF
      ↪  irrelevant)]]: Blue Gem (+1 DEF) [STAT], Yellow Key ×2
→ stat gems behind it (+1 ATK +2 DEF): 371 HP of fighting to reach them, then
↪  they save 75 HP on the other fights listed now → net -296 HP
→ ORDER (this Black Slime at (6,4)): take this fight BEFORE "Fight Slime Lord
↪  at (4,2) dmg=114" — this fight then that route: 175 HP; that route then
↪  this fight: 268 HP (saves 93)
→ keys: behind this fight none; it exposes yellow door (7,5) → none behind it
 4. [open_door] Yellow Door at (7,0) [DEAD END - DO NOT OPEN]

Choose the best action. After your analysis, end your response with:
FINAL_CHOICE: <number>
```

The fixed explanatory note at the top of the action list mentions scouting upstairs, which does not apply to single floors. The note is identical in every observation. It was left unchanged so that every reported episode received the same text; the regression test of Section 5.8 pins it.

A.7 The Planning Instruction

In the committed planning modes (Section 5.5.2), the engine appends one paragraph to the user message. For commit-k with $k \leq 10$, the scope reads "exactly the next $k-1$ fight / door / staircase action(s) you will take after it". In the long-plan modes, the whole-floor wording reads "your plan for the rest of this floor: every fight, door and staircase you will take after it, in order, up to the goal (at most 59)". The max-steps wording reads "as many of the fight / door / staircase actions you will take after it as you can, in order (at most 59)". Listing A.8 shows the paragraph with the max-steps wording.

Listing A.8: The planning instruction appended to the user message (long plan, max-steps wording).

```
Then, on the next line, commit to as many of the fight / door / staircase
↪  actions you will take after it as you can, in order (at most 59). Do not
↪  repeat the FINAL_CHOICE action itself; the first NEXT entry is the decision
↪  AFTER it:
NEXT: <type> (x,y); <type> (x,y); ...
(type = fight / open_door / use_stairs, with the target position; do not list
↪  items, they are collected automatically). They are executed without asking
↪  you again as long as each is still available; free items are picked up in
↪  between.
```

A.8 The Reflection Template and the Validator

Listing A.9 shows the request that the teacher receives at each iteration of the prompt optimiser (Section 5.7), in the single-floor form used for the study of Section 6.5. Two cases use a different request. The merge step of the Pareto loop gives the teacher two prompts and their per-floor outcomes instead. The validator-off run of Section 6.5.4 replaces the rules for the prompt by permission to use positions, orders and conditions. The placeholders in angle brackets are filled with the current prompt, the episode traces and up to two earlier rewrites that scored worse. Under the full observation, the second paragraph lists the annotations; under the bare observation, it states that the agent sees only the raw facts. In the study, the teacher was given no example prompt rules, so the list of allowed content ends with “general strategy rules that you derive from the traces”.

Listing A.9: The reflection template (single floor, full observation, no example rules).

```
You are improving the SYSTEM PROMPT of an LLM agent that plays Magic Tower
↪  (MOTA), a deterministic
turn-based dungeon game. This is SINGLE-FLOOR mode: one floor, the goal is to
↪  kill the boss alive with as much HP as possible. There are NO shops, NO
↪  gold/EXP purchases and NO stairs on these floors.

Each turn the engine gives the agent the exact state and a numbered list of
↪  every legal action with
exact annotations (damage, hpAfter, [FREE if DEF>=X], min-cost@maxStats, an
↪  unlock tree, lookahead,
effCost, stat-gem net value, ORDER lines comparing fight orders, key budget /
↪  key lines, [DEAD END],
[OPTIONAL], [LETHAL]). The agent answers with one action number: FINAL_CHOICE:
↪  <number>.

RULES FOR THE PROMPT YOU WRITE (binding — a prompt that breaks them is
↪  rejected):
- It must work on ANY floor of this kind, not only the one in the trace. So it
↪  must NOT contain:
  coordinates, floor names/ids, monster or item names, positions, lists of
  ↪  fights/doors to take or
  avoid, step-by-step routes, allowlists/forbidden lists, or numbers taken from
  ↪  this floor
  (HP thresholds, costs, counts). Percentages of current HP and the combat
  ↪  formula are fine.
- Allowed content: the exact combat calculation and special abilities; how to
↪  read what the engine
  shows; general strategy rules that you derive from the traces.
- Do NOT mention shops, gold, EXP or stairs: single floors have none.
- Keep the output contract: short reasoning, then "FINAL_CHOICE: <number>".
- Keep what worked; change or add general rules only for failures you can see in
↪  the trace.
- Length: at most ~3000 characters.
<worse_attempts>
CURRENT PROMPT:
```
<curr_param>
```

EPISODE TRACES (per LLM decision: turn, chosen action, hero after, engine
↪  overrides, end of the
agent's reasoning; then the outcome):
```
<side_info>
```
```

```
First diagnose in 3-6 bullet points which decisions lost HP or the game and
↪  which general principle
the agent misapplied (name the annotation it misread, not the position). Then
↪  give the full improved
prompt inside ``` fences.
```

Each episode trace lists every decision of the model. For each, it gives the turn, the chosen action, the hero's HP, ATK and DEF afterwards, any engine override with the model's original choice, and the last 350 characters of the reasoning. The second configuration shows 700 characters and also lists the options offered at each decision. Decisions taken by the engine are listed in one line each. The trace ends with the outcome: whether the boss was killed, the final HP and its share of the exact optimum, the final attributes, the goal score, and whether the hero died.

The validator rejects a candidate if any of the following holds:

- it contains coordinates of the form (x, y);
- it names a tower floor (MT followed by a number);
- it contains the words "ALLOW", "FORBIDDEN", "STEP 1", "walk-through" or "allowlist";
- it names any monster of the game;
- it contains a number of three or more digits;
- on a single floor, it mentions shops, gold, EXP or stairs;
- it lacks the output contract `FINAL_CHOICE`;
- it is longer than 4,500 characters.

A rejected candidate is returned to the teacher once, with the reasons appended to the same request. If the second candidate is also rejected, no candidate is produced in that iteration, and the best prompt so far is evaluated again (Algorithm 5.2).

B Combat Rules, Exact Solver and Floors

This appendix gives the full damage algorithm of Section 4.2 and the internals of the exact solver of Section 4.6. It then describes how the generated and hand-designed floors were built and gives the maps and monster tables of the six fixed single floors. Section B.6 lists where each complex-floor episode was lost (Table B.3), and Section B.7 lists the scripted events of the tower (Table B.4).

B.1 The Damage Algorithm

Algorithm B.1 is the damage computation of the engine. It follows the combat rules of the game; on the six fixed single floors, it gives the same cost for every fight as the published solver.

Algorithm B.1: Damage of one fight. Ability codes follow the game data (1 first strike, 2 magic attack, 3 solid, 4 double, 5 triple, 6 N-strike, 7 break armour, 8 counter attack, 9 purify, 10 mimic, 11 vampire, 22 fixed damage).

Require: hero (HP, ATK, DEF, MDEF), monster (HP, ATK, DEF, abilities, n, vampire ratio, fixed damage)
Ensure: damage D, HP after, whether the fight can be won

1 **if** mimic **then** monster ATK, DEF ← hero ATK, DEF
2 **if** solid and monster DEF < hero ATK − 1 **then** monster DEF ← hero ATK − 1
3 h ← hero ATK − monster DEF ▷ hero damage per strike
4 **if** h ≤ 0 **then return** unwinnable
5 T ← ceil(monster HP / h) ▷ hero strikes
6 **if** magic attack **then** d ← monster ATK **else** d ← max(0, monster ATK − hero DEF)
7 **if** double **then** d ← 2d **else if** triple **then** d ← 3d **else if** N-strike **then** d ← n · d (n = 4 by default)
8 I ← 0 ▷ damage before the fight
9 **if** first strike **then** I ← I + d
10 **if** vampire **then** I ← I + floor(hero HP · vampire ratio)
11 **if** break armour **then** I ← I + floor(0.9 · hero DEF)
12 **if** purify **then** I ← I + 3 · hero MDEF
13 c ← floor(0.1 · hero ATK) **if** counter attack **else** 0
14 D ← I + (d + c) · (T − 1) ▷ the monster strikes T − 1 times
15 **if** fixed damage **then** D ← D + fixed amount
16 D ← max(0, D − hero MDEF)
17 **return** D, hero HP − D, (hero HP − D > 0)

The engine treats a fight with $D \geq$ HP as lethal. If the model chooses it while a

non-lethal action exists, the mechanical override replaces the choice by the action of the rule-based heuristic. If only lethal or unwinnable fights remain, the episode ends as stuck (Section 5.6). The two route-prompt conditions run without the mechanical overrides, so a lethal fight chosen there while another action exists is executed (Section 5.4). On the floors of this thesis, only magic attack, first strike, double and triple attack, the four-strike attack, break armour, fixed damage and vampire occur (Table 4.1).

B.2 The Exact Solver

The solver was published by ckcz123 [11] in Java. All optima in this thesis come from a copy with the correction of Section 4.6.2; the published solver is kept unchanged. The engine of this thesis reads the solver's floor files, so the agent, the solver and the reference agents play identical floors.

B.2.1 Graph Construction and Merging

Every tile that is not floor or wall, and the hero's start tile, becomes a node. Two nodes are joined when the hero can walk between them over floor tiles without passing another node (Section 4.6.1). Two neighbouring nodes are merged when both hold only items, because collecting one always exposes the other. Under a stricter condition, two neighbouring obstacles are merged as well. Neither may be the boss, they may share no neighbour, and all their other neighbours must be joined to one another, so that merging hides no choice. A node that holds an item is never merged with a door or monster, because that would hide the cost of the obstacle.

B.2.2 Published and Corrected Dominance Test

The published search performs two tests on the *dominance key* of a state, the set of cleared door and monster nodes. When a successor is added to the queue, it is discarded if a state with the same key and at least as much HP has been seen. When a state is taken from the queue, a set of keys already expanded is consulted, and the state is discarded if its key is in the set, whatever its HP. Because the queue orders by the number of decisions first, a state with more HP that reaches the same key in more decisions is taken later and discarded. The corrected search replaces the second test by the HP comparison (line 6 of Algorithm 4.1). Table B.1 compares the two versions.

Table B.1: Published and corrected solver. Winning sets are counted by the corrected version only. On the complex floor, the published version reports 13 winning routes, the best of which ends at 22 HP. States = states expanded. Time = search time reported by each version in one run, with both versions run one after the other on the same laptop CPU (Intel Core i5-1135G7, Java 26).

Floor	Version	Optimum HP (ATK, DEF, MDEF)	Winning sets	States	Time
Simple	published	13 (13, 8, 0)	—	412	0.12 s
Simple	corrected	13 (13, 8, 0)	1	412	0.10 s
Complex	published	22 (14, 17, 20)	—	703,456	38 s
Complex	corrected	68 (16, 20, 20)	32	729,953	42 s

The replay of the 68-HP route and the agreement with an independent Python reimplementation, which prunes by HP only, are reported in Section 4.6.2. The two multi-floor test maps that ship with the solver have no solution within 45 decisions and are not used.

B.2.3 Per-Decision Analysis

The per-decision analysis of Section 4.6.3 writes the state after each legal action out as a new floor file and solves that floor. In that file, the cleared tiles are removed and the hero starts with the current attributes. The optimum of that floor is the *state value* of the successor state; it is none when the boss can no longer be defeated. Critical and forced actions are defined in Section 4.5.1. The analysis of hand03 (118,351 states) takes about 70 s.

B.3 Generated Floors

One floor is produced per integer seed, deterministically, in six steps:

1. **Rooms.** An 11×11 grid of walls holds nine 3×3 rooms, separated by the wall lines at $x = 3, 7$ and $y = 3, 7$.
2. **Topology.** The hero starts in a random room of the bottom row and the boss is placed in a random room of the top row. A random spanning tree joins all nine rooms, and one or two further links between neighbouring rooms add alternative routes.
3. **Links.** Each link is one tile in the wall between two rooms. A link into the boss room is always a Slime Lord or a Bat, so the boss is never reached for free. Every other link is a yellow door with probability 0.35, a monster with probability 0.45, and open floor otherwise.

4. **Contents.** No item or extra monster is placed in the boss room. Five or six ATK gems, one or two blue potions and three to five further monsters are placed outside the start room. Three to five DEF gems and three to five red potions may lie in any other room. There is one yellow key per door, or at random one fewer, but at least one.
5. **Tables.** Monsters, item values and the hero's start (470 HP, ATK 9, DEF 4) are those of the simple floor, so only the layout is new.
6. **Acceptance.** The floor is solved exactly and kept only if the boss can be defeated with an optimum between 1 and 250 HP; otherwise the next seed is tried.

The first five floors were drawn from seeds 100 onwards and the other ten from seeds 300 onwards, before any prompt was evaluated on them. All fifteen were re-solved with the exact solver, and no optimum changed. Table 6.12 lists their optima.

B.4 Hand-Designed Floors

Each hand-designed floor was drawn around one difficulty mechanism observed on the simple and complex floors (Table B.2). The layouts were drafted by Claude Opus 5.5 as text grids, together with a monster table and the hero's start, and each version was accepted only after the loop of Section 4.6.3.

Table B.2: Design mechanisms of the hand-designed floors.

Floor	Mechanism	Size
hand01	Key ledger: four yellow doors and one blue door, with the keys inside the rooms they help to reach; every side room can be entered either through a door (costs a key) or past a monster (costs HP).	11 × 11
hand02	Defence thresholds: guards become free at DEF 14, the Skeleton needs ATK; two yellow keys for four doors.	11 × 11
hand03	The complex floor's special abilities (double and triple attack, magic attack, break armour, first strike, four-strike boss) on a new layout; green gems add MDEF.	13 × 13
hand04	Six yellow doors for two keys, a costly shortcut past a Bat, and the boss behind the only blue door.	11 × 11

The first versions and the checks of the final floors against the reference agents are reported in Section 4.6.3.

B.5 Maps and Monster Tables

The following grids show the six fixed single floors as text: `@` hero, `B` boss, `#` wall, `.` empty floor, letters for doors, keys and items as listed under each grid. Coordinates are (x, y) with the origin at the top left, as in the observation. Digits are monster types, numbered per floor in the order of the floor's monster table. Monster names are those that the engine shows to the model; on the complex floor and hand03 they label monster types of the solver's test map and do not match the game's monsters of the same name.

B.5.1 Simple Floor

The map is drawn with the game's tiles in Figure 4.1.

