UCLouvain

epl

École polytechnique de Louvain

Deep Reinforcement Learning for Competitive Agents in MicroRTS

Architecture, Training, and Tournament Evaluation

Author: **Mathis Delsart**
Supervisor: **Éric Piette**
Readers: **Quentin Cappart, Achille Morenville, Benoît Ronval**
Academic year 2025–2026
Master [120] in Computer Science and Engineering

"Give me six hours to chop down a tree and I will spend the first four sharpening the axe."

Abraham Lincoln

"Victorious warriors win first and then go to war, while defeated warriors go to war first and then seek to win."

Sun Tzu

"Life is just like a game,
First you have to learn rules of the game,
And then play it better than anyone else."

Attributed to Albert Einstein

Abstract

Real-time strategy (RTS) games are among the most demanding benchmarks for sequential decision-making: players gather resources, coordinate many units, and plan over long horizons, in real time and within a combinatorial action space. AlphaStar reached Grandmaster level in StarCraft II, but at the cost of hundreds of accelerators running for weeks, beyond academic reach. MicroRTS distills these difficulties onto small grid maps while keeping training tractable on a modest budget, making it the reference academic testbed and the subject of an annual competition since 2017.

This master's thesis investigates deep reinforcement learning (DRL) for MicroRTS, guided by two questions: *which architectural and algorithmic design decisions most improve a MicroRTS agent*, and *whether one competitive with the strongest prior competition entries can be trained within an academic compute budget.* Taking RAISocketAI, the first DRL agent to win the competition, as reference and starting from the Gym-μRTS GridNet baseline, every design decision is evaluated in isolation before combining the best ones. The work contributes ***(i)*** a reproducible Conference on Games (CoG)-style tournament framework over twelve maps and fifteen reference agents under five ranking metrics; ***(ii)*** an extended, modular Java–Python environment stack with composable wrappers and vectorized self-play; ***(iii)*** the UECD architecture, fusing multi-scale convolution, entity-level Transformer reasoning, and bottleneck self-attention to cover an RTS network's local, relational, and global demands; ***(iv)*** a modular PPO pipeline whose mechanisms are ablated individually; and ***(v)*** a formal analysis of a discount-induced reward collapse under shaped-to-sparse annealing.

The resulting agent, `UECD-Best`, combines these under a two-phase opponent-curriculum fine-tuning schedule. On the *basesWorkers16x16A* map, it tops a 19-agent round-robin tournament (96.67% win rate, first on four of the five metrics) and wins 65.7% of its head-to-head games against RAISocketAI, using 9.47 GPU-days and ~350 M steps, below the ~23.6 GPU-days and ~500 M steps RAISocketAI reports for its small-map subset. A second agent, `UECD-MultiMap`, trained across five layouts of three different sizes, spreads its competence evenly with no per-map collapse, showing that the padded environment and a prioritized-level-replay curriculum make cross-layout training feasible, though it does not yet match the single-map specialist's peak.

The open-source pipeline released with this thesis offers a DRL substrate for future generalist agents and hybrid DRL/LLM systems, as the competition shifts toward language-model-based agents.

Keywords

Deep Reinforcement Learning • MicroRTS • Real-Time Strategy Games • Proximal Policy Optimization • Transformer • Game-Theoretic Evaluation • Reward Shaping

Acknowledgments

I would like to express my gratitude to the people who made this thesis possible.

First and foremost, I thank my supervisor ***Éric Piette*** and ***Achille Morenville*** for their guidance and enthusiasm throughout this work. Beyond that, *Éric Piette* encouraged me to submit a paper to *IEEE CoG 2026* (currently under review), a deeply rewarding experience.

I also thank the other members of my jury, ***Quentin Cappart*** and ***Benoît Ronval***, for their time and thoughtful evaluation of this thesis.

As part of *Éric Piette*'s peer-review process, I am grateful to ***Arnaud Bellière***, ***Florent Noiset***, ***Raphaël Vassart***, and ***Éric Piette*** for their feedback and proofreading.

I thank ***Rubens de Oliveira Moraes Filho*** and ***Chang Liu***, organizers of the annual MicroRTS competition, for their quick and helpful responses to my questions about the competition framework.

Computational resources have been provided by the ***Consortium des Équipements de Calcul Intensif (CÉCI)***, funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

I also acknowledge the use of ***Claude (Anthropic)*** as an AI assistant during this thesis for brainstorming ideas, assisting with coding, helping with English grammar, and refining the phrasing.

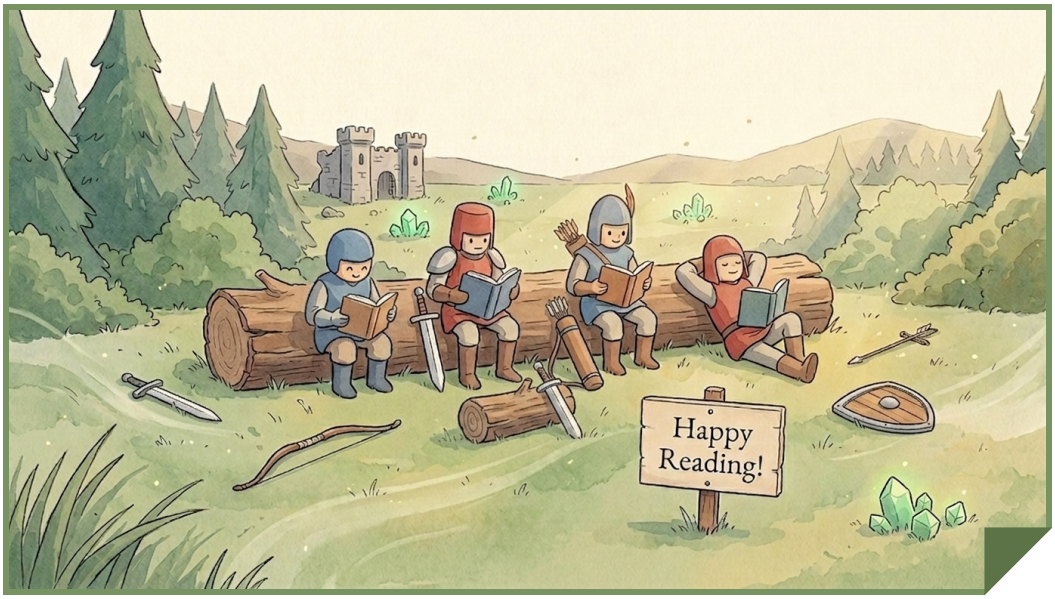
Finally, I thank my girlfriend and my parents for their moral support and their belief in me during the more demanding periods of this work. ❤️

"Last but not least, I want to thank me.
I want to thank me for believing in me.
I want to thank me for doing all this hard work.
I want to thank me for having no days off.
I want to thank me for never quitting."

Snoop Dogg, Hollywood Walk of Fame, 2018

Mathis
June 2026

Interlude



"Happy Reading!" and *"Endless Reading!"* illustrations generated with Google Gemini.

Contents

List of Figures x

List of Tables xii

List of Acronyms xii

1 Introduction 1

I Background

2 MicroRTS as a Sequential Decision Problem 4
- 2.1 Real-Time Strategy Games . . . 4
  - 2.1.1 Genre Overview . . . 4
  - 2.1.2 AI Research Challenges . . . 4
- 2.2 MicroRTS Environment . . . 5
  - 2.2.1 Annual AI Competition . . . 5
  - 2.2.2 Formal Game State . . . 6
  - 2.2.3 Unit Types and Combat . . . 7
  - 2.2.4 Specific Game Rules . . . 7
  - 2.2.5 MicroRTS-Py: Python Interface . . . 9
- 2.3 Mathematical Formulation of MicroRTS . . . 9
  - 2.3.1 From Markov Game to POMDP / MDP . . . 9
  - 2.3.2 MicroRTS as an MDP . . . 10

3 Reinforcement Learning Foundations 12
- 3.1 Policies and Value Functions . . . 12
- 3.2 Core RL Paradigms . . . 13
  - 3.2.1 Model-Free vs Model-Based . . . 13
  - 3.2.2 On-Policy vs Off-Policy . . . 13
- 3.3 Value-Based vs Policy-Based Methods . . . 13
  - 3.3.1 Value-Based Methods . . . 13
  - 3.3.2 Policy-Based Methods . . . 14
- 3.4 From Policy Gradients to PPO . . . 14
  - 3.4.1 Policy Gradients . . . 14
  - 3.4.2 Actor-Critic and Advantage Estimation . . . 14
  - 3.4.3 Proximal Policy Optimization . . . 15
- 3.5 The PPO Training Loop . . . 16
  - 3.5.1 The PPO Algorithm . . . 16
  - 3.5.2 PPO for MicroRTS . . . 17

II State of the Art

4 Classical Approaches in RTS Games 18
- 4.1 Rule-Based Approaches . . . 18
- 4.2 Search-Based Approaches . . . 19
  - 4.2.1 Monte Carlo Tree Search . . . 19
  - 4.2.2 Combinatorial Multi-Armed Bandits . . . 20

4.3 Hierarchical Approaches . . . 21

5 Deep Reinforcement Learning Approaches in RTS Games 22
5.1 Major DRL Breakthroughs . . . 22
5.1.1 Foundational Breakthroughs in Turn-Based Games . . . 22
5.1.2 Scaling DRL to RTS Games . . . 22
5.2 DRL at Full RTS Scale: AlphaStar . . . 23
5.2.1 Observation Encoding . . . 23
5.2.2 Action Space . . . 24
5.2.3 Training Pipeline . . . 24
5.2.4 League Training and PFSP . . . 24
5.3 DRL at MicroRTS Scale . . . 24
5.3.1 Early Exploration (2019–2020) . . . 25
5.3.2 Gym-μRTS and GridNet (2020–2021) . . . 25
5.3.3 RAISocketAI (2023–2024) . . . 25
5.3.4 Recent Directions (2024–2025) . . . 26
5.4 Thesis Contributions . . . 26

III Contributions

6 MicroRTS Tournament Framework 27
6.1 Tournament Infrastructure . . . 27
6.2 Evaluation Maps . . . 28
6.3 Benchmark Agents . . . 28
6.4 Ranking Metrics . . . 30
6.4.1 Primary Metric: Win Rate . . . 30
6.4.2 Complementary Game-Theoretic Metrics . . . 30

7 MicroRTS Environment Stack 31
7.1 Java-Python Bridge . . . 32
7.1.1 *JPype* Integration . . . 32
7.1.2 End-to-End Step Cycle . . . 32
7.2 Observation Encoding . . . 32
7.3 Action Space and Masking . . . 34
7.4 Reward Functions . . . 35
7.5 Vectorized Environments . . . 36
7.5.1 Environment Classes . . . 36
7.5.2 Padding for Multi-Map Training . . . 36
7.6 Composable Wrappers . . . 37

8 Spatial and Entity-Based Neural Architectures 38
8.1 Common Skeleton and Reasoning Axes . . . 38
8.1.1 Actor-Critic Skeleton . . . 38
8.1.2 Three Reasoning Axes . . . 38
8.2 Axis 1: Local Spatial Reasoning . . . 39
8.3 Axis 2: Long-Range Relational Reasoning . . . 40
8.4 Axis 3: Global Spatial Reasoning . . . 41
8.5 UECD: Final Architecture . . . 42
8.6 Architecture Summary . . . 42

9 Training Procedure and Mechanisms **43**
9.1 PPO Baseline Configuration . . . 43
9.2 Action Sampling Refinements . . . 45
9.3 Reward and Value Signal . . . 46
9.4 Opponent Curriculum and Self-Play . . . 49
9.5 Auxiliary Representation Learning . . . 51
9.6 Generalization . . . 52
9.7 Architectural Refinements . . . 52

10 Experimental Results and Analysis **53**
10.1 Experimental Setup . . . 53
10.2 Architecture Ablation Analysis . . . 54
10.2.1 Axis 1: Local Spatial Reasoning . . . 54
10.2.2 Axis 2: Long-Range Relational Reasoning . . . 56
10.2.3 Axis 3: Global Spatial Reasoning . . . 56
10.2.4 Final Architecture Choice . . . 56
10.3 Feature Ablation Analysis . . . 57
10.3.1 Reliable Gains: Perception and Signal Allocation . . . 59
10.3.2 Second Tier: Strong but Budget-Limited . . . 60
10.3.3 The Middle Band: Deferred Judgment, Not Failure . . . 60
10.3.4 Negative Findings: Two Distinct Failure Modes . . . 61
10.3.5 Synthesis . . . 61
10.3.6 Final Feature Selection . . . 62
10.3.7 Selection Validation . . . 62
10.4 Building the Single-Map Agent . . . 63
10.4.1 The Sparse-Reward Trap: A Discount-Induced Rush Collapse . . . 63
10.4.2 Two-Phase Opponent-Pool Fine-Tuning . . . 67
10.4.3 The Emergent Ranged-Only Strategy . . . 68
10.5 Evaluating the Single-Map Agent . . . 69
10.5.1 Win Rate as the Primary Performance Signal . . . 69
10.5.2 Game-Theoretic Metrics as Robustness Checks . . . 71
10.6 Generalization Probes . . . 75
10.7 Behavior-Cloning Warmstart for PPO . . . 76
10.7.1 BC+VF Pre-Training . . . 76
10.7.2 Comparison Setup . . . 76
10.7.3 Results and Discussion . . . 77
10.8 Multi-Map Agent . . . 78
10.8.1 Setup . . . 78
10.8.2 Evaluation . . . 78
10.8.3 Hypothesis Assessment . . . 81

IV Discussion and Conclusions

11 Discussion **82**
11.1 Findings in Relation to Prior Work . . . 82
11.1.1 Where the Findings Differ . . . 82
11.1.2 Where the Findings Agree . . . 82
11.2 Implications for Practitioners . . . 83
11.3 Implications for Researchers . . . 83

12 Limitations **84**
12.1 Deliberate Methodological Choices . . . 84
12.2 Open Residual Limitations . . . 85

13 Future Work **86**
13.1 Short-Term Directions . . . 86
13.1.1 Hybrid DRL and LLM Agents . . . 86
13.1.2 Hierarchical Policy Decomposition . . . 86
13.1.3 Neuro-Symbolic Integration . . . 87
13.1.4 Adversarial Co-Evolution and PCG . . . 87
13.2 Long-Term Vision . . . 87
13.2.1 Spatial State-Action Features . . . 88
13.2.2 Cross-Variant Policy and Value Transfer . . . 88
13.2.3 Self-Play Distribution Shaping . . . 88
13.2.4 A General RTS Game System . . . 89

14 Conclusion **90**

V Appendices

A Derivations Underlying PPO **92**
A.1 Policy Gradient Theorem . . . 92
A.2 Importance-Sampled Surrogate Loss . . . 93

B Evaluation Map Visualizations **94**

C Game-Theoretic Evaluation Metrics **95**
C.1 Nash Averaging . . . 95
C.2 α-Rank . . . 95
C.3 Copeland Score . . . 96
C.4 Regret . . . 96
C.5 An Illustrative Example . . . 97

D UECD Architecture: Full Specification **98**

Bibliography **99**

List of Figures

2.1 Three well-known RTS games . . . 4
2.2 Typical MicroRTS game state . . . 5
2.3 MicroRTS competition timeline (2017–2026) . . . 5
2.4 Unit type relationships (rock-paper-scissors) . . . 7

3.1 Agent-environment interaction loop (MDP) . . . 12
3.2 PPO training cycle . . . 16
3.3 PPO training loop applied to MicroRTS . . . 17

4.1 AI paradigms for RTS games . . . 18
4.2 Branching factors across board and RTS games . . . 19
4.3 Joint action space vs CMAB factorization . . . 20
4.4 Hierarchical decomposition (StrategyTactics) . . . 21

5.1 Major DRL milestones in strategic games . . . 22
5.2 AlphaStar architecture . . . 23

6.1 Three-phase architecture of the tournament pipeline . . . 27

7.1 Java–Python training bridge stack . . . 31
7.2 Standard observation encoding (example cells) . . . 34
7.3 Standard vs destination-aware filtered masking . . . 35
7.4 Padded environment for multi-map training . . . 36

8.1 CBAM-ResBlock (simplified view) . . . 40
8.2 Entity Transformer mechanism (simplified view) . . . 41
8.3 UECD architecture (simplified view) . . . 42

9.1 Training mechanisms on the PPO loop . . . 44
9.2 Hierarchical sub-action masking . . . 46
9.3 Piecewise-linear multi-phase scheduler . . . 47
9.4 PAE on an incomplete rollout segment . . . 49
9.5 Automated opponent curriculum . . . 49
9.6 MCW transition weight vs win rate . . . 50

10.1 Top-5 vs all-features win-rate trajectories . . . 63
10.2 Phase schedule for the 300 M-step run . . . 64
10.3 Discount-induced rush collapse . . . 64
10.4 Rush fragility on held-out opponents . . . 67
10.5 Final tournament standings (single-map, 19 agents) . . . 70
10.6 Head-to-head win-rate matrix (single-map, 19-agent) . . . 70
10.7 Copeland scores across the tournament . . . 72
10.8 α-Rank stationary mass vs selection intensity across the tournament . . . 73
10.9 Nash equilibrium scores across the tournament . . . 74
10.10 Worst-case and mean-case robustness across the tournament . . . 74
10.11 Generalization probes: scale and layout shift . . . 75

10.12 BC-warmstart vs from-scratch PPO 77
10.13 Per-map win rate over the five-map pool 79
10.14 Multi-map field: home-map standings 80
10.15 Global head-to-head win rates (five-map pool) 80

B.1 The eight open evaluation maps 94
B.2 The four closed evaluation maps 94

C.1 Four tournament metrics on a 4-agent example 97

D.1 UECD architecture (fully annotated) 98

List of Tables

2.1 Unit attributes and combat parameters in MicroRTS . . . 8
2.2 Action availability by unit type . . . 8
2.3 Action parameters and execution details . . . 8

4.1 Representative techniques by decision scale in classical RTS AI . . . 21

6.1 Characteristics of the twelve evaluation maps . . . 28
6.2 Built-in MicroRTS baseline agents . . . 29
6.3 Competition agents and GridNet baseline (2017–2024) . . . 29
6.4 Ranking metrics and the evaluation question each addresses . . . 30

7.1 Standard observation encoding (29 channels) . . . 33
7.2 Extended observation encoding (73 channels) . . . 33
7.3 Factored action space per grid cell . . . 34
7.4 Reward functions available during training . . . 35

8.1 Overview of the seven architectures . . . 42

9.1 Baseline PPO hyperparameters . . . 43
9.2 Catalog of flag-gated training mechanisms . . . 44
9.3 Value head configuration . . . 47
9.4 Auxiliary prediction heads . . . 51

10.1 Baseline experiment setup . . . 53
10.2 Architecture ablation: summary across seeds . . . 54
10.3 Architecture ablation: per-seed win rates . . . 55
10.4 Feature ablation: summary across seeds . . . 57
10.5 Feature ablation: per-seed win rates . . . 58
10.6 Two-phase fine-tuning schedule . . . 67
10.7 Per-phase training budget for UECD-`Best` . . . 68
10.8 Top-3 agents per game-theoretic metric . . . 71

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BC	Behavior Cloning
CBAM	Convolutional Block Attention Module
CIG	Conference on Computational Intelligence and Games
CLI	Command Line Interface
CMAB	Combinatorial Multi-Armed Bandit
CNN	Convolutional Neural Network
CoG	Conference on Games
CPU	Central Processing Unit
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EMA	Exponential Moving Average
ETA	Estimated Time of Arrival
FIFO	First-In First-Out
GAE	Generalized Advantage Estimation
GELU	Gaussian Error Linear Unit
GPU	Graphics Processing Unit
HL-Gauss	Histogram Loss with Gaussian Targets
HP	Hit Points
HPC	High-Performance Computing
IDRTMinimax	Iterative Deepening Real-Time Minimax
IMPALA	Importance-Weighted Actor-Learner Architecture
InfoNCE	Information Noise-Contrastive Estimation
IS	Importance-Sampled
JNI	Java Native Interface
JPS	Jump Point Search
JVM	Java Virtual Machine
KL	Kullback-Leibler
LLM	Large Language Model
LSTM	Long Short-Term Memory
MCTS	Monte Carlo Tree Search

MCW	Matchup Competitiveness Weighting
MDP	Markov Decision Process
MLP	Multilayer Perceptron
MOBA	Multiplayer Online Battle Arena
MSE	Mean Squared Error
P0	Player 0
P1	Player 1
PAE	Partial Advantage Estimator
PCG	Procedural Content Generator
PER	Prioritized Experience Replay
PFSP	Prioritized Fictitious Self-Play
PLR	Prioritized Level Replay
POMDP	Partially Observable Markov Decision Process
PopArt	Preserving Outputs Precisely, while Adaptively Rescaling Targets
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
ResNet	Residual Network
RL	Reinforcement Learning
RLaR	Reinforcement Learning as a Rehearsal
RTS	Real-Time Strategy
SCC	StarCraft Commander
SE	Squeeze-and-Excitation
SL	Supervised Learning
SLURM	Simple Linux Utility for Resource Management
SPP	Spatial Pyramid Pooling
SPS	Steps Per Second
SSS	Strategy Selection Search
TD	Temporal-Difference
TPU	Tensor Processing Unit
TRPO	Trust Region Policy Optimization
U-Net	U-shaped Neural Network
UECD	U-Net-Entity-CBAM-Deep
UPGO	Upgoing Policy Update
VF	Value Function
WCCI	World Congress on Computational Intelligence
WR	Win Rate

1 Introduction

Many real-world problems require multiple agents to make decisions simultaneously, in real time, and with limited information: from landing drone swarms [1] to coordinating warehouse robots [2] and piloting fighter jets in air combat [3]. In such settings, agents must allocate scarce resources and anticipate others' actions despite delayed feedback. Real-time strategy (RTS) games pose the same challenge in miniature [4]: multiple players gather resources, build armies, and fight on a shared map, controlling every unit at every moment. The motivation goes beyond academic curiosity: methods developed in RTS games have already reached expert level in demanding real-time control, from championship-level car racing [5] to real-world fusion-reactor plasma control [6]. This is what makes RTS games one of the most demanding benchmarks for sequential decision-making [7].