```
     x = 0 1 2 3 4 5 6 7 8 9 10
y =  0   . a # # # @ . Y Y 1 .
y =  1   k . 4 . # k # # # a .
y =  2   # # # . 4 . . . # p P
y =  3   # a # # # 2 . # # # #
y =  4   k . p # # . 3 . # # d
y =  5   . . . 5 . . # Y . 3 .
y =  6   # # # # # 1 # . # . p
y =  7   d # # 3 Y . # p # # 2
y =  8   . # k . # . # # # # .
y =  9   P 4 . # # 5 # . . 4 .
y = 10   # k d # # B # a d . #
```

Start: 470 HP, ATK 9, DEF 4, no keys. `Y` yellow door, `k` yellow key, `a` red gem (+1 ATK), `d` blue gem (+1 DEF), `p` red potion (+75 HP), `P` blue potion (+200 HP).

Symbol	Monster	HP	ATK	DEF	Ability
1	Green Slime	50	7	3	–
2	Red Slime	70	11	1	–
3	Black Slime	45	5	5	magic attack
4	Slime Lord	80	10	5	–
5	Bat	100	17	2	–
B	Boss	50	20	12	–

B.5.2 Complex Floor

Figure B.1 shows the floor with the game's tiles.

```
     x = 0 1 2 3 4 5 6 7 8 9 10 11 12
y =  0   k K # g d # B # p d .  2  p
```



Figure B.1: The complex floor, drawn with the game's own tiles; positions as in Figure 4.1. The tiles show monster types by their engine names; their attributes are those of the table below, not those of the game's monsters of the same name. (1) the yellow door (8,7), which the single-floor episodes open early; (2) the Red Slime (11,8), the decision that lost most complex-floor episodes (Table B.3). Blue doors and keys are drawn in blue, green gems give +5 MDEF.

```
y =  1   . a b p . # a # # 6 #  #  g
y =  2   6 # # # 7 . b . 5 . #  #  b
y =  3   . d . # . # # # . 2 p  #  d
y =  4   3 # 5 . d . 1 # Y # .  5  p
y =  5   . a . # b # p # d # Y  #  #
y =  6   6 # # # g . 1 # p # 1  .  d
y =  7   . p . # 4 # # # Y # #  .  p
y =  8   # # . 1 . Y . k . . #  2  #
y =  9   7 d 3 . . # K # # . .  .  3
y = 10   b # # 4 # # . # a # 4  #  k
y = 11   . P # . 7 . . # g # .  #  p
y = 12   a a # d d # @ . 1 # P  .  6
```

Start: 1,000 HP, ATK 10, DEF 10, no keys. `Y` yellow door, `b` blue door, `k` yellow key, `K` blue key, `a` red gem (+1 ATK), `d` blue gem (+1 DEF), `g` green gem (+5 MDEF), `p` red potion (+75 HP), `P` blue potion (+200 HP).

Symbol	Monster	HP	ATK	DEF	Ability
1	Green Slime	42	20	0	triple attack
2	Red Slime	50	20	7	–
3	Black Slime	32	49	4	break armour
4	Slime Lord	90	30	0	magic attack
5	Bat	63	34	1	double attack
6	Big Bat	120	33	2	–
7	Red Bat	54	56	3	first strike
B	Boss	178	26	0	four-strike attack

The floor holds three yellow keys for four yellow doors and two blue keys for five blue doors. Only the blue door (6,2) gates the boss: with every other door open and this one closed, the boss cannot be reached. The exact optimum (68 HP, ATK 16, DEF 20, MDEF 20) opens five doors in the order yellow (5,8), blue (0,10), yellow (8,7), blue (6,2) and yellow (8,4). It never opens the yellow door (10,5) or the blue doors (2,1), (4,5) and (12,2). It fights 15 of the 24 monsters besides the boss and still collects all 20 gems. The nine monsters it leaves are the Bat (8,2), the Black Slime (0,4), the Green Slimes (3,8) and (10,6), the Red Bat (4,11), the Red Slimes (9,3) and (11,0), and the Slime Lords (4,7) and (10,10). The published solver's route, which the solver-route prompt lists, opens yellow (8,7), yellow (8,4), blue (6,2), blue (12,2) and yellow (5,8). Opening (8,7) first and (8,4) second lowers the state value to 42 and then 22 HP.

B.5.3 hand01

Figure B.2 shows the floor with the game's tiles and the three actions of the first decision discussed in Section 6.7.1.

```
     x = 0 1 2 3 4 5 6 7 8 9 10
y =  0   a . 4 # . B . # 4 . d
y =  1   . a . # . 6 . # . p .
y =  2   # Y # # # b # # # Y #
y =  3   . 1 . # . . . # . 2 .
y =  4   k . d # . 3 . # a . k
y =  5   # 2 # # . . . # # 1 #
y =  6   a 2 . . # . # . . 1 .
y =  7   . a # . # . # . # d .
y =  8   . . 5 . Y . Y . 3 . .
```



Figure B.2: hand01, drawn with the game's own tiles; positions as in Figure 4.1. The hero starts at (5,10) with one yellow key. The marks are the actions of the first decision, numbered as in Listing G.4: (1) the yellow door (4,8) to the left wing, (2) the Black Slime (5,4) in the central room, which blocks nothing, and (3) the yellow door (6,8) to the right wing, which holds the blue key (8,10) for the blue door (5,2) in front of the Big Bat and the boss.

```
y =  9   # # . . # . # . . # #
y = 10   p . k . # @ # . K . p
```

Start: 400 HP, ATK 9, DEF 4, one yellow key. `Y` yellow door, `b` blue door, `k` yellow key, `K` blue key, `a` red gem (+1 ATK), `d` blue gem (+1 DEF), `p` red potion (+75 HP).

Symbol	Monster	HP	ATK	DEF	Ability
1	Green Slime	50	7	3	–
2	Red Slime	70	11	1	–

Symbol	Monster	HP	ATK	DEF	Ability
3	Black Slime	45	5	5	magic attack
4	Slime Lord	80	10	5	–
5	Bat	100	17	2	–
6	Big Bat	120	20	4	first strike
B	Boss	50	20	12	–

B.5.4 hand02

```
     x = 0 1 2 3 4 5 6 7 8 9 10
y =  0   . . . # a . d # d . d
y =  1   . B . 5 . P . Y . a .
y =  2   . . . # . . . # . . .
y =  3   # 3 # # # b # # # 4 #
y =  4   a . p # . K . # d . p
y =  5   . 1 . Y . 2 . 5 . 5 .
y =  6   . . . # . . . # . . .
y =  7   # 5 # # # 2 # # # Y #
y =  8   k . . # a . . # d . k
y =  9   . @ . Y . 2 . 1 . 1 .
y = 10   p . . # . . . # . . .
```

Start: 62 HP, ATK 9, DEF 4, no keys. `Y` yellow door, `b` blue door, `k` yellow key, `K` blue key, `a` red gem (+1 ATK), `d` blue gem (+1 DEF), `p` red potion (+75 HP), `P` blue potion (+200 HP).

Symbol	Monster	HP	ATK	DEF	Ability
1	Green Slime	50	7	3	–
2	Black Slime	45	5	5	magic attack
3	Bat	100	17	2	–
4	Skeleton	40	12	8	first strike
5	Yellow Guard	60	14	6	–
B	Boss	100	18	10	–

B.5.5 hand03

```
     x = 0 1 2 3 4 5 6 7 8 9 10 11 12
y =  0   B 6 . . # a . d # . g  .  P
y =  1   . . 7 . . . 1 . b . #  .  .
y =  2   # # # # Y # # # . # #  2  #
y =  3   d . 3 . . . # . 4 . #  .  a
y =  4   . # . # . # k # . # .  #  .
y =  5   a # g # . 2 . . . Y .  .  d
```

```
y =  6   . # . # Y # . # . # #  #  5
y =  7   5 . 1 . # . @ . # p .  1  .
y =  8   # Y # # # . . . # # #  Y  #
y =  9   . d . # 4 . K . 4 # .  a  .
y = 10   . p . # . . . . . # .  g  .
y = 11   a . k # . Y # Y . # k  .  d
y = 12   . . 3 . . p # p . . 7  .  .
```

Start: 649 HP, ATK 10, DEF 10, no keys. Symbols as on the complex floor; the monster table is that of the complex floor.

B.5.6 hand04