AlphaStar [8] demonstrated that deep reinforcement learning (DRL) can reach Grandmaster level (*top* 0.15%) in StarCraft II, a commercial RTS game whose hundreds of unit types and rich action parameterization yield roughly 10^{26} legal actions per timestep. However, training required 384 tensor processing units (TPUs) over 44 days and substantial engineering effort, far beyond academic reach. MicroRTS [9] offers an accessible alternative that distills the RTS problem to economy and combat on small grid maps, yet retains the key difficulties: combinatorial action space, multi-unit coordination, and sparse long-horizon planning [4]. Since 2017, an annual competition [10] at the IEEE Conference on Games (CoG), formerly the Conference on Computational Intelligence and Games (CIG), has attracted agent submissions. The Gym-μRTS Python wrapper [11] has made the environment directly usable for reinforcement learning (RL).

This master's thesis investigates the application of DRL to MicroRTS, aiming to produce a competitive agent within an academic compute budget. Because the 2026 MicroRTS competition has shifted its focus to large language model (LLM)-based agents, direct participation falls outside the scope of this work. Instead, the format of the 2023 competition is replicated: a benchmark of fifteen agents spanning seven years of MicroRTS history is assembled across twelve maps, yielding a reproducible evaluation framework. RAISocketAI [12], the first DRL agent to win the competition, serves as the primary competitive reference throughout. Starting from the GridNet baseline introduced alongside Gym-μRTS [11], each architectural and algorithmic improvement is evaluated in isolation before the best components are combined into final agents.

This work is guided by two research questions:

- **RQ1.** Which architectural and algorithmic design decisions most improve a MicroRTS agent's performance, and how robust this effect is across seeds?
- **RQ2.** Can an agent competitive with the strongest prior competition entries be trained within an academic compute budget?

To answer these questions, this thesis contributes a ***(i)*** reproducible CoG-style tournament framework, ***(ii)*** an extended and modular Java–Python environment stack, ***(iii)*** the UECD architecture, ***(iv)*** a modular training pipeline whose mechanisms are ablated in isolation, ***(v)*** a formal analysis of a discount-induced reward collapse, and ***(vi)*** the resulting agent named `UECD-Best`.

On the standard *basesWorkers16x16A* map, `UECD-Best` tops a 19-agent round-robin tournament with a **96.67%** overall win rate (WR) and holds a **65.7%** head-to-head WR against RAISocketAI. This is achieved at a training cost of **9.47** graphics processing unit (GPU)-days and ~**350 M** environment steps, below the ~23.6 GPU-days and ~500 M steps reported by RAISocketAI for its small-map subset ($\leq$ 16×16), trading that agent's multi-layout breadth for single-layout depth.

A paper distilling the contributions and results of this thesis has been submitted to IEEE CoG 2026 (currently under review)[1]. The complete codebase is open-sourced[2], alongside a companion website hosting recorded `UECD-Best` matchups[3].

The work is structured as follows:

Part I — Background

- **Chapter 2 — MicroRTS as a Sequential Decision Problem.** Introduces real-time strategy games and their challenges for Artificial Intelligence (AI). The MicroRTS environment is presented in detail. The problem is formalized as a Markov Decision Process (MDP) under full observability, defining state space, action space, transition dynamics, and reward signal.
- **Chapter 3 — Reinforcement Learning Foundations.** Covers the core RL concepts required for the rest of this work. The chapter concludes with a detailed presentation of Proximal Policy Optimization (PPO) and an overview of the end-to-end training loop applied to the MicroRTS environment.

Part II — State of the Art

- **Chapter 4 — Classical Approaches in RTS Games.** Reviews rule-based, search-based, and hierarchical decomposition paradigms for RTS game AI. Each paradigm is illustrated with concrete agents from the MicroRTS competition that appear in the evaluation benchmark.
- **Chapter 5 — Deep Reinforcement Learning Approaches in RTS Games.** Reviews the milestones that brought DRL from board games to RTS games. AlphaStar is presented in detail as the most influential reference for this work. The chapter surveys DRL work on MicroRTS, from the first Gym-μRTS agent to RAISocketAI, and positions the contributions of this thesis.

Part III — Contributions

- **Chapter 6 — MicroRTS Tournament Framework.** Defines the tournament infrastructure for comparing agents throughout this thesis. The chapter presents the twelve evaluation maps, the fifteen benchmark agents assembled to replicate the 2023 CoG competition format, and the five ranking metrics used to assess agent performance.
- **Chapter 7 — MicroRTS Environment Stack.** Extends the Gym-μRTS framework with substantial modifications to both the Java and Python layers. The Java–Python bridge, vectorized game client, standard and extended observation encodings, action space structure, reward functions, and composable wrapper stack are described in detail.

[1] Titled *"Combining Spatial and Entity-Based Reasoning for Competitive MicroRTS via U-Net and Transformers"*.
[2] M. Delsart, *microrts-drl-uecd*, 2026, GitHub: https://github.com/mathisdelsart/microrts-drl-uecd.
[3] Available at https://mathisdelsart.github.io/microrts-drl-uecd-website.

- **Chapter 8 — Spatial and Entity-Based Neural Architectures.** Organizes the architectural requirements of an RTS network into three reasoning axes (local spatial, long-range relational, global spatial). Presents the seven architectures explored, from the GridNet baseline to the final UECD design. Each architecture is introduced as an incremental modification of the previous one, with the specific motivation and expected effect of each change.
- **Chapter 9 — Training Procedure and Mechanisms.** Specifies the PPO training procedure built on the chosen architecture, then the catalog of optional, flag-gated mechanisms layered on top of it. The mechanisms are presented from those closest to per-step action sampling outward to global generalization: action sampling, reward and value shaping, the opponent curriculum, auxiliary representation learning, generalization, and architectural refinements.
- **Chapter 10: Experimental Results and Analysis.** Reports the empirical evaluation of all contributions. Architecture and feature ablations isolate per-component effects and feed into the single-map agent `UECD-Best`, built via two-phase opponent-pool fine-tuning and motivated by a theoretical analysis of the discount-induced rush collapse observed on a first long-horizon run. `UECD-Best` is evaluated in a 19-agent round-robin tournament, and probed for robustness under layout and scale shift. Two complementary experiments close the chapter: a multi-map variant validating cross-layout training, and a behavior-cloning warmstart quantifying the compute-efficiency benefit of pre-training on bot replays.

Part IV — Discussion and Conclusions

- **Chapter 11 — Discussion.** Steps back from the per-experiment analysis to interpret the results as a whole: how the findings depart from and confirm prior work, and what they imply for practitioners and for researchers.
- **Chapter 12: Limitations.** Distinguishes deliberate methodological choices (trade-offs adopted by design) from open residual limitations (gaps observed in the experimental results).
- **Chapter 13: Future Work.** Outlines two horizons of research directions: short-term extensions of the existing artifacts, and a longer-term program toward general RTS AI built on a cross-game research line on transferable representations.
- **Chapter 14 — Conclusion.** Summarizes the main findings and contributions of this thesis.

Part V — Appendices

- **Appendix A — Derivations Underlying PPO.** Provides the policy gradient theorem and importance-sampled surrogate loss derivations referenced in Chapter 3.
- **Appendix B: Evaluation Map Visualizations.** Shows the twelve evaluation maps used by the tournament framework of Chapter 6 and the experiments of Chapter 10.
- **Appendix C — Game-Theoretic Evaluation Metrics.** Provides the mathematical definitions of Nash averaging, α-Rank, Copeland score, and regret used in tournament analysis.
- **Appendix D: UECD Architecture: Full Specification.** Provides the fully annotated UECD diagram with all channel widths and tensor dimensions needed to reproduce the network.

MicroRTS as a Sequential Decision Problem

2

This chapter introduces the MicroRTS environment and formalizes it as a sequential decision-making problem. The chapter first situates MicroRTS within the broader context of real-time strategy games and identifies the algorithmic challenges they pose for Artificial Intelligence, then describes the MicroRTS environment in detail and casts the resulting problem as a Markov Decision Process under full observability. The notations and abstractions introduced here form the common foundation upon which all subsequent chapters build.

2.1 Real-Time Strategy Games

2.1.1 Genre Overview

RTS games are competitive video games[1] in which two or more players simultaneously gather resources, construct bases, and command armies [4]. Unlike turn-based games, decisions unfold in real time, forcing players to manage their economy, coordinate heterogeneous units, and react to opponent actions simultaneously. Skilled play requires both *macro-management* (economy, production, and build orders) and *micro-management* (individual unit positioning and combat control). Figure 2.1 shows three well-known examples of the genre.

(a) *StarCraft II* (b) *Warcraft III* (c) *Age of Empires II*

Figure 2.1: Three well-known RTS games. Images from Blizzard Entertainment (2010, 2002) and World's Edge & Xbox Game Studios (2019).

2.1.2 AI Research Challenges

RTS games pose several well-identified challenges for AI [4, 7]. First, the joint action space grows combinatorially with the number of units on the field. Second, decisions are made under real-time constraints, leaving no room for exhaustive deliberation. Third, games last thousands of timesteps with a sparse terminal reward signal (win, loss, or draw) providing no intermediate feedback. Fourth, the opponent adapts continuously, making the environment non-stationary from a single agent's perspective. Finally, skilled play requires both macro-management and micro-management simultaneously, forcing the agent to make decisions at multiple temporal scales. These challenges are addressed progressively throughout Chapters 2–5.

[1] Not to be confused with Multiplayer Online Battle Arena (MOBA) games such as League of Legends (Riot Games, 2009) or Dota 2 (Valve Corporation, 2013), which restrict each player to a single hero rather than full army and economy management.

2.2 MicroRTS Environment

Commercial RTS games such as StarCraft II have served as prominent benchmarks for RL research [13], but their scale precludes controlled experimentation, as illustrated by AlphaStar [8], whose training demanded prohibitive computational resources to interface with the game.
MicroRTS[2] avoids these costs by providing a minimalist RTS environment designed for AI research [9]. Built on a Java engine, it uses small grid-based maps and retains the core RTS mechanics (resource gathering, base building, army control) with simplified movement and reduced unit diversity. The environment also supports optional fog-of-war[3] and stochastic damage[4] [14, 15], though neither is used in this thesis. This reduced complexity makes experiments faster and more reproducible, while the combinatorial action space and simultaneous unit control keep the problem challenging. Figure 2.2 illustrates the main elements of the MicroRTS simulator interface.

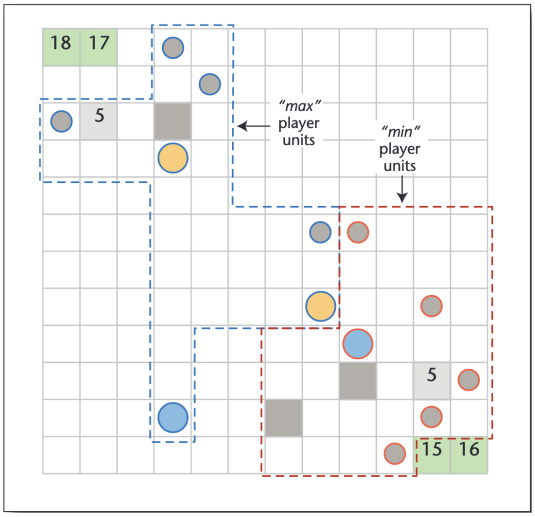


Figure 2.2: Typical game state on a 12×12 grid map. Square entities denote bases (light gray), barracks (dark gray), and resource mines (green); circular units represent workers (small) and military units (large). Numbers on resource mines indicate remaining stock; numbers on bases indicate the player's current resources. Blue units belong to the *max player*, red to the *min player* [10].

2.2.1 Annual AI Competition

MicroRTS has been supported by an annual AI competition since 2017[5], hosted at various IEEE conferences (CIG, CoG, and WCCI, the World Congress on Computational Intelligence). The evaluation benchmark assembled in Chapter 6 draws on the open-source winners of these competition editions. Figure 2.3 summarizes the competition timeline.

Figure 2.3: Timeline of the MicroRTS competition (2017–2026) showing hosting conferences.

[2]S. Ontañón, MicroRTS, 2012, GitHub: https://github.com/Farama-Foundation/MicroRTS.
[3]Fog-of-war is a visibility mechanic where units can only see within a limited radius, concealing the rest of the map.
[4]Stochastic damage adds a random multiplier to attack damage; disabled by default to keep transitions deterministic given fixed actions.
[5]S. Ontañón, MicroRTS AI competition, Google Sites: https://sites.google.com/site/micrortsaicompetition/.

The 2026 edition marks a shift toward evaluating LLMs as game-playing agents[6] [16, 17], a direction orthogonal to this thesis. Since no RL-focused edition is available, a tournament framework is developed in Chapter 6 to reproduce the 2023 competition format as closely as possible, enabling fair and reliable comparison against past competition winners.

2.2.2 Formal Game State

MicroRTS environments are defined over discrete two-dimensional grid maps[7].
Let $H, W \in \mathbb{N}_0$[8] denote the map height and width, respectively. The set of valid cell coordinates is:

$$\mathcal{G} = \{0, \dots, W-1\} \times \{0, \dots, H-1\} \subset \mathbb{N}^2. \tag{2.1}$$

The terrain layout is described by a binary map:

$$\mathcal{M} : \mathcal{G} \to \{0,1\}, \quad (x,y) \mapsto \begin{cases} 1 & \text{if cell } (x,y) \text{ is passable,} \\ 0 & \text{if cell } (x,y) \text{ is a terrain obstacle.} \end{cases} \tag{2.2}$$

Since $\mathcal{M}$ is fixed throughout a game, it is treated as a game parameter rather than part of the dynamic state. At any timestep t, the complete game state is defined as the tuple:

$$\boldsymbol{s}_t = \left(\tau_t,\ \boldsymbol{\rho}_t,\ \mathcal{U}_t\right), \tag{2.3}$$

where $\tau_t \in [\![0, T_{\max}]\!]$[9] is the current simulation timestep, $\boldsymbol{\rho}_t = (\rho_t^1, \rho_t^2) \in \mathbb{N}^2$ the accumulated resource count for each player, and $\mathcal{U}_t = \{\mathbf{u}_1^t, \dots, \mathbf{u}_{n_t}^t\}$ the set of n_t entities present on the map.
Each entity $\mathbf{u}_i^t \in \mathcal{U}_t$ is characterized by the tuple:

$$\mathbf{u}_i^t = \left(\text{player}_i,\ \mathbf{pos}_i^t,\ \text{type}_i,\ \text{hp}_i^t,\ \text{res}_i^t,\ \mathbf{action}_i^t\right), \tag{2.4}$$

where the components are defined as follows:

- $\text{player}_i \in \{p_1, p_2, \text{neutral}\}$: entity ownership;
- $\mathbf{pos}_i^t \in \mathcal{G}$: grid position;
- $\text{type}_i \in \mathcal{K} = \{\text{Resource}, \text{Base}, \text{Barracks}, \text{Worker}, \text{Light}, \text{Heavy}, \text{Ranged}\}$: entity type;
- $\text{hp}_i^t \in [\![0, \text{HP}_{\max}(\text{type}_i)]\!]$: current hit points (HP);
- $\text{res}_i^t \in \mathbb{N}$: carried resources for Workers (0 or 1), remaining stock for Resource nodes ($\in [\![0, \text{res}_{\max}^i]\!]$), and 0 for all others;
- $\mathbf{action}_i^t = (\text{action_type}_i^t,\ \boldsymbol{\varphi}_i^t, \text{timer}_i^t)$: action being executed, where $\text{action_type}_i^t \in \mathcal{A}_{\text{type}} = \{\text{None}, \text{Move}, \text{Harvest}, \text{Return}, \text{Produce}, \text{Attack}\}$, parameters $\boldsymbol{\varphi}_i^t$ (Table 2.3), and timer_i^t the remaining steps before completion.

The entity set $\mathcal{U}_t$ evolves over time: it grows when a *Produce* action completes, spawning a new unit or creating a new structure, and it shrinks when an entity's hit points reach zero through combat damage or when a Resource node is fully depleted by harvesting. In both cases the entity does not appear in $\mathcal{U}_{t+1}$ and its cell becomes unoccupied.

[6] Official 2026 MicroRTS competition description: https://github.com/drchangliu/MicroRTS/blob/master/COMPETITION.md.
[7] The formalization presented in this section is derived from analysis of the MicroRTS source code and game documentation.
[8] This work uses the convention $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{N}_0 = \{1, 2, 3, \dots\}$.
[9] $[\![a, b]\!]$ denotes the set of integers $\{a, a+1, \dots, b\}$ for $a, b \in \mathbb{Z}$.

2.2.3 Unit Types and Combat

Each entity type in $\mathcal{K}$ fulfills a distinct strategic role. Their quantitative attributes are summarized in Table 2.1, available actions in Table 2.2, and action details in Table 2.3.

Resources are static and neutral; they are the only source of income. Each resource node contains a finite stock that can be harvested by Workers and deposited at Bases.

Buildings are static, player-owned structures that define production capabilities. Bases enable Worker production and accept resource deposits. Barracks unlock military production (Light, Heavy, and Ranged units). Neither building can attack.

Mobile units perform all active tasks: resource collection, construction, and combat. Workers are the only units that can harvest, return resources, and construct buildings, but their 1 HP makes them fragile and vulnerable to any attack. They can still attack at melee range with low damage, enabling early-game worker rushes. The three military types form a rock-paper-scissors cycle (Figure 2.4): Light counters Ranged through superior mobility, Ranged counters Heavy by attacking from distance, and Heavy counters Light with high damage. This cyclic dominance forces players to mix unit compositions rather than mass-produce one type.

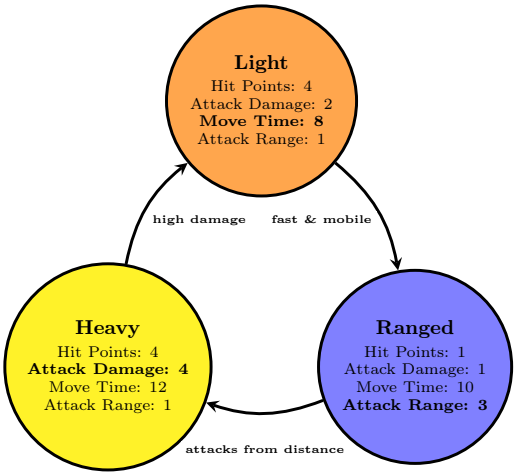


Figure 2.4: Unit type relationships in MicroRTS (rock-paper-scissors cycle): Light beats Ranged thanks to faster movement, Ranged beats Heavy by dealing damage from outside melee range, and Heavy beats Light by dealing fatal damage in one hit. Colors match the in-game units, and bold marks each unit's dominant attribute.

2.2.4 Specific Game Rules

Action execution. Once issued, an action runs to completion; new commands are accepted only when the unit is idle. Any idle unit without a valid command defaults to a *None* action.

Combat. Attacks depend only on Euclidean distance[11], so ranged units can fire through walls and other entities.

Victory. A player wins by eliminating all entities in $\mathcal{U}_t$ owned by the opponent while retaining at least one entity of their own (any structure or mobile unit). The game ends in a draw if both players lose their last entities on the same step, or if τ_t reaches the time limit $T_{\max}$ (commonly 3 000 to 8 000 steps); in the latter case, entity counts and resource stocks are not used as tie-breakers.

[10] Farama Foundation, MicroRTS game definition, 2018, GitHub Wiki: https://github.com/Farama-Foundation/MicroRTS/wiki/Game-Definition.

[11] For two grid positions $(x_1, y_1), (x_2, y_2) \in \mathcal{G}$, the Euclidean distance is $d(\mathbf{p}_1, \mathbf{p}_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. A unit at $\mathbf{p}_1$ can attack a target at $\mathbf{p}_2$ if and only if $d(\mathbf{p}_1, \mathbf{p}_2) \leq r$, where r is the unit's attack range.

Table 2.1: Unit attributes and combat parameters in MicroRTS.[10]

Unit Type	Production Cost	Production Time	Hit Points	Move Time	Attack Time	Attack Damage	Attack Range
Static entities							
Resource†	—	—	—	—	—	—	—
Base	10	250	10	—	—	—	—
Barracks	5	200	4	—	—	—	—
Mobile units							
Worker‡	1	50	1	10	5	1	1
Light	2	80	4	8	5	2	1
Heavy	2	120	4	12	5	4	1
Ranged	2	100	1	10	5	1	3

†Resource nodes have no combat or production role; their only attribute is the remaining resource stock.
‡Workers also perform Harvest (20 steps) and Return (10 steps) actions; see Table 2.3.

Table 2.2: Action availability by unit type in MicroRTS.[10]

Unit Type	None	Move	Attack	Harvest	Return	Produce	$\lvert\mathcal{A}_i\rvert^{\dagger}$
Static entities							
Resource	✓	—	—	—	—	—	1
Base	✓	—	—	—	—	✓	5
Barracks	✓	—	—	—	—	✓	13
Mobile units							
Worker	✓	✓	✓	✓	✓	✓	70
Light	✓	✓	✓	—	—	—	54
Heavy	✓	✓	✓	—	—	—	54
Ranged	✓	✓	✓	—	—	—	54

Checkmarks (✓) indicate available actions; dashes (—) indicate unavailable actions.
† $\lvert\mathcal{A}_i\rvert$ is the API-level (Application Programming Interface) syntactic cardinality discussed in Section 2.3.2; runtime feasibility is enforced by an invalid-action mask.

Table 2.3: Action parameters $\boldsymbol{\varphi}$ and execution details in MicroRTS.[10]

Action	Parameters	Notes
None	Duration	Unit idles for specified time.
Move	Direction†	Move one cell in specified direction. Fails if blocked or occupied.
Attack	Target (x, y)	Melee range 1, Ranged range 3. Damage applied on final cycle only.
Harvest	Direction†	Collect 1 resource from adjacent node. Worker only, empty inventory.
Return	Direction†	Deposit carried resource to adjacent Base. Worker only, full inventory.
Produce	Direction†, Type‡	Base produces Workers; Barracks produces Light, Heavy, or Ranged; Worker constructs Base or Barracks. Requires resources.

†Direction $\in \mathcal{D} = \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$ | ‡Type $\in \mathcal{K} \setminus \{\text{Resource}\}$

2.2.5 MicroRTS-Py: Python Interface

While MicroRTS is implemented in Java, most RL research relies on Python frameworks such as *PyTorch* [18] and *Stable-Baselines3* [19]. The MicroRTS-Py[12] (formerly Gym-μRTS) wrapper [11] bridges this gap via *JPype*[13] with a *Gymnasium*-compatible [20] Python interface to the MicroRTS engine, exposing observations, action spaces, and reward signals usable by standard RL training loops. This wrapper provides the foundation of the environment stack used in this thesis, extended with custom environments and training wrappers (Chapter 7).

2.3 Mathematical Formulation of MicroRTS

This section formalizes MicroRTS as a sequential decision-making problem, providing a common foundation for the classical planning methods, search-based approaches, and reinforcement learning algorithms developed in subsequent chapters.

2.3.1 From Markov Game to POMDP / MDP

RTS games involve $N \geq 2$ competing players. MicroRTS focuses on the 1v1 setting; the underlying problem is therefore formally a *two-player zero-sum*[14] *simultaneous-move Markov game* [21], where players commit to actions at each tick without observing the opponent's choice.
In this thesis, the opponent's actions are absorbed into the transition function $\mathcal{T}$, reducing the problem to a single-agent perspective. For a fixed opponent policy, this induces a stationary single-agent MDP. However, when the opponent changes across training (e.g., curriculum learning or self-play), $\mathcal{T}$ shifts with it, creating non-stationarity. Additionally, a stochastic opponent policy makes $\mathcal{T}$ stochastic from the agent's perspective, even though the underlying engine is deterministic.

For generality, the formulation is first presented with fog-of-war enabled, in which case the problem is a *Partially Observable Markov Decision Process* (POMDP) [22, 23], defined by the tuple:

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma \rangle, \tag{2.5}$$

where:

- $\mathcal{S}$ is the state space, encompassing all game configurations;
- $\mathcal{A}$ is the agent's action space;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$[15] is the state transition function;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function;
- Ω is the observation space;
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ is the observation function;
- $\gamma \in [0, 1)$ is the discount factor.

The discount factor γ sets the agent's effective planning horizon: $\gamma = 0$ makes the agent myopic (only immediate rewards matter), while γ close to 1 lets rewards far into the future still meaningfully influence current decisions.

[12] C. Huang, MicroRTS-Py, 2019. GitHub: https://github.com/Farama-Foundation/MicroRTS-Py.
[13] JPype Project, Jpype – a Python to Java bridge, 2025. GitHub: https://github.com/jpype-project/jpype.
[14] A two-player game is *zero-sum* when one player's gain is exactly the other's loss. In MicroRTS, this holds for the terminal win/loss reward.
[15] $\Delta(\mathcal{X})$ is the set of probability distributions over $\mathcal{X}$.

Since all experiments in this thesis use full observability, $\mathcal{O}$ is the identity mapping and $\Omega = \mathcal{S}$, so the observation components of 2.5 drop out. The POMDP reduces to a *Markov Decision Process* [24], defined by the simplified tuple:

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle. \tag{2.6}$$

The MDP formulation rests on the *Markov property*, which also underlies the *Markov game* and the *POMDP*'s latent state: the next state depends only on the current state and action, not on the history. MicroRTS satisfies this at the level of the full engine state; whether a given *observation* encoding preserves it is a separate question, revisited in Chapter 7 when discussing temporal memory.

2.3.2 MicroRTS as an MDP

Each component of the MDP tuple is now instantiated for MicroRTS, using the formal game state notation introduced in Section 2.2.2. The concrete implementation choices (observation encoding, action space factorization, and reward components) are detailed in Chapter 7.

State Space

Each state $\boldsymbol{s}_t \in \mathcal{S}$ corresponds to the tuple of 2.3: clock, per-player resources, and the entity set with attributes. For a fixed map, $\mathcal{S}$ is finite but combinatorially large: even a coarse upper bound on a 16×16 map exceeds 10^{300} distinct configurations[16], far beyond explicit enumeration. Most of the algorithms used in this thesis therefore operate on compact representations of the state.

Action Space

At each decision step, the agent issues commands to its controllable idle entities, while non-idle entities complete their pending actions; the opponent does likewise, in the Markov-game framing of Section 2.3.1. Let n_t^{own} denote the number of agent-owned entities at time t[17]. The agent's joint action space is the product over its entities:

$$\mathcal{A} = \prod_{i=1}^{n_t^{\text{own}}} \mathcal{A}_i, \tag{2.7}$$

where each individual action $\boldsymbol{a}_i \in \mathcal{A}_i$ is a tuple as defined in Section 2.2.2. Since n_t^{own} varies over time and each $\mathcal{A}_i$ depends on the local context (type, occupied neighbors, available resources), the joint action space is both combinatorial and dynamic.

All entity types share the same action API, so $|\mathcal{A}_i|$ in Table 2.2 reports the API-level syntactic cardinality, not the runtime-feasible one. The *Attack* action notably exposes a fixed 7×7 target offset window (49 cells) regardless of unit type, even though most offsets are physically unreachable for a given unit (e.g., melee units have range = 1, leaving only the four cardinal neighbors valid). As a worked example, the Worker's $|\mathcal{A}_i| = 70$ decomposes as 1 (idle) + 4 (move) + 4 (harvest) + 4 (return) + 8 (produce: 4 directions × {Base, Barracks}) + 49 (attack); the same accounting yields $1+4+49 = 54$ for Light, Heavy, and Ranged, $1+4 = 5$ for a Base (producing only Workers), and $1+12 = 13$ for a Barracks (producing Light, Heavy, or Ranged). At runtime, an invalid-action mask [25] filters out two distinct sources of infeasibility: *static* type-level constraints (e.g., Light cannot Harvest, melee units cannot Attack at offset (3,3)) and *dynamic* state-dependent constraints (e.g., target out of range, no enemy at offset, insufficient resources for production). The masking implementation is detailed in Chapter 7.

[16] Each of the 256 cells of a 16×16 map admits 15 occupancy states (empty, obstacle, resource, or one of 6 player-owned entity types × 2 owners), yielding $\sim 15^{256} \approx 10^{301}$ raw configurations before per-entity attributes (HP, timers, carried resources). This is an upper bound on syntactic configurations; game constraints (limited resources, production costs) make most unreachable in practice.

[17] The case $n_t^{\text{own}} = 0$ corresponds to the agent having lost all its entities; this is a terminal state of the MDP and requires no action.

Transition Dynamics

The MicroRTS engine transitions deterministically: given state $\boldsymbol{s}_t$ and the joint action of both players $\boldsymbol{a}_t = (\boldsymbol{a}_t^1, \boldsymbol{a}_t^2)$, the next state $\boldsymbol{s}_{t+1}$ is computed in three stages. First, the clock advances ($\tau_{t+1} = \tau_t + 1$, with a draw if $\tau_{t+1} = T_{\max}$) and each entity's timer_i^t decrements by one; when a timer reaches zero, the corresponding action resolves (positions update on *Move*, hit points decrease on *Attack*, resources transfer on *Harvest*/*Return*). Second, each player k's resource count adjusts by $\Delta\rho_t^k$ (deposits minus production costs). Finally, $\mathcal{U}_{t+1}$ is obtained from $\mathcal{U}_t$ by removing entities with $\text{hp}_i^t = 0$ or depleted Resource nodes, and adding newly produced entities.

Reward Function

The default reward in MicroRTS is sparse and terminal:

$$\mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = \begin{cases} +1 & \text{Win: agent-owned entities remain in } \mathcal{U}_{t+1} \text{ and opponent-owned do not,} \\ -1 & \text{Loss: opponent-owned entities remain in } \mathcal{U}_{t+1} \text{ and agent-owned do not,} \\ 0 & \text{Draw: no player-owned entities remain in } \mathcal{U}_{t+1}, \text{ or } \tau_{t+1} = T_{\max}, \\ 0 & \text{otherwise (game ongoing).} \end{cases} \tag{2.8}$$

While sufficient for evaluation, this signal is non-zero only at terminal states and provides no intermediate feedback during training. It is therefore replaced by a dense scalar reward [11] computed as a weighted sum of K event-based components:

$$r_{t+1} = \boldsymbol{w}^\top \boldsymbol{r}_{t+1} = \sum_{k=1}^{K} w_k \, r_{t+1,k}, \tag{2.9}$$

where $\boldsymbol{r}_{t+1} \in \mathbb{R}^K$ captures game events from the transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ (resources gathered, units produced, damage dealt), and $\boldsymbol{w} \in \mathbb{R}^K$ controls their relative importance. The index $t+1$ marks the step at which the reward is received (after action $\boldsymbol{a}_t$); r_{t+1} is the indexed form of $\mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$.

Reinforcement Learning Foundations

3

Reinforcement learning is the branch of machine learning concerned with sequential decision-making. Historically, RL emerged from the confluence of two research threads [23]: *trial-and-error learning*, in which actions followed by favorable outcomes are reinforced; and *dynamic programming*, which solves sequential decision problems through value functions and the Bellman equations [26]. Modern RL combines both: an agent learns directly from interaction with an environment, guided by value functions, to maximize a cumulative reward signal. As illustrated in Figure 3.1, at each step t the agent observes state S_t and reward R_t from the previous transition, selects action A_t, and the environment transitions to S_{t+1}, returning R_{t+1}. Throughout this chapter, capital letters S_t, A_t, R_t denote random variables and lowercase s, a, r their realizations[1].

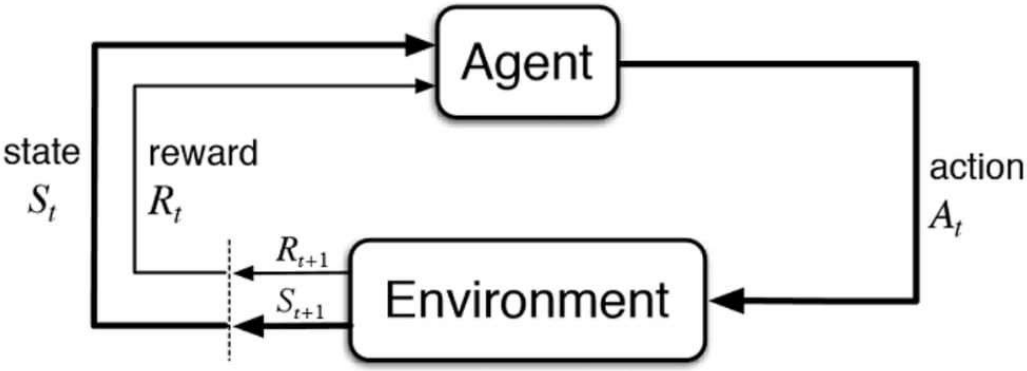


Figure 3.1: The agent-environment interaction loop in a Markov Decision Process [23].

This chapter introduces the core RL concepts used throughout this thesis, building on the MDP notation in Chapter 2. It progresses from value functions and policies to policy-gradient methods and PPO, the algorithm underlying all RL agents in this thesis. Section 3.5 concludes with a visual overview of the full PPO training loop instantiated for MicroRTS.

3.1 Policies and Value Functions

In RL, an agent selects actions according to a *policy* $\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s)$, which may be deterministic or stochastic. The outcome from timestep t is captured by the *discounted return*:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}. \tag{3.1}$$

To quantify the quality of a policy, two *value functions* are defined. The *state-value function* measures the expected discounted return when starting from state s and following π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s], \tag{3.2}$$

[1]Inside expectations, lowercase symbols are used as dummy integration variables, following standard convention in the RL literature.

while the *action-value function* additionally conditions on the first action taken:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t \mid S_t = s,\ A_t = a]. \tag{3.3}$$

Both satisfy the *Bellman expectation equations* [26], which follow from the recursive structure of G_t:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')\right], \tag{3.4}$$

$$Q^{\pi}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{R}(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^{\pi}(s',a')\right]. \tag{3.5}$$

The *optimal policy* π^* is the one that achieves the highest state-value in every state simultaneously:

$$V^{\pi^*}(s) \geq V^{\pi}(s), \quad \forall \pi,\ \forall s \in \mathcal{S}. \tag{3.6}$$

Bellman's optimality theorem [26] guarantees the existence of such a policy in finite MDPs.

3.2 Core RL Paradigms

3.2.1 Model-Free vs Model-Based

Model-free methods learn a policy or value function directly from interaction with the environment, treating it as a black box. Model-based methods instead build or use an explicit model of the transition dynamics $\mathcal{T}(s,a,s')$ and reward function $\mathcal{R}(s,a,s')$, which they use to plan ahead. While more sample-efficient in principle, model-based approaches require an accurate model of the environment: a condition difficult to meet in RTS games, where the action space is combinatorially large and transitions depend on the opponent's unknown actions. Model-free methods, especially policy-gradient methods, are therefore common in modern DRL approaches to RTS games.

3.2.2 On-Policy vs Off-Policy

On-policy methods optimize the policy using data collected by the current policy itself; after each update, previous data becomes stale. Off-policy methods can reuse data from past policies through experience replay buffers, improving sample efficiency at the cost of distribution mismatch between the *behavior policy* (which generated the data) and the *target policy* (being optimized) [23]. In RTS settings, on-policy methods are typically preferred in practice: the non-stationarity introduced by opponent adaptation and curriculum learning causes replay data to become rapidly outdated, undermining the key advantage of off-policy approaches. AlphaStar [8] stands as a notable exception, discussed in detail in Section 5.2.

3.3 Value-Based vs Policy-Based Methods

RL algorithms are commonly divided into two families based on what they optimize: *value-based* methods and *policy-based* methods.

3.3.1 Value-Based Methods

Value-based methods learn an approximation $\hat{Q}_{\theta}(s,a) \approx Q^*(s,a)$ of the *optimal action-value function* $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$, and derive a deterministic policy by greedy selection [27]:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} \hat{Q}_{\theta}(s,a). \tag{3.7}$$

As demonstrated by Deep Q-Networks (DQN) [28] on Atari games, these methods are effective for discrete action spaces of moderate size. However, the arg max becomes intractable when the action space grows combinatorially. In MicroRTS, even under a simplifying assumption of 5 legal actions per unit, 15 units yield $5^{15} \approx 3 \times 10^{10}$ joint actions per timestep.

3.3.2 Policy-Based Methods

Policy-based methods directly parameterize a stochastic policy $\pi_\theta(a \mid s)$ and optimize its parameters to maximize the expected discounted return:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_\theta}[G_0], \tag{3.8}$$

where G_0 is the discounted return of Equation 3.1 evaluated at $t = 0$ along a trajectory τ[2] of length T. By sampling rather than enumerating actions, these methods scale to larger action spaces.

3.4 From Policy Gradients to PPO

3.4.1 Policy Gradients

The *policy gradient theorem* [29] provides a way to estimate $\nabla_\theta J(\boldsymbol{\theta})$ (Equation 3.8) from sampled trajectories alone, without ever computing or differentiating through the environment dynamics:

$$\nabla_\theta J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right]. \tag{3.9}$$

Intuitively, high-return actions are reinforced and become more likely. The full derivation, including the log-derivative trick and the variance-reducing causality refinement, is detailed in Appendix A. The *REINFORCE* algorithm [30] is a direct application of this principle, but suffers from high variance in practice: each G_t aggregates the noise of many stochastic actions, producing unstable gradient estimates. This motivates the use of a learned baseline, presented in the next subsection.

3.4.2 Actor-Critic and Advantage Estimation

Actor-critic methods [31] reduce this variance by introducing a *critic* that learns a value function alongside the policy (*actor*). In practice, actor and critic share a common encoder f_θ with only the final layers differing [32]. The critic provides the actor with an *advantage* signal:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{3.10}$$

Intuitively, $A^\pi(s, a)$ measures whether action a was better or worse *than expected* in state s. Using $A^\pi(s, a)$ rather than G_t centers the gradient signal around zero, so only above-average actions are reinforced while underperforming ones are discouraged. By Bellman's equation, this is equivalent to the expected temporal-difference (TD) residual:

$$A^\pi(s, a) = \mathbb{E}_{s'}[\delta^\pi], \quad \delta^\pi \equiv \mathcal{R}(s, a, s') + \gamma V^\pi(s') - V^\pi(s). \tag{3.11}$$

Equation 3.11 cannot be evaluated directly: the true value function V^π is unknown. The critic only provides a learned approximation $V_\theta \approx V^\pi$. Substituting this approximation into the TD residual yields a one-step estimator:

$$\hat{A}_t^{(1)} = \delta_t = \mathcal{R}(s_t, a_t, s_{t+1}) + \gamma V_\theta(s_{t+1}) - V_\theta(s_t), \tag{3.12}$$

[2] An *episode* (or *trajectory*) is a sequence $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$ of state-action-reward triples generated by the agent interacting with the environment until termination at step T.

which has low variance (a single reward and bootstrap) but is biased by the imperfections of V_θ. At the opposite extreme, the Monte Carlo advantage $G_t - V_\theta(s_t)$ uses the entire trajectory: unbiased, but with high variance from cumulative reward noise. Intermediate k-step estimators interpolate between these two regimes, but committing to a fixed k is rigid.
Generalized Advantage Estimation (GAE) [33] sidesteps this choice by taking an exponentially weighted moving average (EMA) of all multi-step estimators with decay $\lambda \in [0,1]$:

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \delta_{t+k}. \tag{3.13}$$

The geometric factor $(\gamma\lambda)^k$ down-weights TD residuals from the distant future. The parameter λ controls the steepness of this decay: $\lambda = 0$ reduces to the one-step estimator δ_t; $\lambda = 1$ recovers the Monte Carlo advantage. Typical RL implementations use $\lambda \approx 0.95$, favoring observed rewards while retaining mild bootstrapping for variance reduction.

3.4.3 Proximal Policy Optimization

Combining the policy gradient theorem (Equation 3.9) with the GAE advantage estimator (Equation 3.13) yields the actor-critic update rule:

$$\nabla_\theta J(\theta) = \mathbb{E}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot \hat{A}_t\right], \quad \hat{A}_t \equiv \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}. \tag{3.14}$$

Applied as a single gradient step, this is *on-policy*: after each update the data becomes stale and must be discarded, which is sample-inefficient.
PPO [34] allows multiple gradient steps per batch by reformulating Equation 3.14 as the gradient of an *importance-sampled (IS) surrogate loss*:

$$\mathcal{L}^{\mathrm{IS}}(\theta) = \mathbb{E}_t\left[\rho_t(\theta)\hat{A}_t\right], \quad \rho_t(\theta) = \pi_\theta(a_t \mid s_t) / \pi_{\theta_{\mathrm{old}}}(a_t \mid s_t). \tag{3.15}$$

The probability ratio ρ_t equals 1 at $\theta = \theta_{\mathrm{old}}$, where $\nabla\mathcal{L}^{\mathrm{IS}}$ recovers the on-policy form of Equation 3.14 (full derivation in Appendix A). This identity is exact in expectation, but the importance-sampled estimator becomes statistically unstable far from θ_{old}: if multiple gradient steps push θ away, ρ_t can explode in magnitude and the estimate's variance diverges, making the policy update catastrophically unstable[3]. PPO solves this with a *clipped surrogate objective* that bounds the contribution of each sample once ρ_t leaves a small trust region $[1-\epsilon, 1+\epsilon]$:

$$\mathcal{L}^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\rho_t(\theta)\hat{A}_t,\ \mathrm{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{3.16}$$

where $\mathrm{clip}(x,a,b) = \max(a, \min(b,x))$ and $\epsilon \in (0,1)$ is a hyperparameter (typically 0.1 to 0.2).
The min/clip mechanism enforces an asymmetric, conservative update on the policy loss: the policy-gradient contribution of a sample is suppressed only when ρ_t has already moved beyond the trust region in the direction favored by $\hat{A}_t$ (i.e., $\rho_t > 1+\epsilon$ with $\hat{A}_t > 0$, or $\rho_t < 1-\epsilon$ with $\hat{A}_t < 0$), removing the incentive to push further. When ρ_t has moved in the opposite direction, the unclipped gradient still flows and pulls the policy back toward the trust region. This empirically limits policy drift across multiple gradient steps on the same batch, although the actual drift still depends on the optimizer, learning rate, and number of epochs, and the clip imposes no hard constraint on the distance between θ and θ_{old}. This practical stability is why PPO has become the default algorithm for RTS agents [11, 12].

[3] Trust Region Policy Optimization (TRPO) [35], PPO's predecessor, addresses this by enforcing a hard Kullback-Leibler (KL) divergence constraint between the new and old policies. The constraint requires expensive second-order optimization, which PPO avoids by using the clipping mechanism.