```
     x = 0 1 2 3 4 5 6 7 8 9 10
y =  0   a . 6 . # B # . 3 . d
y =  1   # Y # . # b # . # Y #
y =  2   k . 2 . . . Y . . . p
y =  3   # # # 4 # # # 1 # # #
y =  4   d . p . . # . . a . k
y =  5   # 3 # # # # Y # # 2 #
y =  6   . . a . p . . . . . 4
y =  7   # Y # # # 5 # # # Y #
y =  8   p . . 2 . @ . 1 . . d
y =  9   # . # # # # # # # . #
y = 10   K . 4 . a . d . 3 . P
```

Start: 94 HP, ATK 9, DEF 4, no keys. Symbols as on hand02.

Symbol	Monster	HP	ATK	DEF	Ability
1	Green Slime	50	7	3	–
2	Red Slime	70	11	1	–
3	Black Slime	45	5	5	magic attack
4	Slime Lord	80	10	5	–
5	Bat	100	17	2	–
6	Skeleton	60	15	7	–
B	Boss	60	22	10	–

B.6 Where the Complex-Floor Episodes Are Lost

Table B.3 lists, per configuration, the decision after which the boss could no longer be defeated in each complex-floor episode. It covers the episodes of the re-run of Table 6.2 and two earlier Qwen3.5-397B configurations with the single-floor prompt (Section 6.4.5). The per-decision analysis of Section B.2.3 gives the state value after every decision of an episode. A decision lowers the state value by the final HP

it costs, and the decision after which the state value becomes none is the decision that lost the episode. Figure 6.2 in Chapter 6 plots the state value along every episode of Table B.3.

Table B.3: The decision after which the boss of the complex floor can no longer be defeated, per configuration. Exact state values along every episode of the re-run of Table 6.2 and of two earlier Qwen3.5-397B configurations (Section 6.4.5). Turn = game turn. Earlier value losses = decisions that lowered the state value before it.

Configuration	Episodes	Wins	Decision that lost the episode	Earlier value losses
Single-floor prompt, DeepSeek-V4-Flash	5	0	Red Slime (11,8), turn 9 (5)	yellow door (8,7): 68 → 42 (4) or 30 (1)
Single-floor prompt, Mistral Medium 3.5	5	0	Red Slime (11,8), turn 9 (5)	yellow door (8,7): 68 → 30 (5)
Single-floor prompt, Qwen3.5-397B	5	0	Red Slime (11,8), turn 9 (4); Red Bat (4,11), turn 9 (1)	yellow door (8,7): 68 → 42 (3) or 30 (2)
Single-floor prompt, Qwen3.6-35B	5	0	Red Slime (11,8), turn 9 (5)	yellow door (8,7): 68 → 30 (5)
Single-floor prompt, Qwen3.5-397B, earlier configurations (Table 6.5, rows 5 and 6)	10	0	Red Slime (11,8), turn 9 (8) or 12 (2)	yellow door (8,7): 68 → 42 (6) or 30 (2); Black Slime (12,9) at turn 6: 68 → 26 (2)
Solver-route prompt, DeepSeek-V4-Flash	5	4	Red Slime (11,8), turn 12 (1)	yellow doors (8,7), (8,4): 68 → 42 → 22
Solver-route prompt, Mistral Medium 3.5	5	0	Red Slime (11,8), turn 16 (5)	same
Solver-route prompt, Qwen3.5-397B	5	3	Red Slime (11,8), turn 13 (1); Big Bat (9,1), turn 24 (1)	same
Solver-route prompt, Qwen3.6-35B	10	4	Red Slime (11,8), turn 12 (3); Big Bat (9,1), turn 24 (1); Big Bat (0,6), turns 47–49 (2)	same, and Green Slimes (6,4), (6,6) at turns 37–39

The traces and the scripts that compute them are listed in Table D.4.

B.7 Scripted Events of the Tower

Table B.4 lists the scripted events of the tower summarised in Section 4.5.3.

Table B.4: Scripted events of the tower, from the floor files of the game data. Modelled = whether the engine of the final tower observation models the scripted event.

Floor	Tile	Trigger	Effect	Modelled
MT0	fairy (6,9)	talk	gives one yellow, one blue and one red key; the fairy moves to (5,9)	yes (keys granted at the start)
MT0	fairy (5,9)	talk, holding the Cross	HP, ATK and DEF increased by a third, once; also opens the way from MT20 to MT21	yes (stat increase)
MT1	(2,11)	pick up	monster manual; no effect on play	not needed
MT2	old man (8,11), trader (10,11)	talk	Steel Sword (+70 ATK), Silver Shield (+30 DEF)	yes
MT4	thief (6,1)	talk	opens the sealed passage (2,7) on MT2 to the two MT2 characters	yes
MT4	thief (6,1)	talk again, holding the ice pickaxe of MT12	clears (6,9) and (6,10) on MT18	yes
MT5, MT14	key box (1,1), (4,1)	pick up	one yellow, one blue and one red key	yes
MT6	Small Feather (1,1)	pick up	one level: +1,000 HP, +10 ATK, +10 DEF	yes
MT7	Cross (6,5), behind a seal	pick up	Cross for the MT0 fairy; also removes the old man on MT16	yes (seal, Cross); removal not modelled

Floor	Tile	Trigger	Effect	Modelled
MT9	(1,1)	pick up	floor-flying item; walking between visited floors is already free in the engine	not needed
MT10	ice gates	step on	open without a key	yes
MT12	ice pickaxe (11,1)	pick up	needed for the second MT4 dialogue	yes (item only)
MT13	wizard (8,11)	talk	Big Feather: three levels, +3,000 HP, +30 ATK, +30 DEF	yes
MT14	Holy Water (6,4)	pick up	doubles the current HP	yes
MT15	old man (5,4)	talk, with at least 500 EXP	Holy Sword, +120 ATK, for 500 EXP	yes
MT15	trader (7,4)	talk, with at least 500 gold	Holy Shield, +120 DEF, for 500 gold	yes
MT16	old man (5,5)	talk	a story item for the MT0 fairy (needed only for the ending); no effect on combat	no
MT16	Red Demon King (6,6)	after the fight	Spirit Warrior, Nether Captain, Spirit Mage and the King himself become stronger for the rest of the game	yes
MT18	princess (6,5), behind two red doors and the tiles the thief clears	talk	shows the up-staircase (11,11), hidden until then	yes

Floor	Tile	Trigger	Effect	Modelled
MT19	Nether Demon King (6,7)	after the fight	the Nether Demon King becomes stronger (45,000 HP, ATK 2,550, DEF 2,250) for the fight on MT21	yes
MT20	up-staircase (6,8)	handing the Cross to the fairy on MT0	shows the staircase to MT21, hidden until then	yes
MT21	Nether Demon King (6,2)	after the fight	the normal ending of the game	yes
MT21–MT23	Nether Demon King (MT21), fairy	after the fight, talk	further monster upgrades, the three staffs and the hidden ending	no

C The Evaluation Platform

The experiments were run and inspected through the *Magic Tower AI Lab*, a web platform built for this thesis (Section 5.8). This appendix describes its structure, its pages and how it supports reproduction. The screenshots were captured automatically from the running platform.

C.1 Structure

The platform consists of the Python framework, a web server and a browser front end (Figure C.1). The framework contains the engine, the evaluation loops of Chapter 5, the repeated-episode harness and the prompt optimiser. The server, written with FastAPI, exposes the framework through 58 API routes, two routes that serve the front end and one WebSocket endpoint. The front end is a statically exported Next.js application with eight pages: Live, Play, Maps, Experiments, GEPA, Logs, Reference and Projects.

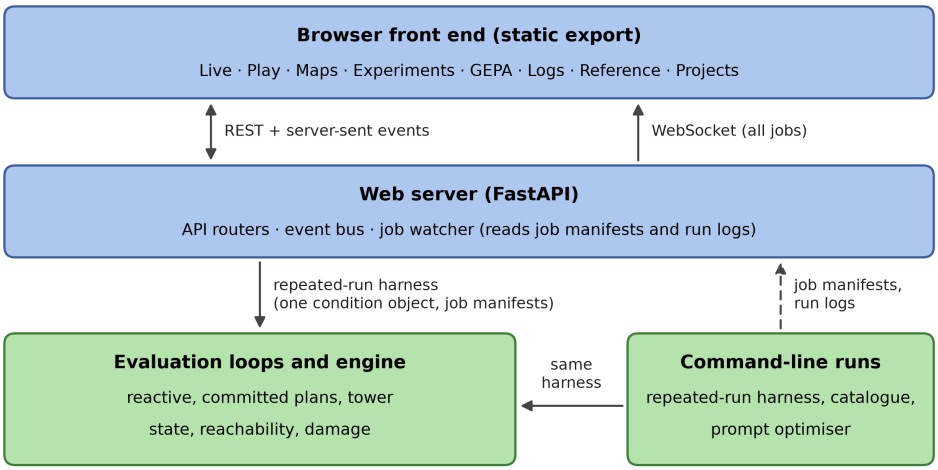


Figure C.1: Structure of the platform. Runs started in the browser and runs started from the command line use the same harness and write the same files; the server's job watcher reads the job manifests and run logs of command-line runs.

Every condition of the command-line harness (floor, model, prompt, planning mode, lookahead depth, engine settings, number of episodes) is also available in the browser through the same condition object.

C.2 Observing and Replaying Runs

Runs started in the browser stream their events (reasoning chunks, turns, floor changes, result) to the page as they happen. Runs started from the command line register a *job manifest* with their label, command line, condition, progress and current run log, and write their run log after every turn. A watcher in the server reads both and republishes every new event on an event bus, to which the browser subscribes over one WebSocket. The Live page (Figure C.2) therefore shows every running experiment, including a tower run of several hours started from a terminal, with its map, hero attributes and reasoning at each turn. Jobs can be paused and resumed between turns and stopped.

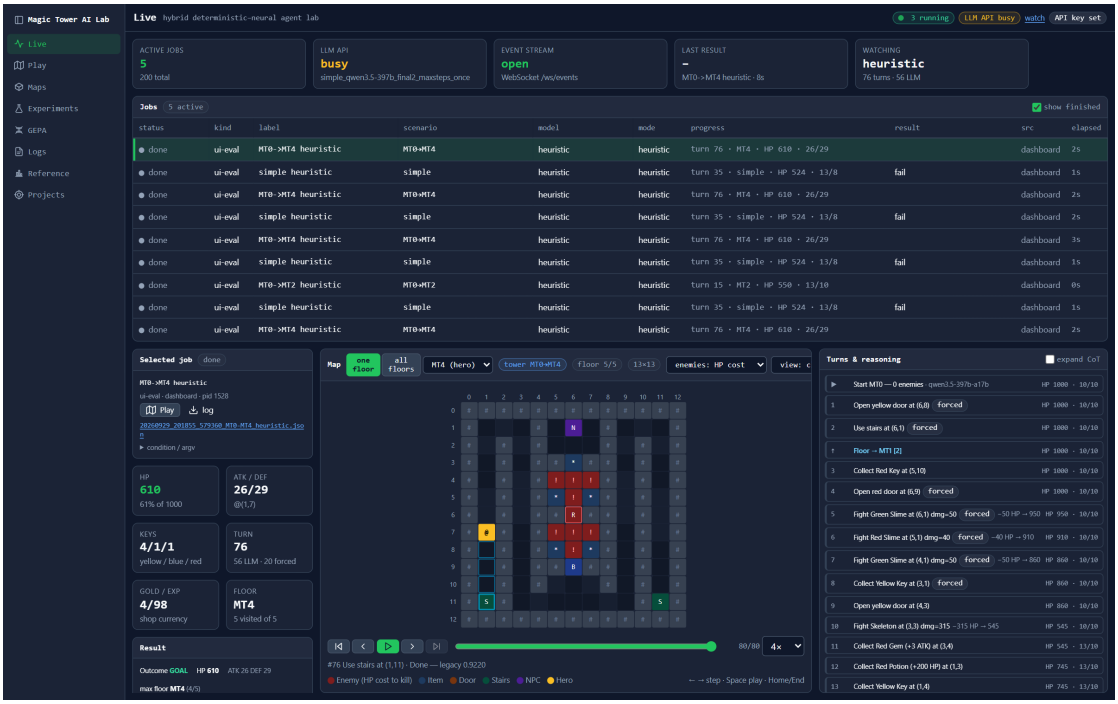


Figure C.2: Live page. The job registry lists every running and finished job with its label, floor, model, progress and result; the selected job's map, attributes and turn cards are mirrored from its run log.

The Play page (Figure C.3) shows one episode. The map is the central element; around it are the hero's attributes, playback controls and one card per turn. Each card shows the text sent to the model, the model's reasoning and answer, the executed action and, where applicable, the engine's override with the model's original choice. In manual mode, a person plays through the same interface as the model: every legal action is ringed and numbered on the map, and clicking a tile takes that action, which makes the claim of Section 5.3 directly checkable. The

exact solver's route can be replayed on the same map. The Logs page (Figure 5.3) lists every stored episode with its outcome, final HP relative to the optimum and model, replays it turn by turn, and compares two runs on the same floor from their first diverging turn. The Maps page shows every single floor and every tower floor with its exact-solver facts and the tower's scripted events, drawn either with coloured cells or with the original game's tiles.

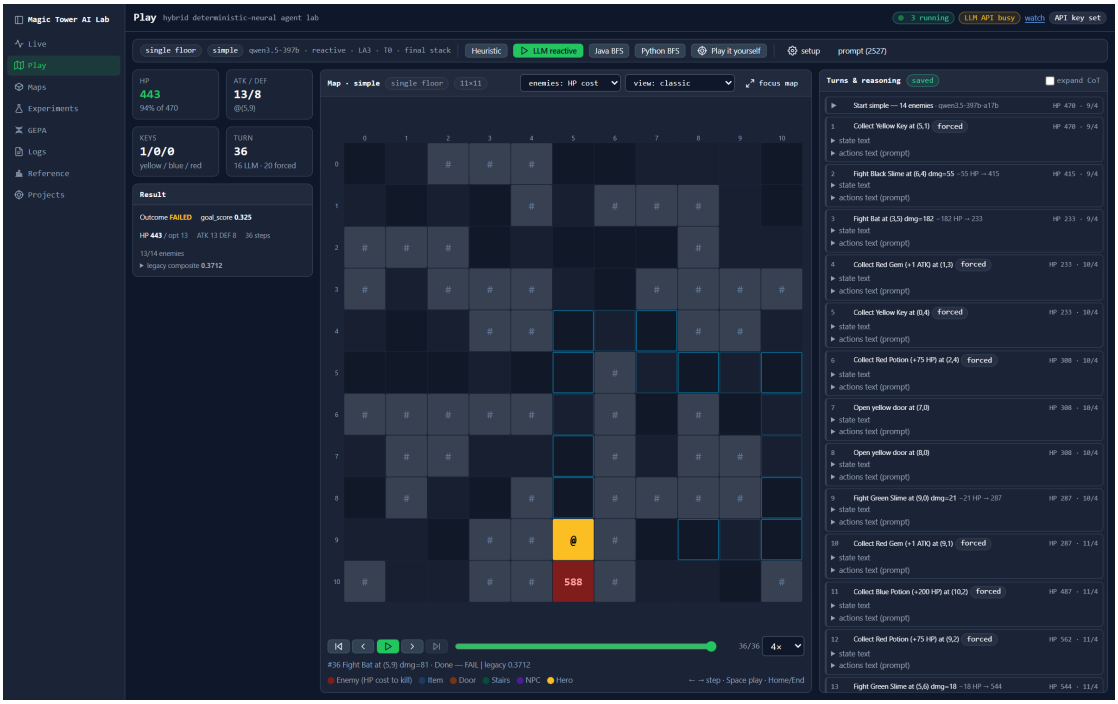


Figure C.3: Play page after a heuristic episode on the simple floor: run settings, attributes and playback on the left, the map in the centre, and the turn cards with the exact observation on the right.

C.3 Experiments and Reproduction

The Experiments page (Figure C.4) lists the experiments of Chapter 6. Its first tab follows Table D.4: for each table, figure and paragraph of Chapter 6 it lists the result files, and for each file the episodes with a link to their replay. A second tab holds catalogue entries with the exact condition of each configuration (floor, model, planning mode, lookahead, prompt, engine settings, N) and the stored result with its Wilson interval. A reader enters an API key for an OpenAI-compatible service; the server holds it in memory only and never writes it to disk or to a log. The reader then selects entries, optionally limits the number of episodes, and starts them. Each configuration runs with the stored arguments under a new label, so

that the thesis results are never overwritten, and the new results appear next to the stored ones. The same catalogue is available from the command line.

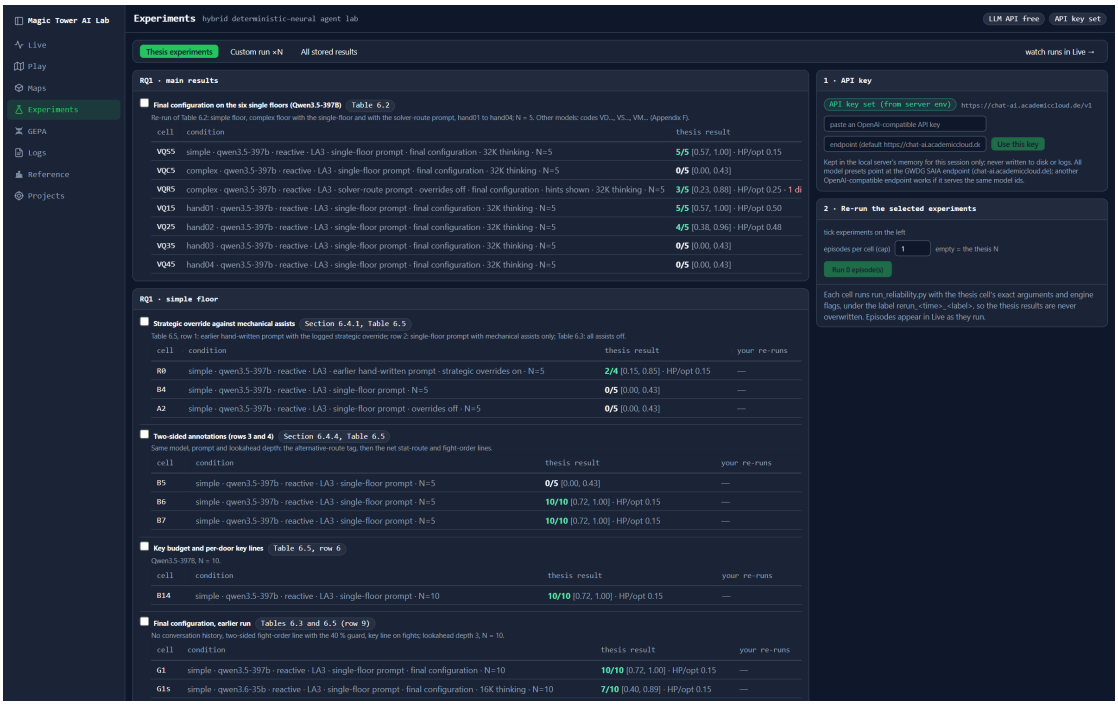


Figure C.4: Experiments page: catalogue entries with each configuration's condition and stored result, the reader's own re-runs, and the panel for the API key and re-runs.

C.4 Prompt Optimisation

The GEPA page (Figure C.5; named after the method of Section 2.6.1) lists every stored run of prompt evolution (Section 5.7) and shows the runs of the (1+1) loop iteration by iteration. For each candidate it shows the score, whether it was kept, the outcome on each training floor with a link to the replay, the prompt as a line-by-line difference from its parent, the teacher's diagnosis and the validator's verdict.

C.5 Logging and Tests

Every episode writes one JSON log with, for each turn, the hero's state, the observation sent to the model, the model's answer and reasoning, the executed action, every override and assisted turn, and the job that produced it. The log is written after every turn, so an interrupted episode leaves a replayable partial log. Because the engine is deterministic, every action sequence in a log can be replayed exactly.

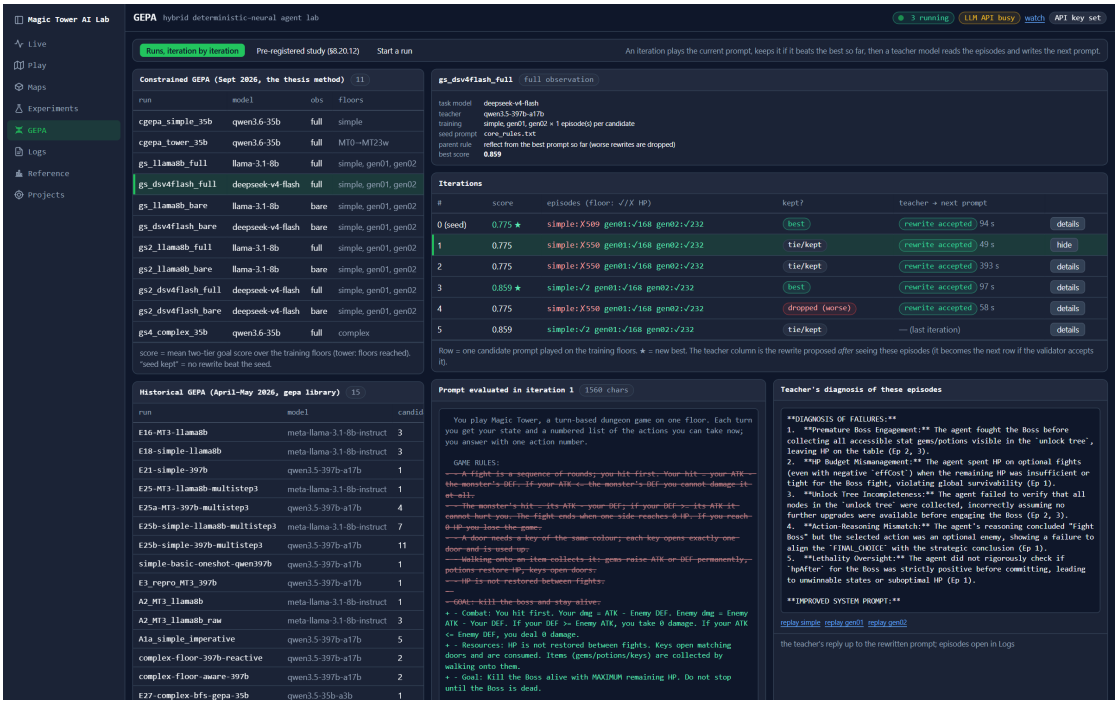


Figure C.5: GEPA page: the runs of the study and one run opened iteration by iteration, with the first rewrite shown as a difference from the seed and the teacher's diagnosis.

Each result file names the run log of every episode, and the run logs are archived as a release asset of the public repository (Appendix D).

Regression tests need no API access. They pin the engine's rules, the route contracts of the server, the four single-floor baselines of the heuristic, the job registry, and the exact text of the observation at a reference decision. A browser test opens every page and checks its layout at several screen sizes.