The full PPO objective to maximize combines three terms:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}^{\text{CLIP}}(\boldsymbol{\theta}) - c_v \mathcal{L}^{\text{VF}}(\boldsymbol{\theta}) + \beta H[\pi_{\boldsymbol{\theta}}]. \tag{3.17}$$

The *value function loss* (VF) fits the critic to a bootstrapped[4] target consistent with the GAE estimator:

$$V_t^{\text{target}} = V_{\boldsymbol{\theta}_{\text{old}}}(s_t) + \hat{A}_t^{\text{GAE}}. \tag{3.18}$$

Following standard PPO implementations [11], the loss is itself clipped to prevent large value updates across the K gradient steps performed on each batch:

$$\mathcal{L}^{\text{VF}}(\boldsymbol{\theta}) = \mathbb{E}_t\left[\max\left(\left(V_{\boldsymbol{\theta}}(s_t) - V_t^{\text{target}}\right)^2, \left(\widetilde{V}_{\boldsymbol{\theta}}(s_t) - V_t^{\text{target}}\right)^2\right)\right], \tag{3.19}$$

where $\widetilde{V}_{\boldsymbol{\theta}}(s_t) = V_{\boldsymbol{\theta}_{\text{old}}}(s_t) + \text{clip}\left(V_{\boldsymbol{\theta}}(s_t) - V_{\boldsymbol{\theta}_{\text{old}}}(s_t), -\epsilon, +\epsilon\right)$ is the clipped value prediction (using the same ϵ as the policy clip in Equation 3.16). The max enforces a pessimistic loss: whichever of the two squared errors is larger is the one used, mirroring the min pessimism of the policy clip and stabilizing the critic across multiple gradient steps.
The *entropy bonus* rewards stochastic policies, encouraging exploration:

$$H[\pi_{\boldsymbol{\theta}}] = -\mathbb{E}_t\left[\sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s_t) \log \pi_{\boldsymbol{\theta}}(a \mid s_t)\right]. \tag{3.20}$$

The hyperparameter c_v weights the critic update relative to the policy update; values $c_v \in [0.5, 1.0]$ are standard, balancing the two heads of the shared encoder. The hyperparameter β controls the strength of entropy regularization: $\beta > 0$ rewards stochastic policies and prevents premature policy collapse to a single action, while $\beta = 0$ removes entropy regularization entirely, relying solely on the stochasticity of $\pi_{\boldsymbol{\theta}}$ for exploration ($\beta \approx 0.01$ is typical). The exact values used in this thesis are reported in Chapter 10.

3.5 The PPO Training Loop

This section assembles the components introduced above into a complete training loop, first at the algorithmic level (Section 3.5.1), then instantiated for MicroRTS (Section 3.5.2).

3.5.1 The PPO Algorithm

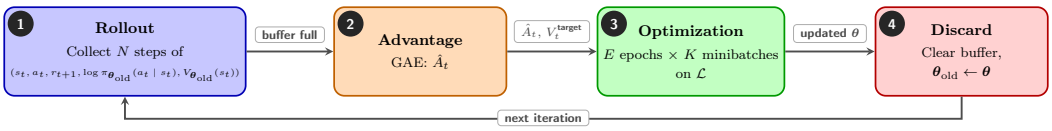


Figure 3.2: PPO training cycle: rollout (1), GAE (2), clipped optimization (3), and buffer discard (4).

Figure 3.2 illustrates the PPO training cycle, which proceeds in four steps at each iteration.

Step 1: Rollout Collection. The agent collects N environment steps with $\pi_{\boldsymbol{\theta}_{\text{old}}}$ and stores them as tuples $(s_t, a_t, r_{t+1}, \log \pi_{\boldsymbol{\theta}_{\text{old}}}(a_t \mid s_t), V_{\boldsymbol{\theta}_{\text{old}}}(s_t))$ in a buffer.

[4] A target is *bootstrapped* when its construction uses the value function's own estimates rather than only observed rewards; the GAE-based target above does so through the $V(s_{t+k})$ embedded in each TD residual δ_{t+k}.

Step 2: Advantage Computation. GAE walks backward through the buffer to compute a fixed advantage $\hat{A}_t$ for each step (Equation 3.13).

Step 3: Optimization. The buffer is shuffled into K minibatches and reused for E epochs. At each minibatch, the network recomputes the new $\pi_\theta(a_t \mid s_t)$ against the stored $\pi_{\theta_{\text{old}}}$ to form ρ_t, then takes a gradient step on $\mathcal{L}$ (Equation 3.17). As θ evolves across these $K{\times}E$ updates, ρ_t drifts from 1; the clipping bounds this drift and prevents the policy from straying too far from the collected data, allowing multiple passes while remaining approximately on-policy.

Step 4: Cycle Restart. The buffer is discarded and $\theta_{\text{old}} \leftarrow \theta$ before restarting the cycle.

3.5.2 PPO for MicroRTS

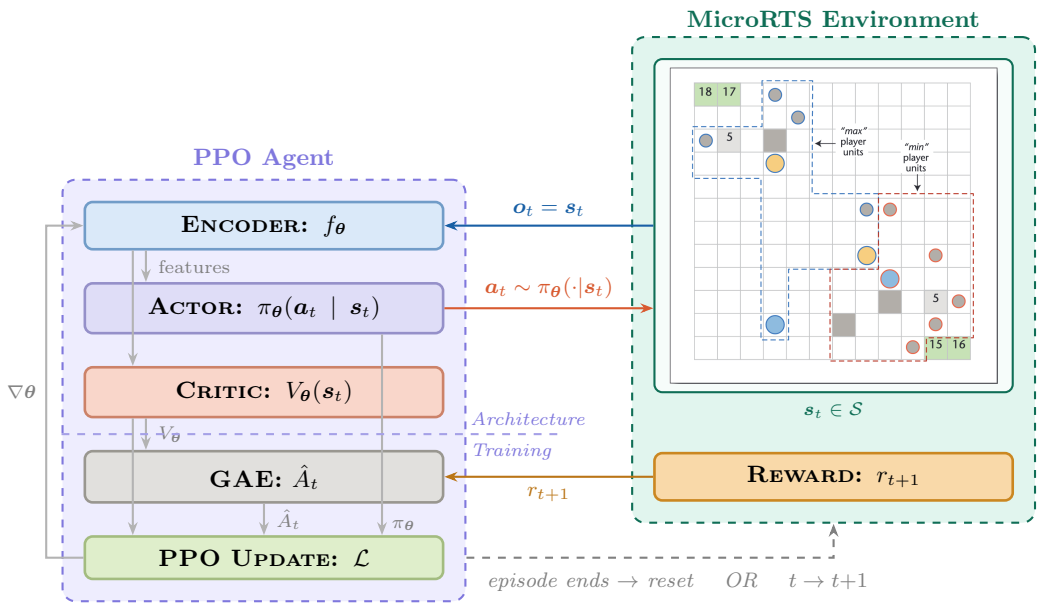


Figure 3.3: PPO training loop applied to MicroRTS. At each timestep, the agent receives an observation and produces an action; the environment returns a reward and advances the game state.

Figure 3.3 shows the PPO loop instantiated for MicroRTS. The shared encoder f_θ processes the game state s_t and feeds both the actor and critic. The actor samples an action a_t, which is applied to the environment to produce the next state s_{t+1} and reward r_{t+1}. GAE combines the critic's estimate V_θ with the reward to compute advantages $\hat{A}_t$, which drive the PPO update of the shared parameters θ via the objective $\mathcal{L}$. The concrete design choices fill the remainder of this thesis: observation encoding, action representation, and reward signals are covered in Chapter 7; neural network architectures in Chapter 8; and training enhancements in Chapter 9.

Classical Approaches in RTS Games

4

This chapter reviews three classical paradigms for RTS AI: rule-based, search-based, and hierarchical. Each is illustrated with agents from the MicroRTS competition, drawn from the evaluation benchmark assembled in Chapter 6. Figure 4.1 arranges them in the chronological order of their introduction in game AI: each emerged to address limitations of existing approaches by integrating new mechanisms [36, 37]. Newer paradigms, however, have not displaced older ones; all three remain in active use today and are routinely combined within single systems [38], as exemplified by AlphaZero, which combines DRL with search [39]. Whether used in isolation or in combination, their shared reliance on hand-designed control motivates the transition to DRL (Chapter 5).

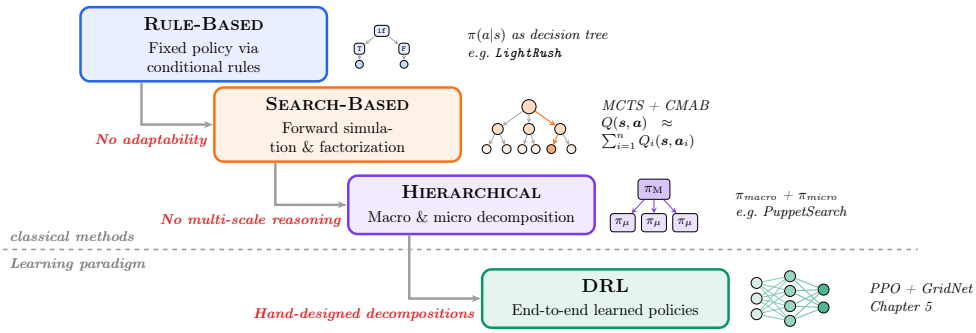


Figure 4.1: AI paradigms for RTS games in chronological order of introduction, from fixed conditional rules to end-to-end learned policies. Each addresses limitations of earlier ones without displacement.

4.1 Rule-Based Approaches

A common classical approach to RTS game AI is rule-based control, which encodes a fixed policy $\pi(a|s)$ through manually designed conditional rules [40]. Each unit's action is selected by traversing a priority-ordered decision tree over the game state, without learning or search.

Listing 4.1: Pseudocode of a simplified `LightRush` policy in MicroRTS.

```
for unit in my_units:
    if is_type(unit, Base):
        if count(Worker) < 2 and resources >= 1:
            train(Worker)
        elif not exists(Barracks) and resources >= 5:
            build(Barracks, adjacent_pos)
    elif is_type(unit, Barracks):
        if resources >= 2:
            train(Light)
    elif (is_type(unit, Light) or is_type(unit, Worker)) and is_idle(unit):
        enemy = nearest_enemy(unit)
        if enemy and in_attack_range(unit, enemy):
            attack(unit, enemy)
        else:
            move_toward(unit, enemy_base)
```

Listing 4.1 illustrates this structure with a simplified `LightRush` strategy. The policy is entirely defined by three conditional blocks: the base produces up to two Workers before constructing a

Barracks; the Barracks then trains Light units; and any idle unit (Workers included) either attacks an enemy in range or advances toward the opponent's base.
Despite this simplicity, rule-based agents remain competitive in MicroRTS competitions: hand-tuned heuristics are highly effective in anticipated scenarios, while full interpretability and low computational cost make them attractive baselines.
Strong examples from recent editions include the following: CoacAI (2020) uses heuristic economy management with A^* pathfinding; Mayari (2021) performs greedy per-action heuristic scoring with local distance estimation; ObiBotKenobi (2023) combines unit-role assignment with Dijkstra-based pathfinding; TMA (2024) selects between four strategy combinations (attack/defense × worker/military), recalibrated every 500 game cycles.
However, rule-based policies adapt only within the scenarios anticipated by their hand-designed rules, with no mechanism to improve from experience. This motivated search-based approaches that adapt behavior to novel states through forward simulation.

4.2 Search-Based Approaches

Search-based approaches derive actions through explicit forward simulation of future consequences. Classical methods such as Minimax with Alpha-Beta pruning [41, 42] defined the early state of the art in turn-based games, culminating in IBM Deep Blue's 1997 defeat of world chess champion Garry Kasparov [43]. A real-time variant, Iterative Deepening Real-Time Minimax (IDRTMinimax), adapts this to RTS games by searching as deep as possible within a fixed time budget [44]. However, feasibility depends critically on the *joint* branching factor b (legal actions per state, taken jointly over all controlled units): for a search depth d, the number of nodes grows as $O(b^d)$. Figure 4.2 compares branching factors across board and RTS games.

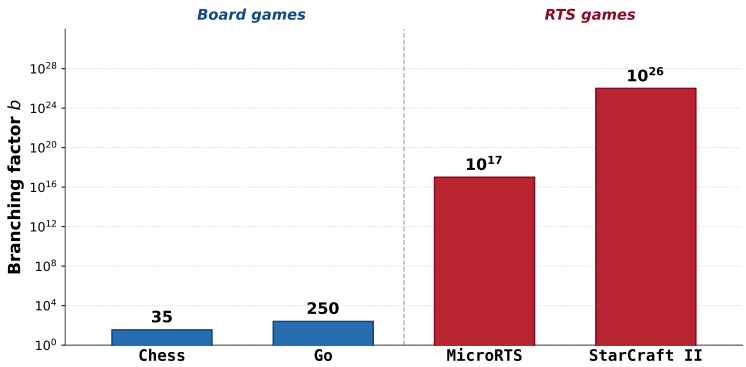


Figure 4.2: Branching factor estimates across board and RTS games. Board game values from [45]; MicroRTS value is the average on a 12×12 map with 4 starting bases [46]; StarCraft II from [8]. RTS values should be interpreted as order-of-magnitude estimates, since the branching factor depends strongly on the state, map, unit count, and action abstraction used.

Branching factors for RTS games exceed even Go's by several orders of magnitude. Despite aggressive pruning, they render exhaustive search infeasible under real-time constraints [46].

4.2.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) replaces exhaustive enumeration with stochastic sampling [47–49]. It estimates action values $Q(s, a)$ by averaging returns over rollouts. A key property is its *anytime*

behavior: it can return a decision at any point, with quality improving as more simulations complete. MCTS is best known as the core algorithm of AlphaGo [45], discussed in Chapter 5.
However, standard MCTS remains insufficient for RTS environments: the exponential joint action space means most actions are never sampled, and the sparse terminal rewards typical of RTS games provide minimal guidance during rollouts. Droplet (2019) targets the action-space problem through script-biased sampling that steers MCTS toward promising action sequences, but the combinatorial explosion ultimately remains. This motivates approaches that explicitly factorize the action space.

4.2.2 Combinatorial Multi-Armed Bandits

Combinatorial Multi-Armed Bandits (CMAB) [46] address the combinatorial explosion of the action space through factorization, approximating the joint action-value function as:

$$Q(\boldsymbol{s}, \boldsymbol{a}) \approx \sum_{i=1}^{n} Q_i(\boldsymbol{s}, \boldsymbol{a}_i), \tag{4.1}$$

where $Q_i(\boldsymbol{s}, \boldsymbol{a}_i)$ estimates the contribution of unit i's action to the global reward. This relies on an *independence assumption*: actions are assumed to have limited cross-dependencies (e.g., units on opposite sides of the map act independently). Under this independence, the number of values to estimate drops from $O(k^n)$ joint-action values to $O(nk)$ per-unit action values, where k counts actions per unit, enabling tractable planning even with dozens of units (Figure 4.3).

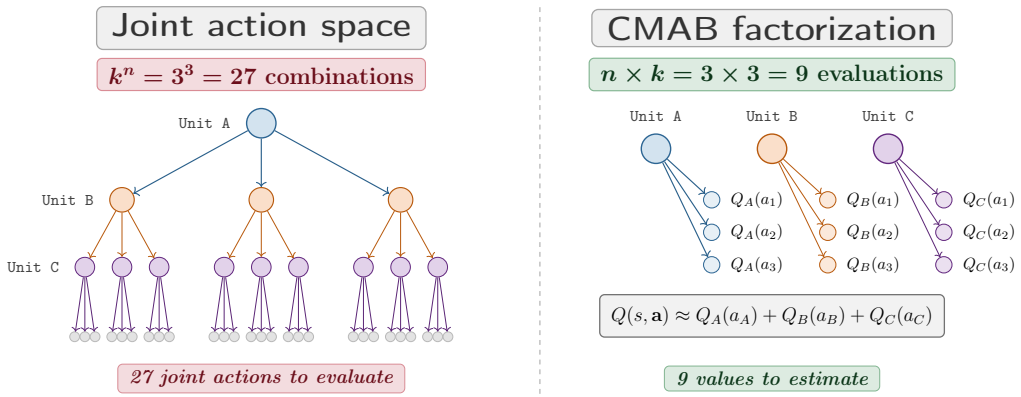


Figure 4.3: Joint action space vs CMAB factorization (3 units, 3 actions each). Left: k^n joint evaluations. Right: nk per-unit evaluations under the independence assumption (Equation 4.1).

NaïveMCTS [46] combines CMAB with MCTS by estimating per-unit Q_i values during the tree search. The Informed variant [50] additionally guides rollouts using handcrafted strategies such as `LightRush` (Listing 4.1). Further gains were achieved by grouping unit commands into higher-level action abstractions [51, 52].
Despite substantial gains over standard MCTS, CMAB remains constrained by limited search horizons and the independence assumption, which breaks down when units must coordinate closely. This factorization principle reappears in Chapter 5, where GridNet (a spatial neural network architecture) applies the same independence assumption to neural network outputs, producing per-cell action distributions $\pi_\theta(\boldsymbol{a}_i|\boldsymbol{s})$ rather than estimating Q_i through search.
Beyond the independence assumption, CMAB treats all decisions at a single temporal scale, conflating strategy with tactics. Hierarchical approaches address this temporal limitation.

4.3 Hierarchical Approaches

As introduced in Section 2.1, RTS games require decisions at two scales: macro-management (economy, production) and micro-management (unit control, combat). Rule-based and search-based methods treat both scales uniformly. Hierarchical approaches instead decompose the policy $\pi(a|s)$ into specialized components for each level, as illustrated in Figure 4.4. Table 4.1 lists representative techniques from the literature at each scale.

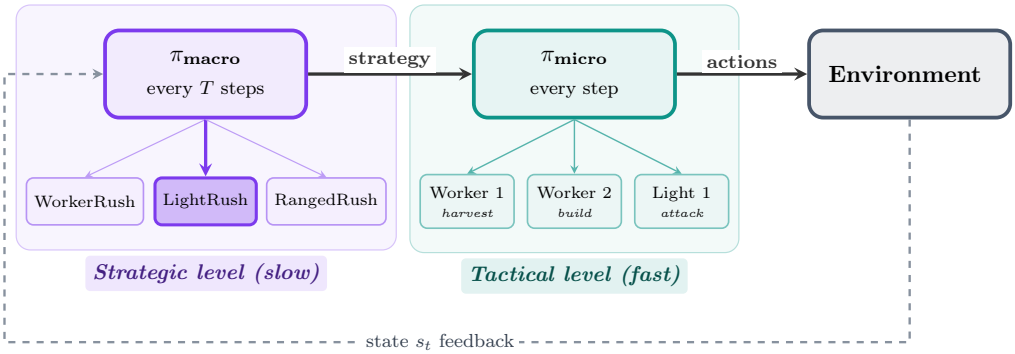


Figure 4.4: Hierarchical decomposition illustrated with *StrategyTactics* (2017) [53, 54]: π_{macro} periodically selects a strategy; π_{micro} translates it into per-unit actions every step.

Table 4.1: Representative techniques used at each decision scale in classical RTS AI.

Technique	**Description**
Macro-management	
Hierarchical Task Networks	Decompose strategic goals into ordered subtasks [55]
Bayesian Opponent Modeling	Infer opponent strategy to adapt plans [56]
Goal-Driven Autonomy	Detect expectation failures and reformulate goals [57]
Micro-management	
Potential Fields	Compute attractive and repulsive forces for unit positioning [58]
Behavior Trees	Provide modular structures for reactive unit control [59]
Neuroevolution	Evolve neural network controllers for micromanagement [60]

MicroRTS competition agents following similar hierarchical designs include Tiamat (2018), which combines evolved action abstractions with online strategy-tactics search; MixedBot (2019), which extends Tiamat's strategic search with CMAB-based tactical execution and dynamic time allocation; and UtsImass (2019), which tunes build parameters [61] via pre-game self-play before switching to rule-based control with Jump Point Search (JPS) [62], an optimized A* variant for grid pathfinding.

From Classical Methods to DRL. All paradigms reviewed in this chapter share a common limitation: their effectiveness rests on hand-designed components that the designer must specify in advance [37, 63]. Figure 4.1 positions DRL as a different paradigm: by learning parts of the policy and representation from experience, DRL reduces the need for hand-designed decision rules, although environment design, action abstractions, rewards, and architectures remain important engineering choices. Design effort thus shifts from encoding behavior directly to specifying the conditions under which behavior can be learned.

Deep Reinforcement Learning Approaches in RTS Games

5

Deep reinforcement learning offers an end-to-end alternative to the classical methods reviewed in Chapter 4. This chapter first reviews some key milestones that brought DRL from board games to real-time strategy, then analyzes AlphaStar [8] in detail as the most influential large-scale reference for this work. MicroRTS-specific work follows, focusing on the Gym-μRTS baseline [11] that this thesis builds upon and RAISocketAI [12] (IEEE CoG 2023 winner), the only DRL agent to have won the MicroRTS competition. A closing section positions the contributions of this thesis.

5.1 Major DRL Breakthroughs

This section reviews some key milestones that established DRL for strategic games [64], illustrated in Figure 5.1.

Figure 5.1: Selected major DRL milestones in strategic games (2013–2020); not exhaustive.

5.1.1 Foundational Breakthroughs in Turn-Based Games

DQN (2013) proved that convolutional neural networks (CNN) could learn control policies from raw pixels on Atari games [28]. AlphaGo [45] (2016) used supervised learning (SL) on human games to initialize a policy network, refined it through RL self-play, and combined it with MCTS to defeat world champion Lee Sedol in Go. AlphaZero [39] (2017) generalized this to Go, Chess, and Shogi through pure self-play without human knowledge. MuZero [65] (2020) extended this to the model-based setting by learning a latent dynamics model without access to game rules, matching AlphaZero's performance while also mastering Atari.
These systems marked major milestones for AI in strategic games. However, compared with RTS games, these environments remain simpler along several dimensions: board games are turn-based and fully observable, while Atari involves a single agent with a much smaller action space and no simultaneous multi-unit control.

5.1.2 Scaling DRL to RTS Games

The transition from turn-based environments to real-time multi-agent games came with AlphaStar [8] (2019), which reached Grandmaster level in StarCraft II by combining SL from human replays with RL through competitive league training. Concurrently, OpenAI Five [66] defeated Dota 2 world champions using PPO with pure self-play. Both systems required massive computational resources: OpenAI Five trained on 256 GPUs and 128 000 central processing unit (CPU) cores for 10 months.

Subsequent work focused on reducing this computational cost. StarCraft Commander (SCC) [67] matched StarCraft II performance with an order of magnitude less computation, while AlphaStar Unplugged [68] trained competitive agents entirely from offline replay data.

5.2 DRL at Full RTS Scale: AlphaStar

AlphaStar [8] reached StarCraft II Grandmaster level (top 0.15% of human players) in 2019, the first AI system to achieve professional-level play in a full-scale RTS environment. Training required 384 TPUs and 1 800 CPUs across the training league over 44 days, with each agent accumulating 200 years of simulated game time. This scale is not reproducible in an academic setting. Nevertheless, several of AlphaStar's design choices transfer to smaller environments and have directly influenced the present design (Chapters 8 and 9); Figure 5.2 gives a high-level overview.

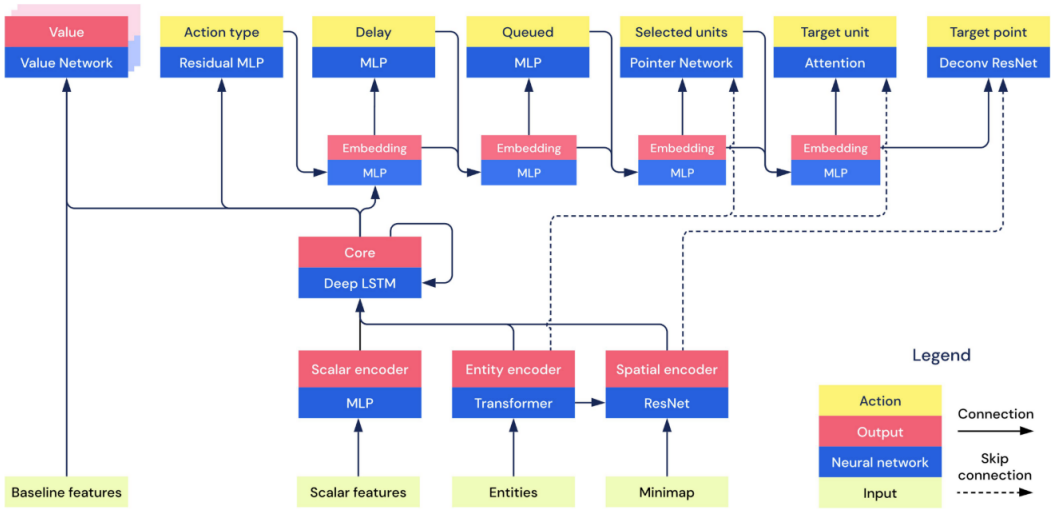


Figure 5.2: AlphaStar architecture from [8]. Three input streams (Scalar features, Entities, Minimap) are processed by a Multilayer Perceptron (MLP), a Transformer, and a Residual Network (ResNet) respectively; the Entity and Spatial encoders are linked through a scatter connection to combine spatial and non-spatial features. The encoded representations feed the core deep Long Short-Term Memory (LSTM) [69], whose output drives six auto-regressive action heads (Action type, Delay, Queued, Selected units, Target unit, Target point) connected through Embedding/MLP modules. Each head uses a specialized output mechanism (Residual MLP, MLP, MLP, Pointer Network [70], Attention, Deconvolutional ResNet). Dashed skip connections route encoder representations directly to the pointer, attention, and deconvolutional heads. A separate Value Network combines the LSTM output with privileged baseline features (training-only state information not accessible to the policy, e.g. the opponent's hidden units) to produce the critic estimate $\hat{V}(s_t)$.

5.2.1 Observation Encoding

Two of AlphaStar's encoding choices respond to fog-of-war: a deep LSTM that aggregates information across time, and privileged opponent features fed to the Value Network during training. Training in the fully observable MicroRTS setting removes one of the main motivations for recurrent memory and privileged critic inputs, although recurrence could still help capture temporal patterns such as opponent strategies or production histories. This thesis therefore opts for a simpler frame-stacking approach (Chapter 9), with the actor and critic sharing the same observation.

What does transfer is the entity-spatial representation pair linked through scatter connections: a Transformer over per-unit features fused with a CNN over the map. The same fusion, well-suited to the heterogeneous unit composition typical of RTS games, is adopted in Chapter 8.

5.2.2 Action Space

StarCraft II's $\sim 10^{26}$-action space forces AlphaStar to sample one action per step through six conditional heads, paying an auto-regressive cost at every decision. MicroRTS factorizes along a different axis: GridNet [11] predicts all per-cell actions in a single forward pass under conditional independence (Chapter 8), which is much cheaper but cannot model dependencies between simultaneous unit decisions. To recover some expressiveness within each unit's factored action, an optional auto-regressive sub-action head inspired by AlphaStar is implemented (Chapter 9).

5.2.3 Training Pipeline

AlphaStar's training has two phases: SL on ~970 000 human replays produces a Diamond-level initialization, then RL refines the policy through league self-play, using V-trace [71] for off-policy correction, upgoing policy update (UPGO) for self-imitation, and TD(λ) for value learning. The supervised warmstart is retained directly, with bot replays substituting for the unavailable human dataset (evaluated in Chapter 10). AlphaStar's *pseudo-rewards* steer the agent toward human strategies: at the start of each episode, a target statistic z (build order, unit composition) is sampled from a database of human replays, and the agent receives auxiliary rewards throughout the episode proportional to how closely its behavior matches z. This thesis instead uses *reward scheduling*, which progressively anneals dense hand-crafted shaping signals (resource gathering, unit production) toward the sparse terminal win/loss reward over training (Chapter 9). The two mechanisms address the same problem (sparse-reward learning) but in opposite directions: AlphaStar adds a dense signal to imitate known human strategies, whereas reward scheduling removes the dense signal so the agent eventually optimizes the true game outcome alone. The off-policy machinery (V-trace, UPGO) is also dropped, since vanilla PPO suffices at MicroRTS scale and avoids the engineering overhead.

5.2.4 League Training and PFSP

Naïve self-play suffers from *strategy cycling*: an agent learns strategy A that beats B, then B that beats A, forgetting earlier solutions in a non-transitive loop. AlphaStar's league of concurrent agents addresses this through three role types. *Main agents* train against the entire population via Prioritized Fictitious Self-Play (PFSP), sampling opponent i with probability proportional to $(1 - \mathrm{WR}_i)^p$ so harder opponents are seen more often[1]. Two auxiliary populations of *exploiters* probe weaknesses and are periodically reset to keep exploration fresh, while frozen checkpoints accumulate to prevent catastrophic forgetting of counter-strategies. This thesis implements the same PFSP weighting and checkpoint pool (Chapter 9); the exploiter populations are omitted as the training budget does not justify the parallel agents.

5.3 DRL at MicroRTS Scale

Attention now turns to MicroRTS, the environment at the center of this thesis. Despite being actively maintained since 2013 and hosting annual competitions since 2017, the DRL literature on MicroRTS remains sparse [72, 73]: only a handful of studies exist, most competition entries remain

[1]The exponent $p \geq 1$ controls how sharply sampling concentrates on the hardest opponents: at $p = 1$, weights are linear in $1 - \mathrm{WR}_i$, while larger p amplifies the gap. AlphaStar [8] uses $p = 2$ in its *hard* PFSP variant.

rule-based, search-based, or hybrid (Chapter 4), and RAISocketAI remains the only DRL agent to have won the competition. This scarcity is a primary motivation for the present thesis.

5.3.1 Early Exploration (2019–2020)

Two foundational studies established the basic learning paradigm. Huang et al. [74] compared global and local representations for both observations and actions: the global setting treats the game as a single decision-maker selecting one action per step, while the local setting decentralizes control by letting each unit independently select its action conditioned on local context. On resource gathering tasks, the local setting consistently outperformed the global one, motivating the per-cell action factorization that GridNet would later formalize (with a centralized observation and a single shared policy). Kelly and Churchill [75] reported that value-based methods (DQN variants [76, 77]) struggled with the combinatorial joint action space of RTS domains, while policy-gradient methods showed more promise. These studies established two design principles adopted by the field: spatial action factorization and policy-gradient training.

5.3.2 Gym-μRTS and GridNet (2020–2021)

The release of Gym-μRTS [11] provided Gymnasium-compatible interfaces with factorized action spaces, invalid action masking [25], and vectorized environments (Chapter 7). The paper also popularizes ***GridNet***, the standard architectural pattern for MicroRTS: a fully convolutional encoder-decoder predicting one action per grid cell in parallel under a conditional independence assumption. A PPO agent trained with GridNet for ~60 hours achieved ~89% WR on the standard *basesWorkers16x16A* map against the 2021-era scripted opponent pool, one of the first DRL systems to become competitive against classical MicroRTS bots. However, training was limited to a single map and the agent was only evaluated as Player 0 (P0) from a fixed starting position.

5.3.3 RAISocketAI (2023–2024)

RAISocketAI [12] became the first DRL agent to win the MicroRTS competition (IEEE CoG 2023), achieving ~72% WR across all matchups. The system uses a *DoubleCone* (4, 6, 4) architecture, adapted from the fourth-place Lux AI Season 2 model: 4 full-resolution residual blocks, an encoder-decoder block downsampling by stride 4 to process 6 residual blocks at quarter resolution before upsampling (with a residual skip across the down-up section), and 4 final full-resolution blocks, each block equipped with Squeeze-and-Excitation (SE) channel attention. Since DoubleCone exceeds the 100 ms per-turn budget on larger maps, RAISocketAI also includes *squnet*, a U-Net variant with more aggressive downscaling that requires up to 8× fewer operations on 64×64 maps.
PPO training, alternating between self-play and scripted opponents in a curriculum, annealing shaped rewards (resource gathering, unit production) toward sparse terminal ones during training. At inference, RAISocketAI deploys a *portfolio* of up to seven specialized models selected by map size, with map-specific fine-tuning on three maps that diverge most from the training distribution. The paper also reports a BC-PPO variant that combines behavior cloning (BC) pre-training on bot replays [78] with PPO fine-tuning, achieving ~88% in benchmark round-robin[2] without map-specific fine-tuning, suggesting that imitation warmstarts improve cross-map generalization. Total compute for the full portfolio reached ~1.5 billion environment steps and ~70 GPU-days, encompassing base self-play training, iterative fine-tuning against CoacAI/Mayari, transfer learning to map-specific models on the three larger Open maps, and the alternative squnet models for larger maps. Of this total, ~23.6 GPU-days were spent solely on the 16×16 DoubleCone model. The follow-up BC-PPO variant required an additional ~72 GPU-days (~23 for behavior cloning, ~49 for PPO fine-tuning).

[2]Single-player round-robin against WorkerRush, LightRush, CoacAI, and Mayari on the eight open maps (Chapter 6).

5.3.4 Recent Directions (2024–2025)

Reinforcement Learning as a Rehearsal (RLaR) [79] couples actor-critic RL with an auxiliary prediction function exploiting privileged information available at training to guide exploration under partial observability. Lemos et al. [80] address a generalization bottleneck of GridNet, namely the critic's reliance on a fixed-size flatten before its dense layers. They integrate a Spatial Pyramid Pooling (SPP) layer into the critic, producing a fixed-length representation regardless of map size and enabling a single agent to play maps of arbitrary dimensions. This thesis adopts both directions in its training system (Chapter 9): an auxiliary opponent-modeling head predicts the opponent's next action to bias the encoder toward strategically informative features, and an SPP-equipped critic decouples the network from a fixed map resolution.

5.4 Thesis Contributions

This thesis builds on Gym-μRTS [11] and draws design inspiration from RAISocketAI [12] and AlphaStar [8]. Six contributions address the gaps surveyed above (sparse DRL literature on MicroRTS, limited Gym-μRTS pipeline, no systematic ablations, 2026 CoG shift to LLM-based agents).

1. Reproducible MicroRTS tournament framework (Chapter 6). A self-contained tournament framework, anchored on the 2023 MicroRTS competition, runs on 12 maps (8 open + 4 closed), 15 benchmark agents spanning seven competition editions, and four game-theoretic metrics (Nash averaging, α-Rank, Copeland, worst-case and average regret).

2. Extended and modular Java-Python bridge (Chapter 7). The Gym-μRTS bridge is extended with self-play, multi-map padding, filtered action masking, symmetry augmentation, composable observation and action wrappers, and a reserved-observation channel, all opt-in and preserving numerical equivalence with the baseline.

3. UECD architecture and incremental design path (Chapters 8 and 10). UECD, one of the first MicroRTS architectures with explicit per-unit Transformer reasoning, integrates a CBAM-gated U-Net (local spatial), an Entity Transformer (unit-to-unit), and bottleneck self-attention (global spatial) into a single policy. Seven intermediate architectures from GridNet to UECD trace the design path, each ablated across three seeds.

4. Modular training pipeline with systematic ablations (Chapters 9 and 10). Flag-gated PPO mechanisms span five axes (action sampling, reward and value signal, opponent curriculum and self-play, auxiliary representation, generalization and architecture). Each is ablated across three seeds, driving the final recipe.

5. An analysis of discount-induced reward collapse (Chapter 10). A formal derivation explains a training instability arising when the shaped reward is annealed to zero: once only the terminal reward survives, the empirical PPO advantage favors shorter episodes by $\gamma^{-\Delta}$, driving the policy into a degenerate rush that wins against the training pool but collapses against held-out opponents. The analysis isolates the three causal conditions and the mitigations that break them, a result transferable to any shaped-to-sparse reward annealing schedule.

6. Compute-efficient single-map champion (Chapter 10). UECD-`Best`, combining UECD with the top features and a two-phase opponent-pool fine-tuning schedule, reaches a **96.67%** overall win rate on the standard *basesWorkers16x16A* map in a 19-agent tournament, ranking first on three of four game-theoretic metrics. Against RAISocketAI, it holds a **65.7%** head-to-head win rate (stochastic 1 000-game protocol) at **9.47** GPU-days of training.

A paper distilling the contributions and results of this thesis has been submitted to IEEE CoG 2026 (currently under review).

MicroRTS Tournament Framework

6

Competition leaderboards change yearly with new opponent pools and maps, making direct comparison across editions unreliable. Moreover, as discussed in Section 2.2.1, the 2026 competition shifted to LLM-based agents, precluding participation with this thesis's RL approach. This chapter introduces a reproducible tournament framework, inspired by the MicroRTS competition and anchored on its 2023 edition. It comprises a tournament runner, twelve evaluation maps, fifteen benchmark agents spanning seven competition editions (2017–2024), and five ranking metrics.

6.1 Tournament Infrastructure

The tournament pipeline (Figure 6.1) consists of three phases (configuration, execution, analysis), orchestrated by a unified Python command-line interface (CLI).

1. Configuration: a JSON file specifies all tournament parameters (maps, competing agents, games per matchup, time budgets, maximum game length, and optional flags for trace saving). This declarative format ensures full reproducibility of any tournament.

2. Execution: a Python tournament runner drives the MicroRTS Java simulation engine through a *JPype* bridge (Chapter 7). The runner loads pre-compiled agent JARs at runtime. For each ordered pair of agents on each map, the engine plays a configurable number of games with alternating starting positions (avoiding positional bias) and records the outcome, game duration, and optional game traces; the concrete per-tournament game counts are reported with each experiment in Chapter 10. Games of the same matchup are vectorized using MicroRTS parallel environments to improve throughput, and matchup groups are distributed across SLURM (Simple Linux Utility for Resource Management) cluster jobs to balance computation time.

3. Analysis: a post-processing pipeline parses the raw CSV output into structured JSON, computes all five ranking metrics (Section 6.4) per map and globally, and generates PDF visualizations.

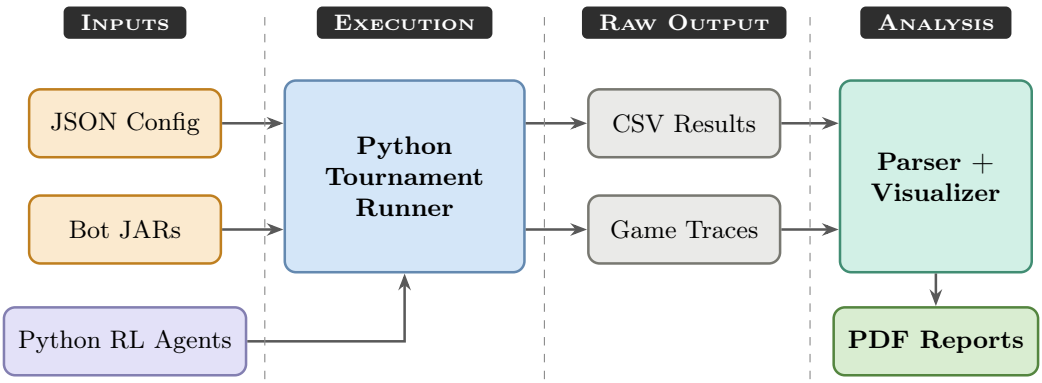


Figure 6.1: Three-phase architecture of the tournament pipeline.

6.2 Evaluation Maps

The benchmark adopts the twelve maps (Table 6.1) of the IEEE CoG 2023 MicroRTS competition[1], split into two pools. The *open pool* contains the eight publicly available maps that have remained the standard set across all competition editions, enabling direct comparison with published results. The *closed pool* contains the four maps held out for the 2023 edition, unknown to competitors before the event itself. These maps are excluded from training here, replicating the closed-pool conditions of the actual competition and providing a held-out generalization test.
The 2023 edition is chosen as the anchor because it is the year RAISocketAI competed, making it the most relevant baseline for this thesis. The twelve maps span grid sizes from 8×8 to 64×64 and vary in starting conditions (number of bases, workers, and barracks), in terrain complexity (open arenas, choke points, dense maze layouts), and in resource distribution (symmetric, contested, or asymmetric). The *Cycles* column reports the maximum game length, expressed in MicroRTS timesteps, which scales with grid size. Visual representations of all maps are provided in Appendix B.

Table 6.1: Characteristics of the twelve evaluation maps, ordered by grid size within each pool.

Map	Size	Cycles	B	W	Bar.	Res. / Total	Key Challenge
Open maps (available for training and evaluation)							
basesWorkers8x8A	8×8	3 000	1	1	0	5 / 40	Minimal; fast games; pure tactical micro
FourBasesWorkers8x8	8×8	3 000	4	4	0	20 / 280	Multi-base management; resource-rich
NoWhereToRun9x8	9×8	4 000	1	0	0	5 / 80	No workers; contested central resource line
basesWorkers16x16A	16×16	4 000	1	1	0	5 / 100	Similar to 8×8 variant with longer horizons
TwoBasesBarracks16x16	16×16	4 000	2	0	2	10 / 120	Pre-built barracks; early military production
DoubleGame24x24	24×24	5 000	2	4	0	10 / 284	Wall corridors; choke points; two-front war
BWDistantResources32x32†	32×32	6 000	1	1	0	20 / 230	Large map; distant resources
BloodBath.scmB†	64×64	8 000	1	0	0	5 / 160	BroodWar conversion; complex maze terrain
Closed maps (evaluation only)							
itsNotSafe	15×14	4 000	1	1	1	5 / 120	Central wall choke point; pre-built barracks
basesWorkers24x24A	24×24	5 000	1	1	0	5 / 100	Similar to 16×16 variant with longer horizons
basesWorkers32x32A	32×32	6 000	1	1	0	5 / 100	Similar to 24×24 variant with longer horizons
chambers32x32	32×32	6 000	1	1	0	5 / 260	Dense maze terrain; multiple chambers

B = bases; W = workers; Bar. = barracks; Res. / Total = starting resources per player / total map resources.
†Asymmetric resource distribution.

6.3 Benchmark Agents

The benchmark spans seven competition editions (2017–2024)[2]. For each year, the highest-ranked open-source agent was selected (Table 6.3), prioritizing paradigm diversity (rule-based, search-based, hierarchical, and DRL). Additional agents were included to cover distinct algorithmic families. All competition agents have been introduced in detail in Chapters 4 and 5.
These competition agents are complemented by four built-in MicroRTS agents (Table 6.2). Three of the four use fixed scripted strategies and establish a controlled difficulty floor against which all other methods can be compared. Tables 6.2 and 6.3 list all selected agents with their key characteristics.

[1]R. Oliveira, MicroRTS 2023 competition, GitHub: https://github.com/rubensolv/MicroRTS2023Competition.
[2]The 2025 edition was excluded because the competing agents' source code references were not yet publicly available in an accessible format at the time of writing.
[3]SSS searches over a small set of high-level strategies rather than individual unit actions.

Table 6.2: Built-in MicroRTS baseline agents used for controlled evaluation.

Paradigm	Agent	Strategy	Role in Evaluation
Rule	***RandomBiasedAI***	Random actions with attack bias	Lower bound; difficulty floor
Rule	***WorkerRush***	All-in worker rush	Early aggression test
Rule	***LightRush***	All-in light rush (Listing 4.1)	Standard rush strategy test
Search	***NaiveMCTS***	MCTS + CMAB	Search-based planning

Table 6.3: Competition agents and GridNet baseline, selected for evaluation, spanning 2017–2024.