D Configuration and Result Files

This appendix lists what is needed to reproduce the results of Chapter 6: the environment, the settings behind each named configuration, the repair call, the names of files and settings, the result file behind every reported cell, the sources of computed facts in other chapters, and the runs excluded from Chapter 6. The commands that run each configuration are documented in the framework's repository.

D.1 Environment

The framework is available at `https://github.com/thetobysiu/mt-ai-lab`. It requires Python 3.12; the web front end requires Node.js 20.9 or later, and the exact solver a Java development kit. All commands run from the framework's root directory. The models are accessed through an OpenAI-compatible endpoint, by default the GWDG SAIA service. The API key is read from the environment variable `SAIA_API_KEY` (a template `.env.example` lists every variable), and `SAIA_BASE_URL` selects another endpoint. Table D.1 maps the model names of this thesis to the framework's presets.

Table D.1: Model presets.

Name in this thesis	Preset	Endpoint model	Mode
Qwen3.5-397B	`qwen3.5-397b`	qwen3.5-397b-a17b	thinking
Qwen3.6-35B	`qwen3.6-35b`	qwen3.6-35b-a3b	thinking
Qwen3.8-27B	`qwen3.8-27b`	qwen3.8-27b	thinking
DeepSeek-V4-Flash	`deepseek-v4-flash-think`	deepseek-v4-flash-0731	thinking
DeepSeek-V4-Flash (no thinking)	`deepseek-v4-flash`	deepseek-v4-flash-0731	instruct
Mistral Medium 3.5	`mistral-medium-3.5`	mistral-medium-3.5-128b	instruct (no reasoning channel)
Llama-3.1-8B	`llama-3.1-8b`	meta-llama-3.1-8b-instruct	instruct

D.2 Settings

The engine is configured by environment variables. Table D.2 lists those that distinguish the configurations of Chapter 6.

Table D.2: Settings. Final = value in the final configuration.

Variable	Final	Meaning
`MOTA_EVAL_TEMPERATURE`	0	sampling temperature of the task model
`LOOKAHEAD_DEPTH`	3 (tower 2)	gateways searched behind an action (Section 5.2.2)
`CONVERSATION_HISTORY_TURNS`	0	earlier observations and answers included in each call (default 6)
`MOTA_ORDER_TWO_SIDED`	1	print the fight-order line in both directions, with the guard at the rule-4 threshold (40 %; tower 60 %)
`MOTA_FIGHT_KEY_FLOOD`	1	key line on fights, computed over the whole reachable region
`THINKING_MAX_TOKENS`	16,384 or 32,768 (tower 8,192)	reasoning cap per call (Table 6.1)
`LLM_TIMEOUT`	600 (planning study 1,800)	seconds per call
`MOTA_PLAN_STEPS`	1	commitment k (Section 5.5.2); 60 = long plan
`MOTA_PLAN_NO_REPLAN`	unset	with $k = 60$: single call
`MOTA_PLAN_WORDING`	unset	`floor` = whole-floor wording of the long plan; unset = max-steps wording
`MOTA_BARE_OBS`	unset	1 = bare observation (Section 5.2.7)
`MOTA_DISABLE_OVERRIDES`	unset	1 = mechanical overrides and automatic 0-HP fights off; on single floors, items and keys are still collected and only-lethal positions still end as stuck; on the tower, collection and the climb to unvisited floors are off as well; set for the solver-route and exact-route prompts (Section 5.4)

Variable	Final	Meaning
MOTA_SUPPRESS_HINTS	unset	0 or 1 = show or hide the annotations independently of the previous variable; unset = follows MOTA_DISABLE_OVERRIDES, which then also hides the annotations of fight actions (Appendix E, incident 8)
MOTA_STRATEGIC_OVERRIDES	unset	1 = strategic overrides of the rule-based heuristic (Table 6.5, row 1)
MOTA_OFFER_LEVELUPS	1 (tower)	1 = offer level-ups in the experience shops of the tower (the prompt still forbids them)
MOTA_TOWER_HP_FLOOR	unset (off)	1 = enforce the prompt's 40 % rule on the tower (the first and second tower observations ran with it)
MOTA_HP_RULE_PCT	unset	pins the threshold quoted by the engine lines that refer to rule 4; unset = 40 on single floors, 60 on the tower (the first and second tower observations pin 40)
MOTA_WAY_UP	1 (final tower observation)	way-up line, actions on the route tagged
MOTA_TOWER_FIDELITY	1 (final tower observation)	item values and scripted events of the game data: MT4 thief event, the Cross at the MT0 fairy, hero level, Small Feather (MT6), wizard with the Big Feather (MT13), the MT7 seal that opens after all four White Warriors, ice gates opened by the engine (Section 4.5.2)

Variable	Final	Meaning
MOTA_PLAN_MEMORY	1 (final tower observation)	the answer ends with a one-line plan that is shown at the next decision (Section 5.5.4)
MOTA_NO_KEY_PURCHASE	1 (final tower observation)	yellow and blue keys are not offered for sale; red keys are
MOTA_WAYUP_COMBINED	1 (final tower observation)	way-up line: purchases combined across shops, lethal fights named as such, route passable in order with the gold/EXP of each fight spent before the next
MOTA_FREE_AFFORD	1 (final tower observation)	a fight that costs HP states what the DEF or ATK purchases affordable now do to it (free, or the cheapest cost); a purchase states its price per point and a better-priced open shop; doors and fights state the gold/EXP of the 0-HP fights behind them; red doors state the red-key price; the Cross states how its gain is computed; treasure state line
MOTA_NPC_TRADES	1 (final tower observation)	the MT15 trades (Holy Sword for 500 EXP, Holy Shield for 500 gold) are actions once affordable, never taken by the engine (Section 4.5.3)
MOTA_KEY_RESERVE	1 (final tower observation)	a door off the way up that leaves fewer keys of its colour than the way up still needs states so

Variable	Final	Meaning
`MOTA_AFTER_BATTLE`	1 (final tower observation)	the `setEnemy` calls of a fight's afterBattle script change those monsters for the rest of the run (MT16, MT19; Table B.4)
`MOTA_BUY_QUANTITY`	1 (final tower observation)	a shop purchase may carry `QUANTITY: n` (at most 10); the engine repeats it without a new call while it is affordable (`quantity` event, `repeat` flag on the repeated turns)
`MOTA_VAMPIRE_TAG`	1 (final tower observation)	monsters that take a share of the current HP (special 11) carry the vampire tag instead of the DEF threshold
`MOTA_EVENT_WAY`	1 (final tower observation)	event lines of item-gated characters and seal lines state the way to the item or guard with its doors and missing keys
`MOTA_BLUE_KEY_PURCHASE`	unset	1 = blue keys are for sale although `MOTA_NO_KEY_PURCHASE` hides yellow keys (continuations from MT18, Section 6.7.3)
`MOTA_RESUME_LOG`, `MOTA_RESUME_BEFORE_TURN`	unset	continue a stored tower run: its turns before the given turn are replayed on the current engine (tagged "replayed"), then the model takes over

Variable	Final	Meaning
MOTA_REMOTE_ACTIONS	1 (second and final tower observation)	action list = every reachable object on every visited floor, travel inside the action, stairs only to unvisited floors, tower key budget (Section 5.5.3); unset = first tower observation with return actions
MOTA_SHOP_ROUTE	1 (second and final tower observation)	price the monsters on the way to a shop not yet reached, now and after 1–3 available ATK purchases
MOTA_MULTI_FLOOR_MAX_-TURNS	per run	turn limit of a tower run

D.3 The Repair Call

When an answer contains no valid action number, or a tower answer is truncated (Section 5.3), the engine repeats the request once with reasoning disabled and a budget of 256 output tokens; a truncated single-floor answer is first sent again. The answer must be a JSON object whose single integer field `action_id` is restricted to the listed action numbers, excluding those the mechanical overrides would replace. The schema is enforced by the service's structured-output interface. If the service rejects the schema, the engine falls back to a list of allowed answers and then to an unconstrained short prompt. If all routes fail, the turn counts as a parse error. If no valid action is obtained in three consecutive attempts, the engine takes the rule-based heuristic's action for that turn (Section 5.3). The mechanism can therefore only reduce the number of failed turns; `MOTA_DISABLE_STRUCTURED_-OUTPUT=1` switches it off.

D.4 Names and Labels

Table D.3: Names used in this thesis and the corresponding internal labels.

Name in this thesis	Internal label
effective cost	`effCost`
alternative-route tag	`[ALTERNATIVE ROUTE]`, file label `altroute`
net stat-route line	file label `order_aware` (with the fight-order line)

Name in this thesis	Internal label
fight-order line	`ORDER` line
individual item counts, potion-free effective cost, affordability guard on the order line, required-monster budget (Table 6.5, row 5)	D1 to D4, file label `d1d4`
key budget, per-door key lines (Table 6.5, row 6)	K1, K2, file label `k1k2`
two-sided fight-order line	F-ORD2, `MOTA_ORDER_TWO_SIDED`
key line on fights	F-K3, `MOTA_FIGHT_KEY_FLOOD`
final configuration	final stack, file label `final2`
re-run of Table 6.2	file label `canon`
single-floor prompt	`floor_basic.txt`, basic prompt
tower prompt	`tower_basic.txt` (final tower observation), `tower_basic_v1_stairs.txt` (first tower observation); the prompt of the second tower observation is stored in each run log
rules-only prompt	`core_rules.txt`, core prompt
solver-route prompt	`complex_bfs_strategy_survival.txt`, BFS prompt, survival prompt, route prompt
exact-route prompt	`complex_exact68_sequence.txt`, exact-68 prompt
(1+1) loop	constrained GEPA, cGEPA, `mota/optimizer/constrained_gepa.py`
Pareto loop	`constrained_gepa.run_pareto_loop`, labels `gs5_pareto_*`
teacher	reflection model, reflector
evolved-route prompt	`seed_prompts/gepa_study/evolved_route_-deepseek.txt`
bare observation	`MOTA_BARE_OBS=1`
map-only condition	`--mode map-enum`
item values and scripted events of the game data	`MOTA_TOWER_FIDELITY=1`, fidelity mode
first, second and final tower observation	`final2`, `final3`, `final11` (Table D.5)
rule 4 threshold	`MOTA_HP_RULE_PCT`
40 % override of the first and second tower observations	HP floor, `hp_floor_40`, `MOTA_TOWER_HP_FLOOR`
plan memory	`MOTA_PLAN_MEMORY`, plan note
way-up line	`MOTA_WAY_UP`, “Way up to MTn”
strategic override	strategic rules of `_safety_correct`
mechanical override	mechanical rules of `_safety_correct`

Name in this thesis	Internal label
a configuration of Chapter 6	registry code in `scripts/run_followups.py` (for example `VQS`, `T11QR`), used by `run_catalog.py --codes`

D.5 Result Files

Table D.4 gives the result file behind each reported table, figure and paragraph of Chapter 6. Each file stores the configuration and, for every episode, its outcome and the name of its run log. The run logs are archived as a release asset of the public repository (`run_logs.zip`). `<m>` stands for the model's file tag: `qwen3.5-397b`, `qwen3.6-35b`, `qwen3.8-27b`, `deepseek-v4-flash` or `mistral-medium-3.5`.

Table D.4: Result files (in `results/reliability/` unless stated).

Result	File
Table 6.2, single-floor columns (re-run of Table 6.2, $N = 5$)	`simple_<m>_canon`, `complex_<m>_canon` (single-floor prompt), `complex_route_<m>_canon` (solver-route prompt; Qwen3.6-35B $N = 10$), `hand0<i>_<m>_canon`; table compiled by `analysis/table62_canon.py`
Table 6.2, heuristic and random play	`simple_heuristic_reactive`, `complex_heuristic_reactive`; hand-designed floors: `scripts/floor_lab.py analyse <floor>` (heuristic and 500 random games)
Table 6.2, tower column	the final-observation rows of Table 6.13
Temperatures 0.6 and 1.0 (Section 6.3, Table 6.3)	the file of the temperature-0 cell with the suffix `_t06` or `_t10`: `simple_<m>_canon_t06`, `complex_<m>_canon_t06`, `complex_route_<m>_canon_t06` and the same with `_t10` (Qwen3.5-397B, DeepSeek-V4-Flash)
Table 6.3, bare observation	`results/gepa_study/gs_397b_bare.json`, `gs_35b_bare.json`