Year	Paradigm	Rank†	Agent	Key Characteristics
2017	Hierarchical	1st	***StrategyTactics***	Puppet Search with scripted action abstractions and NaiveMCTS for micro-management
2018	Hierarchical	1st	***Tiamat***	Evolved action abstractions with real-time Strategy Selection Search (SSS)[3]
2019	Hierarchical	1st	***UtsImass***	Pre-game self-learning to optimize build parameters; in-game rule-based control with JPS pathfinding
2019	Hierarchical	2nd	***MixedBot***	Integrates Tiamat for strategic search and CMAB-based tactical execution with dynamic time allocation
2019	Search	3rd	***Droplet***	Guided Naive Sampling: script-biased MCTS that steers search toward promising action sequences
2020	Rule	1st	***CoacAI***	Heuristic economy management, defensive positioning, A* pathfinding
2021	Rule	1st	***Mayari***	Greedy per-action heuristic scoring with local distance estimation
2021	DRL	N/A	***GridNet***	PPO with GridNet encoder-decoder; baseline for all DRL work in MicroRTS
2023	Rule	3rd	***ObiBotKenobi***	Role assignment and Dijkstra-based pathfinding with collision avoidance
2023	DRL	1st	***RAISocketAI***	PPO with the DoubleCone convolutional encoder-decoder architecture, self-play training, map-specific fine-tuning
2024	Rule	1st	***TMA***	Four strategy combinations (attack/defense × worker/military), recalibrated every 500 cycles

†Rank refers to the agent's placement in its listed competition year. GridNet did not enter any competition but serves as a DRL baseline.

6.4 Ranking Metrics

6.4.1 Primary Metric: Win Rate

The pairwise win rate between agents i and j is:

$$\mathrm{WR}(i,j) = \frac{w_{ij} + 0.5\, d_{ij}}{n_{ij}}, \tag{6.1}$$

where w_{ij}, d_{ij}, and n_{ij} are respectively the wins, draws, and total games of agent i against agent j. Draws contribute half a win to each player (standard round-robin convention).
The overall win rate of agent i is the average across all opponents:

$$\mathrm{WR}_i = \frac{1}{n-1} \sum_{j \neq i} \mathrm{WR}(i,j). \tag{6.2}$$

6.4.2 Complementary Game-Theoretic Metrics

Non-transitive cycles (A beats B, B beats C, C beats A) allow weak agents to artificially inflate win-rate scores. Win rate is therefore complemented with four game-theoretic metrics, computed both globally and per map. Each captures a different aspect of agent strength, as summarized in Table 6.4. Appendix C provides the mathematical formulations and a worked 4-agent example illustrating how the four metrics produce complementary rankings of the same pool.

Table 6.4: Ranking metrics and the evaluation question each addresses.

Metric	Perspective	Question
Win Rate	Aggregate performance	How often does the agent win overall?
Nash Averaging	Worst-case robustness	How well does the agent perform against an optimally adversarial opponent mix?
α-Rank	Evolutionary stability	Can the agent's strategy resist invasion and dominate a competing population?
Copeland Score	Pairwise dominance	How many opponents does the agent beat head-to-head?
Regret	Robustness gap	How much worse does the agent perform than the pool's best counter against each opponent?

Nash Averaging [81] solves the win-rate matrix as a symmetric zero-sum meta-game and evaluates each agent by its expected payoff against the Nash equilibrium mixture. The resulting ranking is invariant to adding redundant or dominated agents. When multiple Nash equilibria exist, scores outside the equilibrium support are not unique. Agents that only beat weak opponents rank low; agents that hold up against the strongest competition rank high.

α-Rank [82] models evolutionary dynamics over a population of agents. Starting from a monomorphic population playing a single strategy, it measures how easily a mutant using a different strategy can invade and take over. Agents that resist invasion and successfully displace others receive higher scores. Unlike Nash averaging, α-Rank always produces a unique ranking.

Copeland Score [83] counts net pairwise victories: each head-to-head win contributes +1, each loss −1, and tied matchups 0 to both agents, regardless of the margin. An agent that beats every opponent is a *Condorcet winner*. Because only the direction of each matchup matters, not whether an agent wins 51% or 99%, the Copeland score captures dominance breadth independently of magnitude.

Regret [84] measures how far an agent falls short of the best available alternative for each matchup. If another agent in the pool achieves 90% win rate against opponent j while a given agent only achieves 60%, its regret for that matchup is 30%. Both *average regret* (overall consistency) and *worst-case regret* (single most exploitable matchup) are reported.

MicroRTS Environment Stack

7

The MicroRTS engine runs on the Java Virtual Machine (JVM) while the RL agent runs in Python. This chapter describes the environment stack that bridges them. Section 7.1 details the Java–Python bridge and the per-step interaction cycle. Sections 7.2 to 7.4 then specify the three signals exchanged through this bridge: the observation tensor, the factored action space with its masking, and the reward signal. Section 7.5 introduces the vectorized environments that orchestrate parallel games, and Section 7.6 the composable wrappers that augment them at training time. The implementation builds on Gym-μRTS [11] with substantial modifications. Figure 7.1 gives the resulting data flow.

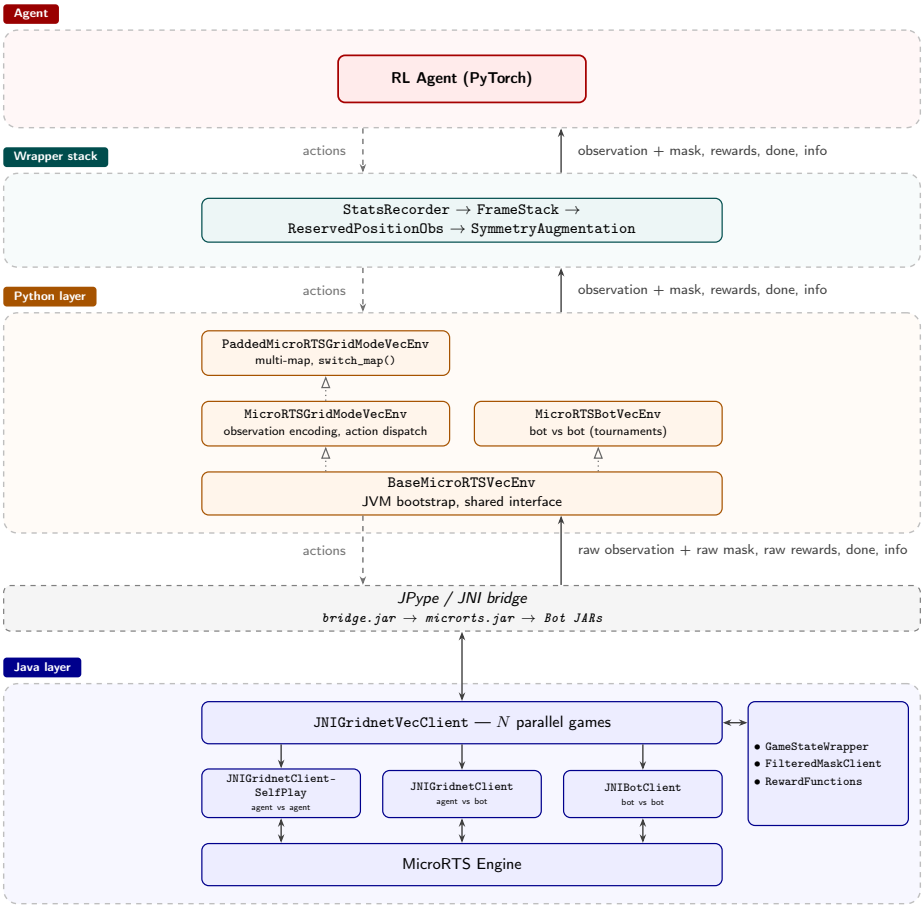


Figure 7.1: Full stack of the Java–Python training bridge, from game engine to RL agent. Read bottom to top: the Java layer executes the game simulation, the JPype bridge batches communication, the Python environment exposes Gym-compatible observations and actions, wrappers transform the training signal, and the RL agent consumes the final tensors.

7.1 Java-Python Bridge

7.1.1 *JPype* Integration

Communication between the two languages happens in-process through *JPype*, which embeds a JVM inside the Python process via the Java Native Interface (JNI). A critical constraint is that the JVM can be started exactly once per process, which shapes the design of the entire stack: all environments, parallel games, and map switches must share a single JVM instance. This rules out a simple design where each environment independently starts and stops its own JVM. Within the shared JVM, a *vectorized client* manages N parallel game instances as a single object. Each call to its reset, step, and mask-extraction methods operates on all N games simultaneously and returns batched arrays. This design minimizes JNI crossing overhead: rather than N separate Java calls per step, a single call processes the entire batch. The same client supports three game modes that can coexist within one batch: *self-play* (the agent controls both sides via paired environments), *RL-vs-bot* (the agent vs a Java AI), and *bot-vs-bot* (two Java AIs, used exclusively for tournaments and producing no observations).

7.1.2 End-to-End Step Cycle

A complete step cycle traverses the full stack as follows:

1. Get action masks: Python requests the action masks from the vectorized client, which returns a binary tensor of shape $(N, H, W, 1+78)$. The first channel is the *source unit mask*, marking cells that contain a controllable unit; the remaining 78 channels (Table 7.3) mask individual action components. Python separates the source unit mask and reshapes the remaining channels to $(N, H\times W, 78)$.

2. Agent decision: the neural network maps the (N, H, W, C) observation tensor (Section 7.2) to 78-dimensional per-cell logits structured by the action representation (Section 7.3); the $(N, H\times W, 78)$ mask filters these into seven masked categoricals per cell, from which actions are sampled, yielding an $(N, H\times W, 7)$ tensor.

3. Action dispatch: Python keeps only the actions for cells containing controllable units (source unit mask), prepends each cell's grid index, and dispatches the result to Java. The bridge converts each entry into an in-game command and assigns the default *None* to idle units.

4. Game step: the Java client advances all N games by one tick, executing both the RL agent's actions (received from Python) and the bot AIs' actions (computed internally). It returns raw observations, raw masks, per-reward-function values (Section 7.4), done flags, and info dictionaries. When a game terminates (or exceeds the maximum step limit), the client performs an *auto-reset*[1].

5. Post-processing: observations are encoded into the agent's input format, rewards are combined via a weighted sum (Section 7.4), done flags trigger episode-end handling (player alternation, map cycling, self-play pair synchronization), and the cycle repeats.

7.2 Observation Encoding

The Python layer supports two observation encodings, both producing a spatial tensor of shape (H, W, C) by stacking encoded planes along the channel axis. The *standard* encoding, inherited from Gym-μRTS [11], represents every feature as a one-hot category, yielding a binary tensor with $C = 29$. The *extended* encoding, adapted from RAISocketAI [12], combines continuous values,

[1]Auto-reset records the terminal signals, resets the game to its initial state, and returns the next observation alongside the terminal outcome.

Table 7.1: Standard observation encoding ($C = 29$ channels).

Feature	Classes	Encoding	Channels
Hit points	$\{0, 1, 2, 3, \geq 4\}$	one-hot clipped	5
Resources carried	$\{0, 1, 2, 3, \geq 4\}$	one-hot clipped	5
Owner	{neutral, self, opponent}	one-hot	3
Unit type	$\mathcal{K} \cup \{\text{empty}\}$	one-hot	7+1
Current action	$\mathcal{A}$	one-hot	6
Terrain	{passable, wall}	one-hot	2
Optional: visibility[†]	{visible, explored}	binary	2
Total			**29–31**

[†]Only relevant under fog-of-war, which is not used in this thesis. These channels are excluded from all experiments ($C = 29$).

Table 7.2: Extended observation encoding ($C = 73$ channels).

Feature	Classes	Encoding	Channels
Hit points	$[0, 10]$	continuous (normalized)	1
Resources	$[0, 40]$	continuous (normalized) + threshold (≥ 1)	2
Owner	{neutral, self, opponent}	one-hot	3
Unit type	$\mathcal{K} \cup \{\text{empty, pending}\}$	one-hot	7+2
Current action	$\mathcal{A}$	one-hot	6
Move direction	$\mathcal{D} \cup \{\text{idle}\}$	one-hot	4+1
Harvest direction	$\mathcal{D} \cup \{\text{idle}\}$	one-hot	4+1
Return direction	$\mathcal{D} \cup \{\text{idle}\}$	one-hot	4+1
Produce direction	$\mathcal{D} \cup \{\text{idle}\}$	one-hot	4+1
Produce type	$\mathcal{K} \cup \{\text{idle}\}$	one-hot	7+1
Attack direction	$\mathcal{D} \cup \{\text{idle, ranged}\}$	one-hot	4+2
ETA[†]	$[0, 255]$	continuous (normalized) + thresholds (≥ 5, ≥ 10)	3
Terrain	{passable, wall}	binary	1
Per-cell subtotal			59
Player resources (self)[‡]	$[0, +\infty[$	continuous (normalized) + 6 thresholds[§]	7
Player resources (opp.)[‡]	$[0, +\infty[$	continuous (normalized) + 6 thresholds[§]	7
Optional: visibility[‖]	{visible, explored}	binary	2
Total			**73–75**

[†] Estimated Time of Arrival (ETA): remaining game ticks before the current action completes; clamped to $[0, 255]$ and normalized to $[0, 1]$.
[‡] Normalized by 32; values above are treated as equivalent. These global resource channels are scalar values broadcast identically to every cell.
[§] Thresholds at $\geq 1, 2, 3, 5, 10, 32$, matching unit production costs: Worker (1), Light/Ranged (2), Heavy (3), Barracks (5), Base (10).
[‖] Only relevant under fog-of-war, which is not used in this thesis. These channels are excluded from all experiments ($C = 73$).

threshold flags, and one-hot fields, reaching $C = 73$ and exposing information the binary scheme cannot represent.
Following Gym-μRTS convention, unit ownership is encoded from the observing player's perspective: the network always sees its own units identically, regardless of which side it controls.

Standard encoding ($C = 29$ channels). The standard encoding uses the `GameState` class to extract 6 raw features per cell. Table 7.1 details the resulting channels and Figure 7.2 illustrates the encoding on three example cells.

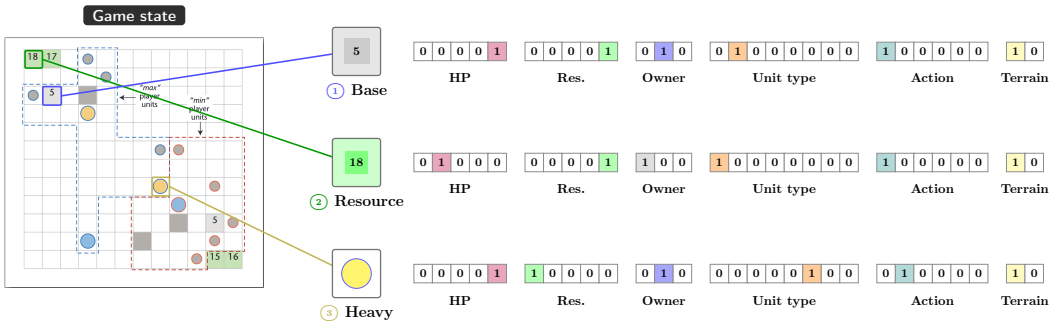


Figure 7.2: Standard observation encoding: three cells from a game state and their encodings. Each cell of the $H \times W$ grid is encoded independently, producing a tensor of shape $(H, W, 29)$.

Extended encoding ($C = 73$ channels). The extended encoding uses `GameStateWrapper` to extract 13 features per cell, complemented by global resource counts broadcast identically to every cell. Table 7.2 details the full encoding.

7.3 Action Space and Masking

The agent outputs per-cell action logits using the GridNet factored action representation. For each of the $H \times W$ grid cells, the network produces logits over seven components (Table 7.3).

Table 7.3: Factored action space per grid cell.

Component	Classes	Encoding	Channels
Action type	$\mathcal{A}$	discrete	6
Move direction	$\mathcal{D}$	discrete	4
Harvest direction	$\mathcal{D}$	discrete	4
Return direction	$\mathcal{D}$	discrete	4
Produce direction	$\mathcal{D}$	discrete	4
Produce type	$\mathcal{K}$	discrete	7
Attack target	7×7 relative grid†	discrete	49
Total per cell			**78**

†A 7×7 grid centered on the acting unit. Index $i = (3 + \Delta y) \times 7 + (3 + \Delta x)$, where $(\Delta x, \Delta y)$ is the relative offset to the target. Index 24 corresponds to the unit's own cell.

The action space is defined as a `MultiDiscrete` of size $H \times W \times 78$ [11, 85]. At each timestep, the

environment provides a binary mask of shape $(N, H \times W, 78)$ indicating valid actions [25]: before sampling, invalid logits are set to -10^8, zeroing their softmax probability. Without this masking, agents would waste most of their exploration budget on illegal actions. Independent per-cell masking, however, still allows a subtle failure mode known as *destination collision*: two units moving or producing into the same cell within a single step, causing one action to fail silently. Inspired by RAISocketAI [12], *filtered masking* tracks destinations of in-progress actions and excludes those cells before generating masks for newly idle units. Figure 7.3 illustrates the improvement.

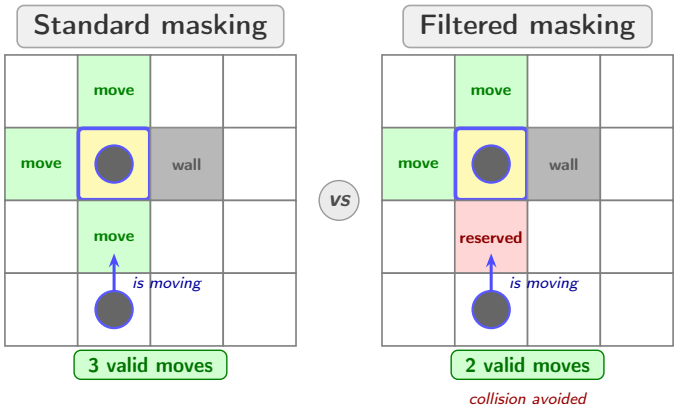


Figure 7.3: *Standard* vs *destination-aware* filtered masking. Left: the selected worker (yellow cell) has three valid moves. Right: cell (2,2) is blocked because the other worker is already moving there.

7.4 Reward Functions

The scalar reward at each step is $r_{t+1} = \mathbf{w}^\top \mathbf{r}_{t+1}$ (Section 2.3.2), where $\mathbf{r}_{t+1}$ is a 9-component raw reward vector [86] and $\mathbf{w}$ the weight vector. Table 7.4 lists the nine components: a single sparse Win/Draw/Loss signal at game termination, plus eight dense rewards triggered by intermediate game events. The terminal signal alone yields too few learning samples over the thousands of timesteps of a typical game, so these shaped rewards [87] densify the feedback, with default weights set so that the terminal outcome still dominates the cumulative return.

Table 7.4: Reward functions available during training.

Reward function	**Signal description**	**Type**	**Default weight**
Win/Draw/Loss	+1 on win, 0 on draw, −1 on loss	sparse	10.0
Resource gathering	+1 per resource unit delivered to a base	dense	1.0
Worker production	+1 per worker trained at a base	dense	1.0
Light production	+1 per light unit trained at barracks	dense	4.0
Heavy production	+1 per heavy unit trained at barracks	dense	4.0
Ranged production	+1 per ranged unit trained at barracks	dense	4.0
Base construction	+1 per base built by a worker	dense	0.1
Barracks construction	+1 per barracks built by a worker	dense	0.1
Attack damage	+1 per attack action issued on an enemy	dense	1.0

7.5 Vectorized Environments

The original Gym-μRTS codebase provided a single monolithic environment class handling all use cases. It is refactored here into three specialized classes built on a shared base, separating bot-only evaluation from RL training and adding support for the extended encoding, filtered masking, self-play, multi-map training, and side alternation introduced in Sections 7.2 to 7.4.

7.5.1 Environment Classes

Three vectorized environment classes specialize the per-step logic:

Bot-vs-bot environment. Runs N parallel games between two Java AIs and is used exclusively for round-robin tournament evaluation. No observation tensor is returned; only the win/draw/loss outcome is collected for each game.

Agents environment. The main training environment. It manages $N_{\text{sp}} + N_{\text{bot}}$ parallel games combining self-play pairs (the agent controls both sides via a shared underlying game instance) and RL-vs-bot games (the agent vs a Java AI). Bot games start on a configured side (P0 or P1) and optionally alternate sides at episode end to balance the experience distribution. The per-step data flow follows the pipeline of Section 7.1.2.

Padded environment. Extends the agents environment with zero-padding to enable multi-map training without restarting the JVM (Section 7.5.2 below).

7.5.2 Padding for Multi-Map Training

A neural network expects a fixed input shape throughout training, but MicroRTS maps have different dimensions. RAISocketAI [12] rewrote the entire Java–Python bridge to natively dispatch on map shape at runtime. An alternative would project every map to a common fixed-size representation, but this would discard the per-cell spatial structure on which the GridNet factored action representation depends (Section 7.3). Given the complexity of the first approach (compounded by the JVM single-startup constraint, Section 7.1.1) and the information loss of the second, a simpler approach is adopted here: padding all observations to a fixed maximum size, as illustrated in Figure 7.4.

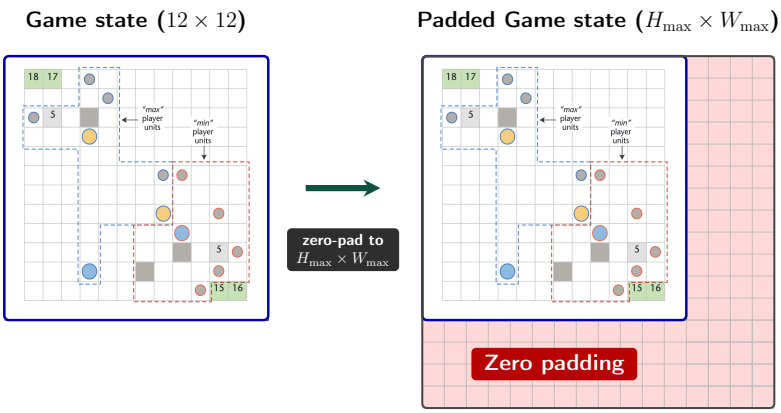


Figure 7.4: Padded environment for multi-map training. The game state (left) is embedded into a fixed-size tensor sized to the largest map in the training pool (right). Padded cells contain zeros and receive all-zero action masks, preventing the agent from acting outside the actual map boundaries.

The padded environment maintains two sets of dimensions: the *actual* map size (H, W) used by the Java engine, and a fixed *padded* size $(H_{\max}, W_{\max})$ seen by the agent. Observations are zero-padded from (N, H, W, C) to $(N, H_{\max}, W_{\max}, C)$, with padded cells appearing as empty terrain. Action masks follow the same treatment, so that padded cells receive all-zero masks and the agent cannot act on them. Actions go in the reverse direction: cropped to the actual (H, W) region before being dispatched to Java. This approach preserves a uniform tensor shape across all maps but introduces computational overhead proportional to the padded-to-actual area ratio, since convolutional layers still process the padded cells (the action mask only applies at output time). For instance, a 16×16 map padded to 64×64 wastes 93.75% of the spatial computation on empty cells.
Together, these properties enable the central capability that motivated the padded environment: training on multiple maps within a single run. The active map can be switched at any time without restarting the JVM, and since the padded dimensions remain fixed, the network architecture does not change across maps. The base environment classes could not support this; each map size would have required its own network and a separate training run.

7.6 Composable Wrappers

A key design principle separates *environments* from *wrappers*. The vectorized environment classes handle the Java interface, game simulation, and core observation/action encoding. Most additional processing is implemented as composable Gym wrappers that sit between the environment and the agent, ensuring each wrapper can be toggled independently via CLI flags.
Four wrappers are available, composed in a fixed order:

1. **`StatsRecorder`.** Accumulates each of the nine raw reward components separately over an episode, as both raw and discounted ($\gamma^t \cdot r_{t+1}$) sums. The cumulative per-component values are emitted at episode termination for monitoring tools such as TensorBoard.

2. **`FrameStack`.** Stacks the last k observations along the channel axis, producing tensors of shape $(H, W, k \cdot C)$ [28]. This gives the convolutional encoder access to temporal patterns (unit movement, production sequences) without recurrence. Action masks remain unstacked since they depend only on the current game state.

3. **`ReservedPositionObs`.** Companion to destination-aware filtered masking (Section 7.3): the mask filters out reserved cells but does not reveal them to the agent. This wrapper appends a binary observation channel ($C \to C + 1$) flagging cells reserved by pending actions, letting the network plan around spatial conflicts.

4. **`SymmetryAugmentation`.** Exploits the spatial symmetry of MicroRTS maps. At each episode start, each environment is assigned one of four random flip modes (identity, horizontal flip, vertical flip, or 180° rotation). Observations and action masks are transformed accordingly; actions are unflipped before being dispatched to the engine. Direction-dependent components (move, harvest, return, produce directions, and the 7×7 attack target grid) are remapped consistently. This effectively quadruples the diversity of training positions without additional environment interaction [88]. It is applied only during training, never during evaluation, to ensure deterministic and unbiased assessment.

Spatial and Entity-Based Neural Architectures

8

The neural network is the function approximator that PPO optimizes. Its capacity and inductive bias bound what the agent can ultimately represent. Seven architectures are presented in this chapter, each adding one or two contributions on top of the previous one. The progression is therefore incremental by construction. Rather than presenting these architectures chronologically, the chapter organizes the architectural requirements of an RTS network into three spatial and relational reasoning axes; temporal and curriculum aspects are addressed in Chapter 9. Each contribution naturally targets one specific axis. Section 8.1 introduces these three axes and the actor-critic skeleton common to all seven networks. Sections 8.2 to 8.4 present the architectures that progressively address each axis. Section 8.5 synthesizes the contributions into the final UECD architecture. Section 8.6 provides a recap table of all seven designs, whose contributions are quantified empirically in the architecture ablation of Chapter 10.

8.1 Common Skeleton and Reasoning Axes

8.1.1 Actor-Critic Skeleton

All seven architectures follow the shared-encoder actor-critic pattern recalled in Chapter 3. A single encoder f_θ first compresses the observation tensor (Section 7.2) to a bottleneck representation of shape (B, C', H', W'), where C' is an architecture-dependent channel count, (H', W') the compressed spatial resolution and B the batch size (the N parallel environments of Chapter 7 during rollout, or the minibatch size during the PPO update). A decoder then upsamples this representation back to full resolution (B, C'', H, W) before it is consumed by the two heads. Sharing this encoder-decoder backbone between the actor and the critic, rather than training two independent networks, roughly halves the parameter count for the same representational capacity, since both heads need the same spatial features (unit positions, threat regions, resource distribution). The actor head outputs per-cell logits of shape $(B, H, W, 78)$, factored as in Section 7.3 and masked to -10^8 before sampling. The critic head outputs a scalar value estimate $\hat{V}(s)$.

8.1.2 Three Reasoning Axes

Strategic play in MicroRTS requires three qualitatively different forms of reasoning:

1. Local spatial reasoning. Most tactical decisions are settled within a unit's immediate neighborhood: which adjacent cell to step into, which neighbor to attack, where to deposit a carried resource. A Worker-vs-Worker engagement on a contested resource patch is decided entirely by the local configuration of the two units and the adjacent terrain. Convolutional layers are the canonical inductive bias for this regime [89]: their translation equivariance encodes the prior that such an engagement looks the same whether it occurs near the home base or at the opposite corner of the map, and weight sharing extracts the same tactical patterns from any position with a fraction of the parameters of a fully connected alternative.

2. Long-range relational reasoning. Units act as a coordinated group, and the relationships that matter are *entity-to-entity* rather than cell-to-cell. A Heavy advancing in front of friendly Ranged

units must absorb incoming damage while the Ranged behind focus their fire on the same enemy to bring it down within a single tick; a Worker rush detected at the opponent base should trigger a defensive recall regardless of how far the defenders stand. These dependencies are sparse, since only the n_t active units participate rather than $H \times W$ cells, and irregular, since any pair of units may interact at any spatial separation. A purely convolutional backbone captures them only indirectly, through stacked receptive fields that must grow until they happen to overlap the relevant units. Self-attention over unit tokens fits this structure directly, computing pairwise interactions at any distance in a single layer.

3. Global spatial reasoning. A third class of decisions hinges on region-to-region trade-offs not localized to any specific entity: committing to a frontal push vs answering pressure on the home base, or shifting production between two barracks on opposite sides of the map. Standard CNNs reach such a global view only by stacking enough downsampling layers for the receptive field to span the map, which mixes long-range information with the strong locality bias of convolutions and dilutes the signal. Spatial self-attention applied at the bottleneck provides this view directly: every spatial token attends to every other in a single layer, at a cost quadratic in the reduced number of tokens rather than the full grid.

Existing convolutional architectures for MicroRTS, including GridNet [11] and the DoubleCone backbone of RAISocketAI [12], primarily address the local axis and capture long-range dependencies only indirectly through stacked convolutions and downsampling. The final architecture of this thesis (Section 8.5) covers the three axes explicitly through three complementary mechanisms: a CNN backbone for local reasoning, a Transformer encoder over unit tokens for relational reasoning, and a self-attention layer over the bottleneck grid for global reasoning. This three-way design mirrors AlphaStar's entity-spatial pair (Section 5.2), adapted here to MicroRTS scale.

8.2 Axis 1: Local Spatial Reasoning

Three architectures (`GridNet`, `IMPALA-CNN`, `U-Net`) progressively refine the convolutional backbone responsible for the local axis. Convolutional Block Attention Module (CBAM), a more expressive attention module, is introduced here and later folded onto `U-Net-Entity`.

GridNet. `GridNet` [90], with per-cell action prediction motivated in Section 5.3, serves as the baseline, following the implementation of [11]. Its encoder applies four convolutional blocks (3×3 convolution, ReLU (Rectified Linear Unit) activation function, stride-2 max-pooling), expanding channels as $C \to 32 \to 64 \to 128 \to 256$ and collapsing the spatial resolution to a 1×1 bottleneck. The actor mirrors this path with four transposed convolutions; the critic flattens the 256-dimensional bottleneck through a two-layer MLP. Two limitations follow. First, the extreme bottleneck discards spatial detail that the per-cell decoder must then reconstruct from a single 256-dimensional vector. Second, the flatten in the critic ties the network to a fixed map size of 16×16.

IMPALA-CNN. `IMPALA-CNN` replaces `GridNet`'s plain convolutions with the residual encoder of the Importance-Weighted Actor-Learner Architecture (IMPALA) [71]: three stages of {Conv → MaxPool → 2×ResBlock} producing 32, 64, and 128 channels and compressing the spatial resolution to $H/8 \times W/8$. Each ResBlock follows a pre-activation scheme (ReLU → Conv → ReLU → Conv + skip). These residual blocks enable the deeper backbones of subsequent architectures without unstable gradient flow. The decoder mirrors the encoder with transposed convolutions, and the critic replaces `GridNet`'s fixed flatten with adaptive average pooling. From this architecture onward, the network is decoupled from the map resolution and can be used in the padded multi-map training environment of Section 7.5.2.

U-Net. U-shaped Neural Network (`U-Net`) introduces two changes to the IMPALA-CNN backbone. First, `U-Net` skip connections [91] route encoder features at matching resolutions directly into the decoder before each upsampling stage, addressing the spatial detail loss observed at the `GridNet` and `IMPALA-CNN` bottlenecks; attention gates on such skip connections have also been studied [92]. The encoder uses three stages with two stride-2 convolutions instead of max-pooling, reaching a spatial resolution of $H/4 \times W/4$. Second, every residual block is gated by SE channel attention [93], the same mechanism used in RAISocketAI's DoubleCone backbone (Section 5.3): global average pooling and a bottleneck MLP produce a per-channel importance vector that rescales feature maps to emphasize channels relevant to the current situation.

CBAM attention. CBAM [94] replaces SE in every residual block, extending it with an explicit spatial attention stage (Figure 8.1). The feature map is pooled along the channel axis (average and max), the two saliency maps are concatenated, and a 7×7 convolution followed by a sigmoid produces a per-position weight in $[0,1]^{H \times W}$. The channel gate inherited from SE is applied first, emphasizing situationally relevant feature maps (e.g., Ranged-unit and HP channels during combat); the spatial gate follows, emphasizing critical map regions (contested frontlines, exposed worker positions, choke points). A residual connection then adds the block input back before the final ReLU. The parameter overhead over SE is negligible (≈100 weights per block).

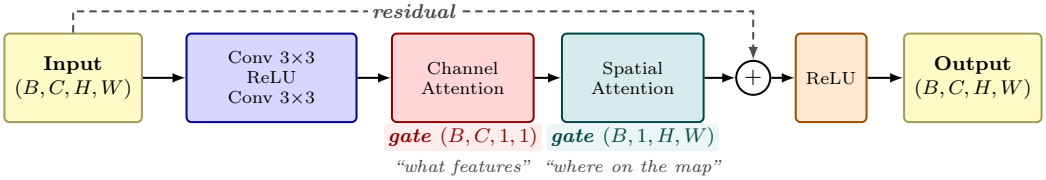


Figure 8.1: CBAM-ResBlock (simplified view). Channel attention selects *what features* matter; spatial attention selects *where on the map* to look. Residual shown in gray dashed.

8.3 Axis 2: Long-Range Relational Reasoning

Although stacked convolutions can in principle bridge long spatial distances, the resulting cell-level features blur identifiable units into pooled channel statistics: two units twenty grid steps apart interact only through several rounds of downsampling, and the output represents not the two specific units but the activity of their respective regions. The relational axis preserves unit identity by maintaining one explicit token per controllable unit and processing the resulting set with a Transformer encoder [95, 96], following AlphaStar's entity-spatial pair (Section 5.2).

Token construction. For each unit on the grid (friendly or enemy), a token is built by concatenating three complementary sources: raw observation features carry unit-specific semantics (type, HP, carried resource, current action) that the CNN may have abstracted away; the CNN features encode spatial context (nearby allies, terrain, threats); and the normalized grid coordinates (row, col) make position recoverable, which the permutation-invariant attention would otherwise discard. A linear projection maps each token to $d_{\text{model}} = 128$, then a two-layer, four-head Transformer encoder processes the resulting $(B, n_t, 128)$ tensor. Since strategic decisions depend equally on intra-player coordination (which Worker should harvest where) and inter-player relations (which enemy Ranged threatens which Light), every unit attends to every other regardless of ownership, with no relational mask; a padding mask only hides unused slots in the fixed-size tensor. The two layers and four heads provide enough relational depth to capture multi-hop interactions.
The enriched representations are projected back to spatial channels and scattered into the convo-

lutional stream by additive insertion at each unit's downsampled grid position. The per-step cost remains manageable at MicroRTS unit counts (n_t is bounded by the unit cap, keeping $\mathcal{O}(n_t^2)$ small) and is acceptable relative to the representational gain (quantified in Chapter 10). It does, however, noticeably reduce throughput, as shown in Table 8.1.

IMPALA-CNN-Entity. `IMPALA-CNN-Entity` is the first integration. A lightweight full-resolution CNN (two 3×3 convolutions, 64 channels) runs in parallel with the `IMPALA-CNN` encoder to provide the per-cell features used in token construction, since the `IMPALA-CNN` bottleneck is too coarse. The enriched tokens are projected and scattered back at the `IMPALA-CNN` bottleneck only. Two limitations follow: the parallel CNN performs redundant feature extraction that the main encoder also does at lower resolution, increasing parameters without sharing computation; and the single bottleneck-scale scatter dilutes the entity information by the time the decoder reaches full resolution.

U-Net-Entity. `U-Net-Entity` combines the Entity Transformer with the `U-Net` backbone of Section 8.2, removing both inefficiencies of the previous design. The parallel CNN is replaced by `U-Net`'s full-resolution skip connection *skip1*, which already encodes high-quality features. Enriched representations are then scattered back at two scales rather than one: into the bottleneck ($H/4 \times W/4$) for global strategic reasoning (army composition, resource allocation), and into the mid-resolution skip connection *skip2* ($H/2 \times W/2$) for spatially precise per-unit influence (positioning, target selection). When multiple units fall in the same target cell after downsampling (more frequent at the bottleneck than at *skip2*), their projected embeddings are summed via PyTorch's `index_add_`, a permutation-invariant aggregation that preserves all per-unit contributions without averaging or learned pooling. Figure 8.2 summarizes the mechanism end-to-end with this backbone.

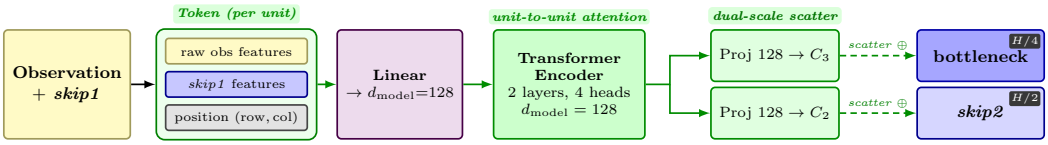


Figure 8.2: Entity Transformer mechanism (simplified view) with the `U-Net` backbone.

U-Net-Entity-CBAM. Folding the CBAM gate of Section 8.2 onto `U-Net-Entity` yields `U-Net-Entity-CBAM`, the sixth architecture. It adds spatial attention on top of the relational backbone at negligible parameter cost, isolating the contribution of the spatial gate in the ablation of Chapter 10.

8.4 Axis 3: Global Spatial Reasoning

Decisions such as committing to a frontal push while pressure mounts on the home base require relating features at opposite ends of the map in a single forward pass. Convolutions are local operators. The theoretical receptive field of a stride-2 3×3 convolutional stack grows only as $\mathcal{O}(2^d)$ with depth d, and the effective receptive field is known to be substantially smaller [97]; even on 16×16 maps, a unit at one edge cannot reliably attend to the opposite edge through convolutions alone. Two complementary mechanisms address this limitation.

Bottleneck self-attention. A four-head spatial self-attention layer is inserted after the bottleneck residual blocks. Each spatial position at the bottleneck is treated as a token of dimension $C_3 = 4C_1$, and standard multi-head self-attention [95] is applied across all tokens, allowing every position to attend to every other regardless of distance. The placement at the deepest stage keeps the cost small: at $H/4 \times W/4$ the grid holds 16× fewer tokens than the full-resolution map, so the $\mathcal{O}(N^2)$ self-attention is $16^2 = 256\times$ cheaper than at full resolution.

Pyramidal depth profile. Earlier architectures use a uniform two residual blocks per U-Net stage. This is replaced by the configuration (2, 3, 4, 3, 2), peaking at four blocks at the bottleneck. Capacity is concentrated there for two reasons. First, bottleneck features synthesize information from the full map and serve two roles: feeding the global self-attention module and receiving the Entity Transformer's scattered output. Depth there therefore has the largest strategic effect. Second, the bottleneck's small spatial resolution makes computation cheapest.

8.5 UECD: Final Architecture

The final architecture, UECD (U-Net-Entity-CBAM-Deep), combines the contributions of the three previous sections. Figure 8.3 shows the simplified structure, with ≈4.7 M parameters at the default base width $C_1 = 48$ and the 73-channel extended observation encoding (Section 7.2). The fully annotated architecture with all channel widths and tensor dimensions is presented in Appendix D.

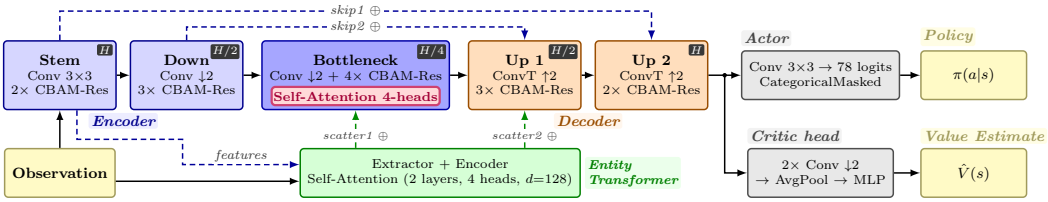


Figure 8.3: UECD architecture (≈4.7 M parameters at C_1=48). Top: CBAM-gated U-Net backbone (blue encoder, orange decoder) with pyramidal depth profile (2, 3, 4, 3, 2) and bottleneck self-attention (red). Bottom: Entity Transformer (green) extracts per-unit features and scatters back into the bottleneck and *skip2* (green dashed). U-Net skips between encoder and decoder (blue dashed).

The four modules of UECD map onto the three reasoning axes as follows. The CBAM-gated U-Net backbone realizes the local spatial axis (Section 8.2). The Entity Transformer with dual-scale scatter into the bottleneck and *skip2* realizes the relational axis (Section 8.3). The bottleneck self-attention layer, together with the pyramidal depth profile (2, 3, 4, 3, 2) that concentrates capacity there, realizes the global spatial axis (Section 8.4). The actor and critic heads read from the decoder's full-resolution output: the actor applies a single 3×3 convolution mapping C_1 channels to the 78 per-cell action logits (masked and sampled as in Section 7.3), while the critic applies two stride-2 3×3 convolutions, an adaptive average pool to 1×1, and a two-layer MLP, producing $\hat{V}(s)$.

8.6 Architecture Summary

Table 8.1 recaps the seven architectures, whose empirical contributions are quantified in Section 10.2.

Table 8.1: Overview of the seven architectures.

Architecture	Params	SPS†	Axes‡	Key addition
GridNet	838 k	2 688	S	Baseline encoder-decoder
IMPALA-CNN	1.02 M	2 416	S	+ Residual blocks, adaptive critic pool
U-Net	2.59 M	2 121	S	+ U-Net skips, SE attention
IMPALA-CNN-Entity	1.37 M	1 903	S / R	+ Entity Transformer
U-Net-Entity	2.90 M	1 518	S / R	+ Entity scatter integrated with U-Net (dual-scale)
U-Net-Entity-CBAM	2.90 M	1 289	S / R	+ Spatial attention (CBAM)
U-Net-Entity-CBAM-Deep	4.72 M	1 176	S / R / G	+ Pyramidal depth profile, self-attention

†Steps per second, measured on *basesWorkers16x16A* with 24 parallel environments.
‡Axes covered: **S** ≡ local **S**patial, **R** ≡ **R**elational, **G** ≡ **G**lobal spatial.

9 Training Procedure and Mechanisms

Chapter 8 fixed the network architecture and, with it, the capacity and inductive biases available to the agent. Architecture alone does not determine behavior: the training procedure decides which of those biases are used and how. Two agents sharing the same UECD backbone can learn very different policies depending on the reward components they receive, the opponents they train against, and the schedule controlling the exploration-exploitation trade-off.

This chapter specifies the training procedure built on top of UECD. The mechanisms are presented in conceptual order, from those closest to per-step action sampling to those targeting global generalization. Figure 9.1 offers a complementary view along a different axis: it places each group on the PPO data-flow loop, with each tag pointing to the stage it modifies. Section 9.1 defines the PPO baseline used by every experiment. Section 9.2 refines how the factored action is sampled and credited. Section 9.3 groups the eight mechanisms acting on the reward and value signals that feed the PPO update. Section 9.4 handles non-stationarity from multiple opponents. Section 9.5 adds auxiliary supervision to the shared encoder. Section 9.6 covers symmetry and multi-map training. Section 9.7 closes with two architectural refinements evaluated alongside the training features rather than the architectures of Chapter 8. Every mechanism is gated by CLI flag and disabled by default (Table 9.2), supporting the controlled feature ablation reported in Chapter 10.