Tables 6.3 and 6.4, map only	`simple_qwen3.5-397b_mapenum`, `simple_qwen3.6-35b_mapenum`; Table 6.4 counted from the visible answers by `analysis/mapenum_rescore.py` (the `map_enum` fields of the result files also count options named in the reasoning)

Result	File
Section 6.2 and Table 6.3 (earlier episodes of the final configuration, depth 3), Table 6.5 (row 9)	`simple_<m>_final2_la3` (Qwen3.5-397B, Qwen3.6-35B)
Section 6.3, hand01 and hand02 at depth 3	`hand0<i>_deepseek-v4-flash_final2` (32,768 tokens: `…_reactive_t32`)
Section 6.1.2, tower settings on single floors	`<floor>_<m>_canon_hp60` (prompt `floor_basic_hp60.txt`, `MOTA_HP_RULE_PCT=60`; the solver-route cells keep their prompt) and `…_canon_planmem` (`MOTA_PLAN_MEMORY=1`), Qwen3.5-397B and DeepSeek-V4-Flash
Section 6.3, hand01 and hand02 at depth 1 and with the bare observation	`hand0<i>_deepseek-v4-flash_final2_la1`, `…_bare`
Table 6.3, lookahead depth	`simple_<m>_final2_la<d>`
Table 6.3, conversation history	`simple_qwen3.5-397b_basic_k1k2`, `simple_qwen3.6-35b_basic_k1k2_16k`; without history `…_hist0`
Table 6.5 and Figure 6.1, rows 1 to 9	`simple_<m>_` followed by `legacy_strategic_control`, `basic_mechanical_override`, `basic_mech_altroute`, `basic_order_aware`, `basic_d1d4`, `basic_k1k2` (Qwen3.6-35B: `basic_k1k2_16k`), `basic_k1k2_hist0` (Qwen3.6-35B: `basic_k1k2_16k_hist0`), `final_la3`, `final2_la3`
Section 6.4.1, override census	`analysis/override_census.py` over the stored run logs (94 replaced turns in 68 episodes)
Section 6.4.5, Table B.3 and Figure 6.2	`complex_value_trace_[a-e].json`, written by `analysis/complex_value_trace.py` and summarised by `analysis/complex_value_summary.py`
Table 6.6 and Section 6.5.2 (second configuration, reference episodes on the simple floor)	`results/gepa_study/gs_*` (first configuration), `gs2_*` (second configuration); loops in `results/gepa/gepa_result_gs_*.json` and `…_gs2_*.json`; reference runs on the simple floor `gs_397b_*`, `gs_35b_*`

Result	File
Table 6.7	`analysis/gepa_rule_themes.py` over the candidates in `results/gepa/gepa_result_gs*_*.json`
Section 6.5.2 and Figure 6.3, paired confirmation	`results/gepa_study/gs3_dsbare_confirm.json`
Section 6.5.2, complex floor	`results/gepa_study/gs4_complex_35b.json` (loop: `results/gepa/gepa_result_gs4_complex_35b.json`)
Table 6.8, second study	`results/gepa_study/gs5_pareto_dsthink_bare.json`, `gs5_oneplusone_dsthink_bare.json` (loops: `results/gepa/gepa_result_gs5_*.json`)
Section 6.5.4, evolved-route prompt	loop `results/gepa/gepa_result_evolved_route_complex_-deepseek.json`; evaluation `complex_evolved_route_deepseek-v4-flash`
Tables 6.9 and 6.10 and Figure 6.4, single floors	`simple_<m>_final2_plan<k>`, `…_planf` (long plan, whole-floor wording, re-planned), `…_oneshot_plan` (single call, whole-floor wording), `…_maxsteps`, `…_maxsteps_once`; reactive arms `simple_qwen3.5-397b_final2_la3`, `simple_deepseek-v4-flash_final2_reactive` and `…_reactive_t32`, `complex_deepseek-v4-flash_survival_final2_reactive` and `…_reactive_t32`, `hand0<i>_deepseek-v4-flash_final2` and `…_reactive_t32`, `simple_qwen3.8-27b_final2_reactive_t32_t06`, `complex_qwen3.8-27b_survival_final2_reactive_t32_t06`; committed modes `complex_deepseek-v4-flash_survival_final2_*`; `hand0<i>_deepseek-v4-flash_final2_plan<k>`, `…_maxsteps`; Qwen3.8-27B: `simple_qwen3.8-27b_final2_*_t06`, `complex_qwen3.8-27b_survival_final2_*_t06`
Table 6.9, tower row (second tower observation)	`tower_deepseek-v4-flash_final3_reactive`, `…_plan<k>`, `…_maxsteps`, `…_maxsteps_once`
Section 6.6, exact route	`complex_<m>_exact68_final2`
Table 6.11, hand-designed floors	`hand0<i>_<m>_canon`
Appendix G.3, earlier episodes on hand01	`hand01_qwen3.5-397b_final2`, `hand01_deepseek-v4-flash_final2`

Result	File
Table 6.12, generated floors	`gen0<i>_qwen3.5-397b_final2` (gen01 to gen03), `gen<i>_deepseek-v4-flash_final2` (gen03 to gen15); pooled counts from all files that name the floor, except `gs3_dsbare_confirm`
Table 6.13, tower	`tower_<m>_<label>_reactive` with the labels of Table D.5 (Qwen3.5-397B, first observation: `tower_qwen3.5-397b_final2`); continuations from MT18 `tower_<m>_final11_resume_blue`; heuristic `tower_MT0-MT23w_heuristic_final3`, with the game's item values `tower_MT0-MT23w_heuristic_final7`
Figures 6.1 to 6.4	plotted by `scripts/thesis_figures.py` from the files of Table 6.5 (Figure 6.1), the value traces (6.2), `gs3_dsbare_confirm.json` (6.3) and the planning files (6.4)

Table D.5: Tower observations (Section 5.5.3, Table 6.13) and their result-file labels. Flags = settings added to the row above, relative to the defaults of the code at the time of the runs (the defaults have since changed, so that Table D.2 lists some of these settings as unset). Every tower run also uses the settings of the final configuration and lookahead depth 2; reactive runs use a reasoning cap of 8,192 tokens. The first tower observation ran under the single-floor final configuration of its time, whose result files share the label `final2`.

Observation	Label	Flags added	Tower prompt (characters)
first	`final2`	none; the 40 % override was on by default	`tower_basic_v1_-stairs.txt` (3,629)
second	`final3`	`MOTA_REMOTE_ACTIONS=1`, `MOTA_SHOP_ROUTE=1`	stored in each run log (3,725)
final	`final11`	`MOTA_WAY_UP=1`, `MOTA_TOWER_FIDELITY=1`, `MOTA_TOWER_HP_FLOOR=0`, `MOTA_OFFER_LEVELUPS=1`, `MOTA_PLAN_MEMORY=1`, `MOTA_NO_KEY_PURCHASE=1`, `MOTA_WAYUP_COMBINED=1`, `MOTA_FREE_AFFORD=1`, `MOTA_NPC_TRADES=1`, `MOTA_KEY_RESERVE=1`, `MOTA_AFTER_BATTLE=1`, `MOTA_BUY_QUANTITY=1`, `MOTA_VAMPIRE_TAG=1`, `MOTA_EVENT_WAY=1`; rule 4 at 60 %	`tower_basic.txt` (6,035)
final, resumed on MT18, blue keys for sale	`final11_resume_-blue`	`MOTA_RESUME_LOG=20261004_052939_498308_MT0-MT23w_llm.json`, `MOTA_RESUME_BEFORE_TURN=708`, `MOTA_BLUE_KEY_PURCHASE=1`, `MOTA_MULTI_FLOOR_MAX_TURNS=860`	`tower_basic.txt` (6,035)

The continuations from MT18 resume the run log `20261004_052939_498308_MT0-MT23w_llm.json`, a run of Qwen3.5-397B with a predecessor of the final observation (label `final10`). That run was played before the hidden staircases of MT18 and MT20 were modelled; its result file records MT21, and MT18 is its highest floor

on the current engine (Appendix E, incident 17).

For tower runs, the field `llm_calls` of a result file counts every turn not taken by the engine, including executed plan steps (Appendix E, incident 15). The number of model calls reported in Table 6.9 is the number of `thinking` events in the run log (`analysis/tower_calls.py`). Later result files also store it as `model_calls`.

Table D.6 gives the sources of facts that were computed or checked rather than read from a result file.

Table D.6: Sources of computed facts.

Fact	Source
Table 4.1, special abilities per floor	`analysis/special_abilities_census.py` over the floor files and `data/games/24/project/enemys.js`
Table B.4, scripted events	`scripts/inventory_floor_events.py` over `data/games/24/project/floors/*.js`
Sections 4.5.1 and 4.6.3, Tables 4.3 to 4.5, Figure 4.3, floor facts	`scripts/floor_lab.py analyse <floor> --decisions`; generated floors: `data/floors/gen/manifest.json`
Figures 2.4, 4.1, 4.2, B.1 and B.2	`scripts/thesis_floor_figure.py` (game sprites), `scripts/solver_graph_figure.py` (node graph before merging)
Table B.1, solver timings	published solver `magic-tower-AI/AI/src`; exact solver `tools/java_bfs` (`--summary`)
Listing 5.1, Listing A.7 and Listing G.4	`analysis/render_first_prompt.py` (final configuration, first decision; simple floor by default, after the engine has collected the key at (5,1); hand01 for Listing G.4)
Character counts of Appendices A and F	length of the prompt file without leading and trailing white space (`analysis/appI_prompt_chars.py`); prompts not stored as files: the `prompt` field of the run log (`analysis/appI_runlog_prompt.py`)
Appendix G, reasoning excerpts	`analysis/extract_cot.py <label> <run> <turn>`
Verification of the references (Chapter 3)	`thesis/reference/gemini_review_verification_2026-09-21.md`, `perplexity_review_verification_2026-09-21.md`

Fact	Source
Section 6.5, noise floor, G4, trial floors and paired counts	`analysis/ch6b1_gs.py`, `analysis/ch6b1_pairs.py` over `results/gepa_study/gs*_*.json`
Section 6.6 and Table 6.9, tower planning runs	`analysis/ch6b1_plan.py`, `analysis/ch6b1_tower.py`; tower calls = `thinking` events in the run log (`analysis/tower_calls.py`)
Section 6.7.1, first door on hand01 and mechanical overrides	`analysis/ch6b_hand_canon.py`, `analysis/ch6b2_hand_ov.py`, `analysis/ch6b2_hand01_cot.py` over the `hand0<i>_<m>_canon` run logs
Sections 6.2 to 6.4, simple floor: first decision, winning trajectory and departures	`analysis/simple_turns.py <result files>` (the model's turns per episode); state values from `scripts/floor_lab.py analyse data/floors/simple.txt --decisions`
Table 4.4 and Section 6.4, costs along the optimal route and the winning trajectory, state values at turns 18 and 22	`analysis/simple_route_costs.py`; `analysis/simple_turn_values.py <result file> <run>` (state value after every legal action at each decision of an episode)
Section 6.7.3, the stored position on MT18	`analysis/ch6b2_tower_more.py`, `analysis/ch6b2_blue.py`; run log `20261004_052939_498308`

D.6 Results Excluded from Chapter 6

Results whose harness or setup was defective are kept under a file-name prefix that names the defect and are not pooled with valid results (Table D.7). Most of them come from the planning study (Section 6.6). After a first trial in which the plan

line was not yet parsed, the plan harness was corrected four times before valid data were obtained: a plan line was discarded when reasoning preceded it, the parser of an earlier single-plan mode matched almost no plan, models repeated the chosen action as the first planned step, and a planned step that had become unwinnable was handed to the heuristic instead of ending the plan. In addition, the reasoning cap was raised, because short caps truncated plan answers. The earlier episodes of Qwen3.8-27B at temperature 0 are not reported, because their committed plans exist only in the defective `_v1next_` version; Table 6.9 reports this model from later episodes at temperature 0.6.

Table D.7: Excluded results (prefixes in `results/reliability/`).

Prefix	Defect	Effect	Episodes
`_void_nexttrunc_`	The answer extractor kept only the `FINAL_CHOICE` line when reasoning preceded it, so the `NEXT:` plan line was lost.	DeepSeek-V4-Flash executed no commitments (its planning cells were reactive in effect); Qwen3.5-397B about half.	32 (simple, tower)
`_void_oneshotparser_`	An earlier single-plan mode expected coordinates in a `PLAN:` line; Qwen3.5-397B wrote none, and the other plans listed items and unreachable tiles.	In 9 of 10 episodes, the heuristic chose every decision.	10 (simple)

Prefix	Defect	Effect	Episodes
`_v1next_`	The instruction "the next $k-1$ actions after it" was read as including the chosen action; a committed step that had become unwinnable was handed to the heuristic instead of ending the plan. In two cells, the daily request limit of the API key was also exhausted (Appendix E, incident 14).	DeepSeek-V4-Flash repeated the chosen action as the first planned step in every plan; three to eight heuristic turns per cell.	44 (simple, complex, tower)
`_void_preoverridefix_`	Same defect as `_v1next_`; one episode started before the fix.	—	1 (simple)
`_void_…_noparse`	First planning trial, before the plan line was parsed.	—	2 (simple)
`_ablation_effcost_-numeric`	A placeholder in the effective-cost annotation was replaced by a number (Appendix E, incident 1).	Kept as an unintended ablation: 0 of 20.	20 (simple)
`_stale_shop_prompt`	A surviving process ran an obsolete configuration alongside the corrected one (Appendix E, incident 4).	—	3 (simple)

Prefix	Defect	Effect	Episodes
`_partial_`	Cell stopped at the user's request after one episode.	—	1 (complex)

E Validity Incidents

Section 7.3 draws six methodological lessons from the threats to validity that the study encountered. Table E.1 gives the dated record of all eighteen incidents: how each was detected, what it changed and how it was corrected. Results in Sections 6.2 to 6.7 were obtained after the corresponding correction. There is one exception: the Qwen3.6-35B cell in row 5 of Table 6.5 (incident 9). Results affected by incidents 12 and 17 were re-expressed rather than re-run.

Table E.1: Validity incidents in the order in which they were found.

#	Found	Incident	Detection	Effect on results	Correction
1	2026-09-20	**Annotation placeholder.** The effective cost of gateway fights without lookahead value was printed as `None`; an audit replaced it with the numeric damage, assuming a formatting error.	The first 20 repeated episodes (Qwen3.5-397B and Qwen3.6-35B) won none, against 6 of 14 stored episodes; the stored reasoning read the placeholder as "not recommended yet".	0 of 20; after reverting, 5 of 10 per model. The 20 episodes are kept as an unintended ablation.	Placeholder restored; a regression test pins the observation text at a reference decision.
2	2026-09-20	**Two evaluation loops.** The command-line loop and the platform loop applied different overrides under the same name "reactive"; all stored wins came from the platform loop.	Code review during incident 1.	The first comparison of new and stored episodes was confounded.	The harness uses the platform loop; every result records its loop.
3	2026-09-20	**Wrong default prompt.** A validation script mapped the simple floor to the prompt written for floors with shops.	Provenance check of the stored wins.	A hidden factor between runs.	The engine's default prompt per floor; every result records its prompt file.
4	2026-09-21	**Stale process.** A child process of an aborted batch continued an obsolete configuration next to the corrected one.	The result file recorded the old prompt; the time per episode halved when the process ended.	Three invalid episodes and about 2.5 hours of contention for the API.	Process stopped, results quarantined, cell re-run.
5	2026-09-21	**Scorer gating.** The penalty for a surviving boss applied only to floors with a stored reference.	Unit test written for the success criterion.	None on stored results; a new boss floor would have scored a surviving boss as a clean run.	Penalty keyed on the floor's goal.

#	Found	Incident	Detection	Effect on results	Correction
6	2026-09-21	**Models removed from the service.** Three models of the early experiments (Qwen3.5-35B, Qwen3.5-122B, Qwen3.5-27B) were removed from the service.	Availability check against the service's model list.	Early model comparisons and the first complex-floor wins cannot be reproduced; the successor Qwen3.6-35B won 0 of 10 on the prompt with which its predecessor had won 2 of 18.	Only served models are used; the old presets are kept so that historical labels resolve.
7	2026-09-24	**Silent strategic overrides.** The strategic override replaced the model's choice whenever the heuristic judged another action much cheaper, without logging it.	Comparison of the model's answer with the executed action in every stored log.	94 replaced turns in 68 episodes; all 21 stored simple-floor wins contained a replacement (Section 6.4.1).	Overrides reduced to mechanical cases and logged per turn; strategic rules only on explicit request.
8	2026-09-25	**Annotations hidden with the overrides.** Switching the overrides off also removed the annotations of fight actions, while door actions kept theirs.	A drop from 4 of 5 to 1 of 5 on the complex floor after a change to the door annotations (results not reported in Chapter 6).	Earlier results with overrides off were obtained with less information than reported.	Separate settings for overrides and annotations.
9	2026-09-26	**Repair chose an unwinnable fight.** With a reasoning cap of 8,192 tokens, truncated answers were repaired by constrained decoding, which chose the unwinnable boss on 12 turns of one episode.	Log inspection of a 1-of-5 result.	That cell measured the heuristic fallback rather than the model.	The repair call only offers actions the mechanical overrides would not replace; cap raised to 16,384.
10	2026-09-26	**Crash on door-only action lists.** A variable of the annotation code was undefined when no fight was listed.	Two tower runs ended with an error.	Both runs void and re-run.	Code fixed and covered by a test.
11	2026-09-26	**Concurrency.** Four experiments in parallel on one API key slowed Qwen3.5-397B from about 20 s to about 320 s per call.	Latency census.	No invalid result from the slowdown itself (the invalid cell of incident 4 came from a stale process), but runs of a day.	At most two concurrent experiments per key.

#	Found	Incident	Detection	Effect on results	Correction
12	2026-09-29	**Reference solver.** The published solver keeps, for each set of cleared obstacles, the first state it expands instead of the best (Section 4.6.2).	An exact re-solve during the construction of the hand-designed floors.	The complex floor's optimum is 68, not 22; every claim of an optimal complex-floor win was an artefact.	Exact solver; all stored results re-expressed against 68.
13	2026-09-29	**Planning harness.** Four defects of the committed-plan loop (Appendix D.6).	Reading the logs of finished cells.	Several planning cells measured reactive or heuristic play.	Each defect fixed; affected results excluded.
14	2026-09-29	**API quota.** The daily request limit of the second API key was exhausted during a batch.	Empty answers and fallback turns in two planning cells.	Both cells (DeepSeek-V4-Flash, complex floor, commit-3 and commit-5) belong to the excluded results of Table D.7 (Appendix D.6).	Batches moved to the first key.
15	2026-10-01	**Tower call count.** The field `llm_calls` of tower result files counted every turn not taken by the engine, including executed plan steps.	Recount of the model calls from the reasoning events of the run logs.	The call counts first reported for the tower planning comparison were counts of decided turns.	Calls counted from the run logs; result files store `model_calls` from then on (Appendix D).
16	2026-10-03	**Purchase enumeration.** The in-order way-up line priced every combination of up to ten purchases of each of eight shop offers for each fight on the route.	A tower run of Qwen3.5-397B with a predecessor of the final observation slowed to about 80 minutes per turn from turn 560, on MT15 with 1,940 gold and 3,034 EXP unspent.	The run was stopped after reaching MT16 and has no result file.	Exact table of the gain per currency; the in-order line enumerates at most 20,000 combinations and otherwise prices ATK-only and DEF-only purchases.
17	2026-10-04	**Hidden staircases.** The up-staircases of MT18 and MT20, which the game hides until the hero talks to the princess and hands in the Cross, were shown from the start.	Tile-by-tile check of the floor files of the game data against the engine.	A tower run of Qwen3.5-397B with a predecessor of the final observation reached "MT21" instead of MT18; its highest floor was overstated by three floors.	Staircases hidden until their scripted events (Table B.4); the run's arrival on MT18 is the stored position continued in Section 6.7.3.
18	2026-10-05	**Misattributed losing decision.** The failures on the complex floor with the single-floor prompt were attributed to the yellow door (10,5), the conspicuous error that the solver-route prompt calls a trap.	Exact state value of every state of every stored complex-floor episode, computed with the exact solver.	No success rate changed; all 30 failures had been lost at an earlier fight, 29 of them at the Red Slime (11,8), and the door came after the loss.	Section 6.4.5 reports the decision after which the boss becomes unwinnable (Table B.3, Figure 6.2).

Incidents 1 to 6 were found during the first campaign of repeated episodes.

F Prompt Lineage

The input of the model has three layers, which are kept apart throughout this thesis. The *system prompt* is the strategy text; it is the only layer that the prompt optimiser changes and the only one called a prompt version. The *static floor map*, generated by the engine, is appended once to the system prompt at the start of an episode. The *per-turn observation*, also generated by the engine, is the user message of each call (Section 5.2). The second and third layers are identical for every prompt version; a change to them is a change of the observation, not of the prompt.

Table F.1 lists the system prompts of the conditions reported in Chapter 6 with their role; the earlier hand-written prompt of Table 6.5, row 1, is stored in the run logs of that row. Prompts with a listing number are reproduced in Appendix A. The other evolved prompts are stored in the repository and in the result files of Appendix D.

Table F.1: System prompts. Characters as sent to the model.

Prompt	Characters	Role	Used in
Single-floor prompt	2,526	Combat rules, how to read the observation, five prompt rules	Chapter 6, every single-floor result of the final configuration (Listing A.1)
Single-floor prompt at 60 %	2,526	The single-floor prompt with its HP limit (rule 3 of that prompt) at 60 % instead of 40 %; no other change	Section 6.1.2, tower settings on single floors
Tower prompt	6,035	As above, with scouting, shop rules and cross-floor reading	Chapter 6, final tower observation (Listing A.2)
Tower prompts of the first and second tower observation	3,629 and 3,725	Earlier versions of the tower prompt (Appendix A.2); the second is stored in each run log	Tables 6.9 and 6.13

Prompt	Characters	Role	Used in
Rules-only prompt	826	Game rules, goal and answer format only	Prompt optimisation study, seed and reference (Listing A.3)
Evolved prompt, paired test	3,152	Rewrite by the optimiser for DeepSeek-V4-Flash (no thinking), bare observation	Section 6.5.2 (Listing A.4)
Other evolved prompts of the first study	826 to 2,423	Best prompt of each other run; five runs returned the seed unchanged	Table 6.6
Evolved complex-floor prompt	3,238	Best prompt of the run that started from the single-floor prompt on the complex floor (Qwen3.6-35B)	Section 6.5.2
Best prompts of the second study	3,292 ((1+1) loop), 2,958 (Pareto loop)	Best of eight candidates each, DeepSeek-V4-Flash with thinking, bare observation	Table 6.8
Evolved-route prompt	2,682	Best prompt of the (1+1) loop on the complex floor with the validator off; it names no position	Section 6.5.4
Solver-route prompt (with survival rule)	5,473	Published solver's route as an allowlist, door order and phases, rule never to take a lethal fight	Compiled-knowledge condition (Listing A.5)
Exact-route prompt	2,101	The 21 decisions of the exact optimal route with expected HP	Section 6.6 (Listing A.6)

G Reasoning Excerpts

This appendix reproduces the models' reasoning at the decisions discussed in Chapter 6, verbatim from the run logs, together with the observation at the first decision on hand01. Each reasoning listing names its run log, which the platform replays with the full observation (Appendix C); the run logs are archived as a release asset of the public repository (Appendix D). Omissions are marked "[…]"; nothing else is changed.

G.1 The Decisive Turn of the Simple Floor

Both episodes stand at turn 9 with 336 HP, ATK 9 and DEF 5 (Listing 6.1). The first ran on the configuration of Table 6.5, row 2, and failed; the second on row 4, and won.

Listing G.1: Qwen3.5-397B at turn 9, row 2 (failure, 3,896 characters; run log `20260924_092937_769885_simple_llm.json`).

```
The user wants me to choose the best action in this Magic Tower game.

**Current State:**
- HP: 336
- ATK: 9
- DEF: 5
- Keys: Y=2, B=0, R=0
- Position: (1,10)
- Goal: Defeat Boss at (5,10) with max HP.
- Boss Requirement: Need ATK ≥ 13 to deal damage (Boss DEF=12). Current ATK=9.
↪  Need +4 ATK.
- Current Gems collected: 1 DEF gem (from T7).
- Remaining Gems on floor: 4 ATK gems, 3 DEF gems.
- Total potential stats if all collected: ATK=13, DEF=8.

**Analysis of Options:**

1.  **Fight Slime Lord at (1,9)**
    - Cost: 95 HP (hpAfter=241).
    - Lookahead: 290 (Blue Potion +200 HP, Blue Gem +1 DEF).
    - EffCost: -195 (Huge gain).
    - Unlocks: +1 DEF, +200 HP.
    - This is right next to me at (1,10). Slime Lord is at (1,9).
    - Getting +1 DEF here brings me to DEF=6.
    - Getting +200 HP brings me to ~536 HP.
    - This seems very efficient.

2.  **Fight Bat at (3,5)**
    - Cost: 168 HP (hpAfter=168). Risky.
```

```
    - Lookahead: 205 (Red Potion +75, Red Gem +1 ATK, Yellow Key).
    - EffCost: -37.
    - Unlocks: +1 ATK.
    - This is far away. I am at (1,10). To get there I need to traverse.
    - High HP cost now.

3.  **Fight Slime Lord at (4,2)**
    - Cost: 95 HP.
    - Lookahead: 0.
    - Unlocks path to another Slime Lord -> Red Gem +1 ATK.
    - Far away.

4.  **Fight Red Slime at (5,3)**
    - Optional. Skip.

5.  **Fight Bat at (5,9)**
    - Cost: 168 HP.
    - Leads to Boss (cannot win yet).
    - High cost, no immediate stat gain listed in lookahead compared to option
    ↪  1.

6.  **Open Yellow Door at (7,0)**
    - Dead end / far away. I am at (1,10).

7.  **Open Yellow Door at (7,5)**
    - Far away.

**Comparison:**
I am currently at (1,10).
Action 1 (Slime Lord at 1,9) is adjacent.
It costs 95 HP but gives 200 HP potion + 1 DEF gem.
Net HP change: -95 + 200 = +105 HP.
Stat gain: +1 DEF.
This is a massive positive value move (effCost = -195).
It also clears the path potentially towards other items.
I need +4 ATK to beat the boss. I need to collect ATK gems.
Option 1 gives DEF and HP.
Option 2 gives ATK but costs 168 HP net (after potion).
Option 1 is safer and builds HP buffer.
Also, I am standing right next to (1,9).
The Slime Lord at (1,9) unlocks Blue Potion (+200 HP) and Blue Gem (+1 DEF).
With 336 HP, spending 95 is safe (leaves 241), then gaining 200 puts me at 441.
This increases my HP budget significantly.
The DEF increase helps reduce future fight costs.
The ATK gems are crucial, but I should grab the "free" HP/DEF first if they are
↪  low cost/negative effCost.
Action 1 has effCost = -195. This is the best value on the board.
Action 2 has effCost = -37.
```

```
Action 7 has effCost = -36 (net_lookahead=237, cost=162+55+48+95? No, the text
↪  says net_lookahead=0 for door, but inside tree it shows gains. Wait, Action
↪  7 says net_lookahead=0(raw=162-gw_cost=198). That implies positive cost.
Action 1 is clearly the winner. It is adjacent, negative effective cost (gain),
↪  and provides resources.

[...]

Final decision: Action 1.
```

Listing G.2: Qwen3.5-397B at turn 9, row 4 (win, 1,329 characters, complete; run log `20260924_170639_455104_simple_llm.json`).

```
Looking at the current state:
- HP=336, ATK=9, DEF=5, Y=2 keys
- Need ATK≥13 to damage Boss (currently ATK=9, need +4)
- Several stat gems still available

Key observations from the action list:

1. Action 6 (Open Yellow Door at 7,0) has net_lookahead=237 with effCost showing
↪  it unlocks a path to +1 ATK gem with only 16 HP fighting cost. This gives
↪  +78 HP net benefit - this is excellent value!

2. Action 1 (Fight Slime Lord at 1,9) has lookahead=290 but costs 95 HP now. The
↪  note says it wastes 12 HP if done before getting the ATK from action 6.

3. Action 2 (Fight Bat at 3,5) costs 168 HP and unlocks +1 ATK, but the note
↪  says it wastes 22 HP if done before getting ATK from action 6.

4. Action 7 (Open Yellow Door at 7,5) has net_lookahead=0 and would cost 198 HP
↪  to get +1 ATK +1 DEF - not as efficient as action 6.

The clear winner is Action 6 - opening the yellow door at (7,0). It:
- Costs 0 HP to open (just uses a key)
- Unlocks a path with only 16 HP fighting cost
- Gets +1 ATK gem
- Saves 94 HP on future fights
- Net +78 HP benefit

This is the most efficient action - it gets us closer to the ATK=13 needed for
↪  the boss with minimal HP cost, and the lookahead shows it's highly valuable.

After this, we should collect the ATK gem, then reassess which fights to take
↪  with the improved stats.

FINAL_CHOICE: 6
```

G.2 A Positive Order Line Read as an Instruction

Qwen3.6-35B at turn 4 of the simple floor with 391 HP, on the configuration of Table 6.5, row 8 (Listing 6.2). The chosen Bat costs 182 HP, 46.5 % of the hero's HP.

Listing G.3: Qwen3.6-35B at turn 4, row 8 (failure, 6,459 characters, end of the reasoning; run log `20260927_023900_034601_simple_llm.json`).

```
[...]
The explicit ordering advice in the action description is a huge hint: "take
↪  this fight BEFORE 'Open yellow door at (4,7)'".

What about Action 2? Slime Lord at (4,2). Damage 114. Unlocks another Slime Lord
↪  giving +1 ATK. But effCost is None, and it says "wastes 8 HP". So skip.

Action 4 is optional. Skip.
Action 5 leads to boss but can't win. Skip.
Action 6 dead end. Skip.
Action 7 uses last key, bad. Skip.

Therefore, Action 1 is clearly the intended move. It aligns with "Stat items
↪  first", gives necessary ATK, has positive effCost (52), and the description
↪  explicitly advises doing it before the door.
[...]
Wait, let's look at Action 3 again. "net_lookahead=23". Opening a door costs 0
↪  HP. It just uses a key. But to get the items, you fight Black Slime (55
↪  dmg). So effectively you spend 55 HP to get +2 DEF and +2 keys. That's very
↪  cheap! But does it help with ATK? No.
Action 1 gives +1 ATK for 182 dmg. But it also gives a key and potion.
The prompt says for Action 1: "ORDER ... take this fight BEFORE 'Open yellow
↪  door at (4,7)'". This is a direct instruction from the engine's lookahead
↪  analysis. I should follow it.
Also, priority 1: "Stat items first."
Action 1 is definitely correct.
[...]
```

G.3 The First Decision on hand01

Listing G.4 shows the observation at this decision, with the explanatory note and repeated gateway annotations shortened; the map of hand01 is given in Appendix B.5.3. The reasoning excerpts come from episodes of this configuration on an earlier version of the code. In it, DeepSeek-V4-Flash opened the left door (4,8) in all five episodes and Qwen3.5-397B the right door (6,8) in all five; in the re-run of Table 6.2, 18 of 20 episodes opened the right door first (Section 6.7.1).

Listing G.4: The observation at the first decision on hand01 (final configuration). "[…]" marks repeated gateway annotations and the fixed explanatory note.

```
=== Floor: hand01 | Turn 0/300 ===
Hero: HP=400 (100.0% [OK]) ATK=9 DEF=4 MDEF=0 Gold=0 EXP=0
Keys: Y=1 B=0 R=0
Key budget: yellow 1 in hand + 3 on this floor = 4 for 4 closed yellow door(s);
↪  blue 0 in hand + 1 on this floor = 1 for 1 closed blue door(s)
Position: (5,10)
Enemies remaining: 13/13
Uncollected items still on floor: ATK+1@(0,0), DEF+1@(10,0), ATK+1@(1,1), Red
↪  Potion (+75 HP)@(9,1)(+75HP), Yellow Key@(0,4), DEF+1@(2,4), ATK+1@(8,4),
↪  Yellow Key@(10,4), ATK+1@(0,6), ATK+1@(1,7), DEF+1@(9,7), Red Potion (+75
↪  HP)@(0,10)(+75HP), Yellow Key@(2,10), Blue Key@(8,10), Red Potion (+75
↪  HP)@(10,10)(+75HP)
DEF threshold tip: Collect +3 more DEF gems → Green Slime ×3 become FREE (saves
↪  72 HP; +3 DEF gems remain on map)
Goal: defeat Boss while keeping maximum HP. You do NOT need to kill all enemies
↪  — only fight enemies that grant stat gems, keys, or HP needed to reach and
↪  defeat the boss. There is NO shop on this floor. Boss at (5,0): ATK=20
↪  DEF=12 HP=50 | if fought now: UNWINNABLE — need ATK≥13 to deal damage (you
↪  have ATK=9) Use LOOKAHEAD TREE to find the cheapest path to the boss.
HP budget at max stats (all 5 ATK / 3 DEF gems collected): 625 HP available (HP
↪  + 225 potions) vs 468 HP for the 1 monsters that block the way to the boss +
↪  the boss → margin +157 HP. 11 other monsters (228 HP at max stats) block
↪  nothing on the way to the boss; they are not in this budget — fight one only
↪  if its tree shows a stat gem or potion you need, and its cost then comes off
↪  this margin.

Available actions:
  (Note: lookahead = total item/stat value unlocked by this action
  ↪  [HP-equivalent]. [...])
  1. [open_door] Yellow Door at (4,8) — net_lookahead=121(raw=233-gw_cost=112) |
  ↪  +75 HP potions behind it (collect any time)
 → next fight: ⚔ Green Slime@(1,3) dmg=24(6.0%), FREE@DEF≥7(+3), ⚔ Red
 ↪  Slime@(1,5) dmg=56(14.0%), FREE@DEF≥11(+7), ⚔ Red Slime@(1,6)
 ↪  dmg=56(14.0%), FREE@DEF≥11(+7), ⚔ Bat@(2,8) dmg=182(45.5%),
 ↪  FREE@DEF≥17(+13)
 → unlocks: Red Potion (+75 HP), Yellow Key
   → via [Red Slime@(1,6) [dmg=56 (14%), FREE if DEF≥11 (+7 needed)]]: Red Gem
   ↪  (+1 ATK) [STAT] ×2
     → via [Red Slime@(1,6) [...] → Red Slime@(1,5) [...]]: Blue Gem (+1 DEF)
     ↪  [STAT], Yellow Key
 → stat gems behind it (+2 ATK +1 DEF): 112 HP of fighting to reach them, then
 ↪  they save 20 HP on the other fights listed now → net -92 HP
 → keys: uses 1 yellow (yellow 1→2 after collecting what is behind it); behind
 ↪  it: +2 yellow; it exposes yellow door (1,2) → none
```

```
  2. [fight] Black Slime at (5,4) — monATK=5 monDEF=5 [magic-atk] [magic-atk:
  ↪  DEF useless — only ATK+MDEF reduce cost] — damage=55 (13.8%) hpAfter=345
  ↪  [SAFE]  ⚡[OPTIONAL — NOT required for boss, skip to conserve HP]
 → ORDER (this Black Slime at (5,4)): taking this fight now wastes 15 HP — after
 ↪  "Open yellow door at (6,8)" (+1 ATK +1 DEF; 48 HP of fighting to reach
 ↪  them) it costs 40 instead of 55; fight + route together: 88 instead of 103
  3. [open_door] Yellow Door at (6,8) — net_lookahead=170(raw=218-gw_cost=48) |
  ↪  +75 HP potions behind it (collect any time)
 → next fight: ⚔ Green Slime@(9,5) dmg=24(6.0%), FREE@DEF≥7(+3), ⚔ Green
 ↪  Slime@(9,6) dmg=24(6.0%), FREE@DEF≥7(+3), ⚔ Black Slime@(8,8)
 ↪  dmg=55(13.8%), magic-atk, ⚔ Red Slime@(9,3) dmg=56(14.0%), FREE@DEF≥11(+7)
 → unlocks: Blue Key, Red Potion (+75 HP)
   → via [Green Slime@(9,6) [...]]: Blue Gem (+1 DEF) [STAT]
     → via [Green Slime@(9,6) [...] → Green Slime@(9,5) [...]]: Red Gem (+1 ATK)
     ↪  [STAT], Yellow Key
 → stat gems behind it (+1 ATK +1 DEF): 48 HP of fighting to reach them, then
 ↪  they save 15 HP on the other fights listed now → net -33 HP
 → keys: uses 1 yellow (yellow 1→1 after collecting what is behind it); behind
 ↪  it: +1 yellow, +1 blue; it exposes yellow door (9,2) → none

Choose the best action. After your analysis, end your response with:
FINAL_CHOICE: <number>
```

Listing G.5: DeepSeek-V4-Flash at the first decision on hand01 (1,368 characters, complete; run log `20260929_073109_306644_hand01.txt_llm.json`).

```
Looking at the floor:

- We start with 400 HP, ATK=9, DEF=4, 1 yellow key.
- The boss needs ATK≥13 to even damage it. We currently have ATK=9, so we need
↪  at least +4 ATK from gems.
- Available ATK gems: (0,0), (1,1), (8,4), (0,6), (1,7) — five total.
- The two yellow doors both lead to stat gems and potions.

Action 1 (Yellow Door at (4,8)) leads to +2 ATK and +1 DEF, plus a potion and a
↪  yellow key. It costs some fights but those fights are cheap (Green Slimes
↪  24, Red Slimes 56). This gets us closer to ATK≥13.

Action 3 (Yellow Door at (6,8)) leads to +1 ATK and +1 DEF, plus a blue key and
↪  a potion. Also cheap fights.

Both are good. Which should be taken first?

Action 1 gives more ATK (+2) and also a DEF gem. Action 3 gives +1 ATK and +1
↪  DEF. Since we need ATK badly, Action 1 seems slightly better because it
↪  provides +2 ATK.

Also, Action 1 has a higher net lookahead (121 vs 170? Wait, actually Action 3
↪  has net_lookahead=170, Action 1 has 121). But lookahead includes potions and
↪  keys. Both give a potion. Action 3 gives a blue key which might be needed
↪  later for the blue door near the boss area.

However, the priority is getting ATK to 13. Action 1 gives +2 ATK, Action 3
↪  gives +1 ATK. So Action 1 gets us closer faster.

Also, the fights behind Action 1 are similar cost to Action 3. Both are safe.

I'll choose Action 1 first to boost ATK quickly.
```

Listing G.6: Qwen3.5-397B at the first decision on hand01 (12,168 characters, end of the reasoning; run log `20260929_185121_900081_hand01.txt_llm.json`).

```
[...]
   - Check Action 1 again. "net_lookahead=121". Action 3 "net_lookahead=170".
   - Higher lookahead means more value. Action 3 has higher value (170 vs 121).
   - Action 3 effCost is lower (better).

   - Final check:
   - I need to open a door. I have 1 Yellow Key.
   - Action 3 opens (6,8).
   - Action 1 opens (4,8).
   - Action 3 is recommended by the hint in Action 2 description ("after Open
   ↪  yellow door at (6,8)...").
   - Action 3 provides the Blue Key needed for the Boss door (5,2).
   - Action 3 is cheaper to clear.

   - Therefore, choose Action 3.
[...]
   - Reasoning summary:
   - Goal: Beat Boss. Need Blue Key (for door 5,2) and ATK gems (need +4).
   - Action 3 provides Blue Key and +1 ATK/+1 DEF for 48 HP cost.
   - Action 1 provides +2 ATK/+1 DEF for 112 HP cost.
   - Doing Action 3 first reduces the cost of Action 1 (the larger cost) via
   ↪  stat gain.
   - Action 2 hint explicitly suggests opening (6,8) [Action 3] before fighting
   ↪  Black Slime.
   - Action 3 has higher net_lookahead (170 vs 121).
   - Thus, Action 3 is the best choice.
```

Both models read the same two measures. DeepSeek-V4-Flash notes that the right door rates higher and overrides the measure with the count of ATK gems. Qwen3.5-397B follows the measures and the fight-order line of the optional Black Slime.

Note on AI Usage

This thesis was prepared with the help of the following Artificial Intelligence tools and services. The language models evaluated in the experiments are the object of the research and are not listed here; their use is described in Chapters 5 and 6.

- GitHub Copilot with Claude models (including Claude Opus 5.5) by Anthropic, for writing, revising and debugging parts of the framework, the web platform, the analysis scripts and the tests. Every change was reviewed, tested and run by the author. On the author's instruction, the assistant also launched experiment runs and analysis scripts and compiled tables and figures from the stored result files; every number was checked by the author.
- Claude Opus 5.5, for drafting the layouts of the four hand-designed floors from a description of the intended difficulty. Each draft was accepted only after it was solved exactly, tuned and checked by the procedure of Section 4.6.3.
- Gemini Deep Research and Perplexity, for finding candidate literature and summarising papers. Every cited reference was verified by the author on arxiv.org or on the publisher's page; references that could not be verified were not cited. In one tool's output, eleven entries attached an invented title to the identifier of an unrelated paper; none of them is cited.
- Claude Opus 5.5, for restructuring the chapters, drafting passages from the author's notes and results, checking wording, the consistency of numbers across chapters and conformity with a style guide derived from two theses supervised by the same first supervisor, and converting the manuscript to LaTeX. All numbers were checked against the stored result files.
- Gemini (Nano Banana Pro) by Google, for drawing the schematic Figures 1.1, 2.1–2.3, 5.1 and 5.2 from the author's specifications. Data figures were plotted from the result files, and platform screenshots were captured automatically.

The author takes full responsibility for the content of this thesis.

Declaration of Academic Honesty

I, King Wai Siu, hereby declare that I have not previously submitted the present work for other examinations. I wrote this work independently. All sources, including sources from the Internet, that I have reproduced in either an unaltered or modified form (particularly sources for texts, graphs, tables, and images), have been acknowledged by me as such. I understand that violations of these principles will result in proceedings regarding deception or attempted deception.

Bonn, 5 October 2026

Place, date	King Wai Siu