9.1 PPO Baseline Configuration

All agents train with the PPO [34] loop of Section 3.5. The baseline uses PPO-Clip (Equation 3.17) with GAE [33] (Equation 3.13); Table 9.1 lists the full configuration.

Table 9.1: Baseline PPO hyperparameters used by every experiment of Chapter 10 when no flag-gated mechanism is active. Experiment-specific settings (map, opponents, hardware) appear there.

Hyperparameter	Value	
PPO objective		
Discount γ / GAE λ	0.99 / 0.95	
Clip range ϵ	0.1	
Value loss coefficient c_v	0.5	
Entropy coefficient β	$0.01 \rightarrow 0.001$	*linearly annealed (Section 9.3)*
Stabilization		
Learning rate	$2.5 \times 10^{-4} \rightarrow 0$	*linearly annealed to prevent late-training oscillations*
Max gradient norm	0.5	*prevents destabilizing updates*
Advantage normalization	per-minibatch	*prevents single-episode gradient domination*
Rollout and batch		
Rollout length T	512	
Update epochs / minibatches	2 / 3	
Parallel environments	24	
Batch size	12 288	

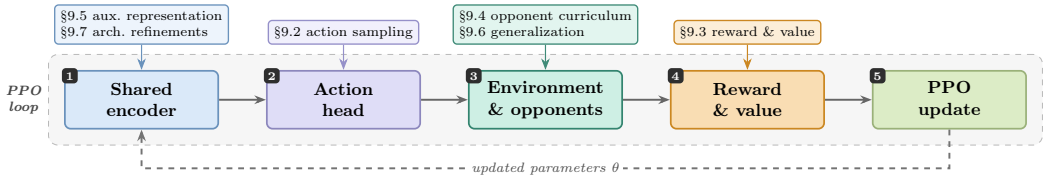


Figure 9.1: Each training mechanism acts on one stage of the PPO loop. The boxes form the core loop of Figure 3.3 in data-flow order; the tags above each box name the sections modifying it.

Table 9.2: Training mechanisms introduced in this chapter, each gated by a CLI flag and disabled by default. Only the principal flag per mechanism is listed; secondary flags tuning internal hyper-parameters keep their defaults. The final group lists observation and environment features from Chapter 7, included here for the feature ablation in Chapter 10.

Mechanism	CLI flag	Section	Acts on
Action sampling			
Autoregressive head	`-autoregressive`	9.2	sub-action logits
Hierarchical masking	`-hierarchical-mask`	9.2	log-prob & entropy terms
Reward and value signal			
Entropy scheduling	`-ent-coef-end`	9.3	entropy bonus
Reward scheduling	`-reward-schedule`	9.3	reward weight vector
Multi-phase schedule	`-phase-steps`	9.3	LR / entropy / blend / vf coef per phase
Build-time rewards	`-reward-unit-build-time`	9.3	production reward
Decomposed value heads	`-triple-value-heads`, `-advantage-weights`	9.3	critic / advantage blend
PopArt normalization	`-popart`	9.3	value targets
HL-Gauss value	`-hl-gauss`	9.3	value loss
PAE	`-pae-keep`	9.3	advantage estimate
Opponent curriculum and self-play			
Adaptive opponents	`-adaptive-opponents`	9.4	env opponent mix
Historical self-play	`-num-selfplay-envs`, `-pfsp`	9.4	env opponents
MCW	`-prioritized-sampling`	9.4	policy-gradient weights
Auxiliary representation			
Spatial ownership	`-aux-spatial`	9.5	shared encoder
Unit count	`-aux-unit-count`	9.5	shared encoder
Temporal contrastive	`-aux-contrastive`	9.5	shared encoder
Opponent action modeling	`-aux-opponent-modeling`	9.5	shared encoder
Generalization and architecture			
Symmetry augmentation	`-augment-symmetry`	9.6	observations / masks
Multi-map + PLR	`-multi-map`, `-plr`	9.6	map sampling
GELU activation	`-gelu`	9.7	encoder / decoder
SPP critic	`-spp-critic`	9.7	critic pooling
Observation and environment (introduced in Chapter 7)			
Extended observations (73ch)	`-extended-obs`	7.2	observation encoding
Filtered masks + reserved obs	`-filtered-masks`, `-reserved-obs`	7.3	action mask / observation
Frame stacking	`-frame-stack`	7.6	observation history

9.2 Action Sampling Refinements

The factored action head of Section 7.3 produces $|\texttt{nvec}| = 7$ sub-action logits per cell, sampled independently and summed uniformly in the PPO loss. Two refinements modify this default. *Autoregressive sampling* introduces dependencies between sub-actions; *hierarchical masking* removes the credit signal of semantically inactive sub-actions. Both alter the policy objective without changing the rollout: the same actions are sampled and dispatched to the engine, only the way logits are produced and credited changes.

Autoregressive sub-action sampling. Sub-actions are not independent: when a unit chooses *produce*, the production direction and unit type should be jointly coherent. Following the autoregressive action decomposition of AlphaStar [8], an autoregressive head conditions each sub-action on the previously sampled ones via learned embeddings. The standard per-component logit $\ell_k = W_k \cdot \mathbf{f} + b_k$ becomes

$$\ell_k = W_k \cdot [\,\mathbf{f},\ e_1(a_1),\ \ldots,\ e_{k-1}(a_{k-1})\,] + b_k, \tag{9.1}$$

where $\mathbf{f}$ is the per-cell spatial feature vector from the actor's pre-projection layer and $e_j(\cdot)$ is a learned embedding of dimension $d_e = 8$ for sub-action j. The first sub-action (`action_type`) is conditioned only on $\mathbf{f}$. The factorization is intra-unit: inter-unit dependencies still flow indirectly through the shared spatial and entity representations of Chapter 8.

Hierarchical sub-action masking. For any given `action_type`, only a subset of the seven sub-actions is semantically active (Figure 9.2). When an agent selects *attack*, the sub-actions for *move direction*, *harvest direction*, *return direction*, *produce direction*, and *produce unit type* have no effect on the environment. The standard implementation [11] nevertheless aggregates all seven log-probabilities and entropies into the PPO objective, diluting credit assignment across non-informative sub-actions. Following RAISocketAI [12], a hierarchical sub-action mask zeroes out the log-probability of each conditional sub-action when its parent `action_type` is inactive. Unlike the invalid-action mask of Section 7.3, which prevents illegal actions at the environment level, this mask operates purely on the PPO objective: the contributions of semantically inactive sub-actions are zeroed in the loss. Let $a_0 \in \{0,1,2,3,4,5\}$ denote the chosen `action_type` and let $\textsc{parent}(k)$ map sub-action k to its activating type:

$$\textsc{parent}(k) = \begin{cases} \text{MOVE} & k = 1 \\ \text{HARVEST} & k = 2 \\ \text{RETURN} & k = 3 \\ \text{PRODUCE} & k = 4 \\ \text{PRODUCE} & k = 5 \\ \text{ATTACK} & k = 6 \end{cases}$$

Sub-action k=0 (the `action_type` itself) is always active. The masked log-probability becomes

$$\log \pi(\mathbf{a} \mid s) = \log \pi_0(a_0 \mid s) + \sum_{k=1}^{6} m_k \log \pi_k(a_k \mid s, a_{<k}), \tag{9.2}$$

where $m_k = \mathbb{1}[\,a_0 = \textsc{parent}(k)\,] \in \{0,1\}$ gates sub-action k on its parent. The same gating applies to the per-sub-action entropy terms. The credit signal becomes cleaner: the distribution over *move direction* updates only when the unit actually moves, rather than receiving phantom gradients during attack, harvest, or production steps.

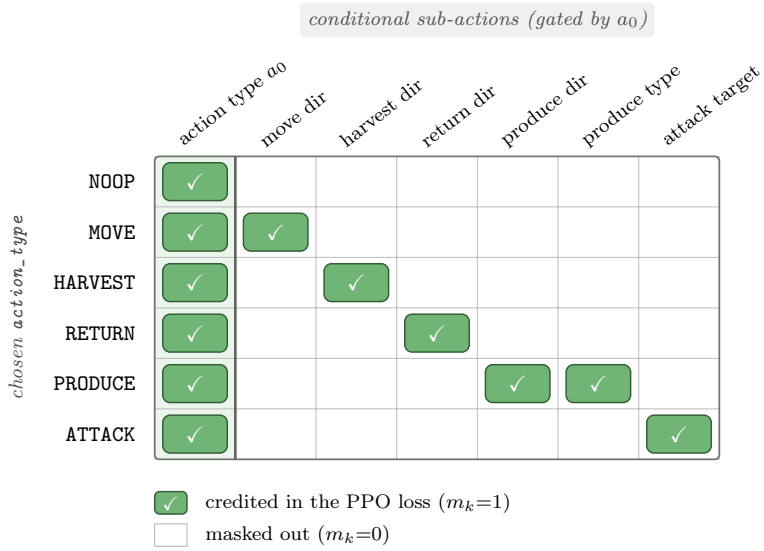


Figure 9.2: Hierarchical sub-action masking. Rows are the chosen `action_type` a_0; columns are the seven sub-actions. A filled cell marks a sub-action credited in the PPO objective; an empty cell is masked out ($m_k = 0$). Only `action_type` is always credited; each conditional sub-action contributes under its single parent, so each row credits at most three of the seven sub-actions.

9.3 Reward and Value Signal

The PPO objective depends on two scalar signals per transition: the reward r_{t+1} feeding the policy gradient through the GAE advantage, and the bootstrapped value $V(s_t)$ that grounds the critic's update. Eight mechanisms refine how these scalars are constructed, scaled, or weighted. The first four reshape the policy update signal: entropy regularization, dense-to-sparse annealing, multi-phase scheduling, and build-time weighting; the next two restructure the value function (decomposed heads, PopArt); the last two are experimental alternatives: HL-Gauss for the value loss and the PAE for the GAE advantage.

Reshaping the Reward

Entropy scheduling. The entropy bonus $H[\pi_\theta]$ in the PPO objective (Equation 3.17) is controlled by a coefficient β, linearly annealed to a configurable end value via the scheduler below. This encourages exploration early, then shifts toward exploitation as the policy matures.

Reward scheduling and multi-phase training. Dense reward signals accelerate early learning but risk producing policies that over-optimize intermediate proxies at the expense of the true objective, a phenomenon formally established by [87]. RAISocketAI [12] addresses this in MicroRTS with a shaped-to-sparse transition [98]. A piecewise-linear scheduler (Figure 9.3) generalizes this transition: given N phases with plateau values $[v_1, v_2, \ldots, v_N]$ and boundary steps $[s_1, e_1, s_2, e_2, \ldots]$, each hyperparameter holds v_i until step s_i, interpolates linearly to v_{i+1} by step e_i, then holds v_{i+1} until the next transition. The dense-to-sparse transition uses this scheduler to interpolate the reward weights from the dense configuration of Table 7.4 to the sparse win/loss configuration $[10, 0, \ldots, 0]$ over $[t_{\text{start}}, t_{\text{end}}]$ (by default, 30% to 70% of training), shifting from "play economically well" toward "win the game". The same scheduler drives the learning rate, entropy coefficient, advantage blend weights, and value loss coefficients, enabling multi-stage curricula within a single run.

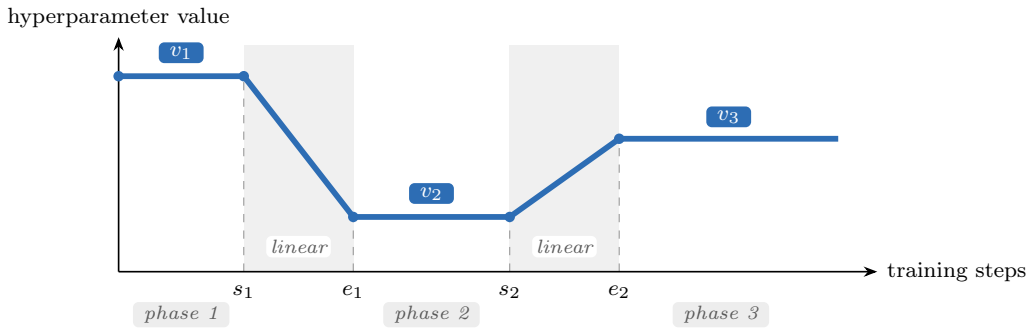


Figure 9.3: Piecewise-linear multi-phase scheduler. Each hyperparameter holds plateau value v_i until step s_i, interpolates linearly to v_{i+1} by step e_i, then holds again. The same scheduler drives the reward weights (dense-to-sparse), learning rate, entropy coefficient, and value-head weights.

Build-time-scaled unit production rewards. The default reward function assigns the same weight (4.0) to producing any military unit. These units have very different production costs in game ticks: Light (80), Ranged (100), Heavy (120). Uniform weighting biases the policy toward cheap units regardless of tactical fit. Following RAISocketAI [12], this mechanism scales each production reward by build time, using $(4.0, 5.25, 6.0)$ for (Light, Ranged, Heavy). The Light and Heavy weights are strictly proportional to build time; Ranged receives an extra 0.25 for its long-range advantage.

Restructuring the Value Function

Decomposed value heads. A single value function struggles to track the shifting reward signal: the critic must estimate returns from dense shaping rewards and sparse win/loss signals at different discount horizons. Up to three independent critic heads decouple this target [86], following RAISocketAI [12] (Table 9.3). Each head receives its own per-step reward: *shaped* sums the 9 components of Section 7.4; *sparse* uses only Win/Draw/Loss; *cost* uses the temporal difference of the normalized Military score. Each head computes advantages via a separate GAE pass (Equation 3.13) and is trained with its own value loss.

The per-head advantages live in the natural scale of their reward signals, which differ substantially. To make contributions comparable, each advantage is z-scored per minibatch (zero mean, unit variance) before being combined into a single advantage driving the policy gradient:

$$\hat{A} = w_{\text{sh}} \cdot \hat{A}_{\text{shaped}} + w_{\text{sp}} \cdot \hat{A}_{\text{sparse}} + w_{\text{co}} \cdot \hat{A}_{\text{cost}}, \qquad w_{\text{sh}} + w_{\text{sp}} + w_{\text{co}} = 1. \tag{9.3}$$

Table 9.3: Value head configuration. Higher γ and λ capture longer-horizon dependencies. Blend weights w sum to 1; value-loss coefficients c_v scale each head's contribution to the total loss.

Head	Reward signal	γ / λ	w	c_v	Role
Shaped	Weighted sum of all 9 components (Table 7.4)	0.99 / 0.95	0.5	0.5	Dense feedback
Sparse	Win / Draw / Loss only	0.999 / 0.99	0.3	0.1	Terminal outcome
Cost	Temporal Δ of Military score†	0.999 / 0.99	0.2	0.2	Army strength proxy

† Cost-head reward $r_t^{\text{co}} = s_t - s_{t-1}$, the temporal difference of the normalized Military score $s_t = \frac{S_{\text{own}} - S_{\text{opp}}}{S_{\text{own}} + S_{\text{opp}} + 1} \in (-1, +1)$, where $S = \sum_{\text{units}} \text{cost} \times (1 + \text{HP}/\text{HP}_{\text{max}})$ is the player's total army value (cost-weighted, HP-adjusted). $s_{t-1} = 0$ at the start of each episode.

PopArt value normalization. When the reward signal varies across orders of magnitude, as in reward scheduling with a shifting target distribution, standard Mean Squared Error (MSE) value loss can produce destabilizing gradient spikes. Preserving Outputs Precisely, while Adaptively Rescaling Targets (PopArt) [99] keeps a running mean μ and standard deviation σ of value targets via EMA:

$$\mu_{\text{new}} = (1-\alpha)\cdot\mu_{\text{old}} + \alpha\cdot\bar{G}_{\text{batch}}, \qquad \sigma^2_{\text{new}} = (1-\alpha)\sigma^2_{\text{old}} + \alpha\cdot\overline{(G-\bar{G}_{\text{batch}})^2}, \tag{9.4}$$

with smoothing factor $\alpha = 3\times10^{-4}$. The critic learns to predict in a normalized space $\hat{V}_{\text{norm}} \approx \mathcal{N}(0,1)$, and predictions are denormalized for GAE computation:

$$\hat{V}(s) = \sigma\cdot\hat{V}_{\text{norm}}(s) + \mu. \tag{9.5}$$

On each update of μ and σ, the last linear layer's weights and bias are simultaneously rescaled so the denormalized output $\hat{V}(s)$ remains identical for every s. The network's predictions thus stay continuous when the normalization statistics change, preserving training stability. When multiple value heads are active, PopArt is applied independently to each.

Experimental Alternatives

Value estimation via classification (HL-Gauss). Histogram Loss with Gaussian Targets (HL-Gauss) [100], recently applied to DRL value estimation by [101], is included as an experimental alternative to scalar value regression. It discretizes the value range into 255 uniformly spaced bins and predicts a categorical distribution over them. Returns are first compressed via the symmetric logarithm $\text{symlog}(x) = \text{sign}(x)\cdot\ln(|x|+1)$ from DreamerV3 [102], mapping large values into $[-10, 10]$. Target labels are Gaussian distributions centered on the compressed return with width $\sigma_{\text{HL}} = 0.75$:

$$p_i^{\text{target}} \propto \exp\left(-\frac{(c_i - \text{symlog}(G_t))^2}{2\sigma^2_{\text{HL}}}\right), \tag{9.6}$$

where c_i is the i-th bin center. The loss is the cross-entropy between predicted logits and these soft targets. At inference, the scalar value is recovered as

$$\hat{V}(s) = \text{symexp}\left(\sum_i p_i\cdot c_i\right). \tag{9.7}$$

This formulation bounds gradient magnitudes (cross-entropy is bounded, unlike MSE), naturally handles multimodal return distributions, and is more robust to non-stationary targets. The bin count and symlog range are kept at their published defaults rather than tuned to the MicroRTS return scale, a choice revisited when the ablation evaluates this feature in Chapter 10.

Partial Advantage Estimator. When PPO collects fixed-length trajectory segments of T steps that do not reach a terminal state, GAE estimates near the segment's end suffer from high bias: fewer future TD residuals and an inaccurate bootstrap value $V(s_T)$. The Partial Advantage Estimator (PAE) [103] addresses this by zeroing advantages for the last $(1-\kappa)\cdot T$ steps of incomplete segments, so only the first $\lfloor\kappa\cdot T\rfloor$ steps contribute to the policy gradient (Figure 9.4). Segments containing a terminal state are used in full, since the terminal reward eliminates bootstrap bias. The value function is still trained on all steps, since even slightly biased return targets remain informative for regression. This matters in MicroRTS, where episodes last 3 000 to 8 000 steps but rollouts collect only $T = 512$, so most segments are incomplete. Despite using fewer transitions per update, PAE is reported to improve win rates in MicroRTS in its originating work [103]; the ablation of Section 10.3 re-examines this on the present setup ($\kappa = 0.95$).

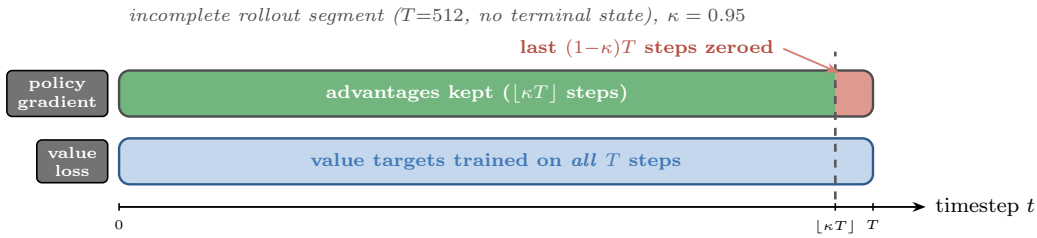


Figure 9.4: PAE on an incomplete rollout segment ($T = 512$, $\kappa = 0.95$). Advantages are zeroed for the last $(1-\kappa)T$ steps where the inaccurate bootstrap $V(s_T)$ would propagate bias through GAE; only the first $\lfloor \kappa T \rfloor$ steps drive the policy gradient (green). The value loss uses all T steps (blue), since even biased return targets remain informative for regression.

9.4 Opponent Curriculum and Self-Play

Training against a fixed scripted opponent leads to overfitting to a single strategy. By default, bot environments are distributed across a fixed diverse pool, emphasizing two strong opponents (CoacAI and Mayari) while retaining exposure to simpler strategies (RandomBiasedAI, WorkerRush, LightRush). This fixed mix is the baseline of Chapter 10. Three mechanisms layer on top of it. *Adaptive scheduling* replaces the fixed mix with a difficulty-progression curriculum; *historical self-play with PFSP* adds past agent versions as opponents; *Matchup Competitiveness Weighting* reweights transitions according to opponent difficulty within the PPO update itself. The last two are orthogonal to the bot pool and combine with either the fixed mix or the adaptive curriculum.

Adaptive opponent scheduling. When adaptive scheduling is enabled, all bot environments start against RandomBiasedAI. An `AdaptiveOpponentScheduler` tracks per-opponent win rates over a sliding window (default: 50 games) and escalates the difficulty (Figure 9.5). When the agent surpasses a threshold win rate (default: 70%) against the currently trained bot, the matching environments are promoted to the next tier. A minimum of two environments per tier is kept to prevent forgetting earlier strategies. The interaction with self-play is gated. While the terminal tier is not yet mastered, the self-play environments remain dormant: their transitions are filtered from the PPO update and their results ignored by every scheduler. When all terminal-tier opponents reach a configurable mastery threshold (default: 90%), the dormant environments switch to PFSP-hard sampling against historical checkpoints while the adaptive bot curriculum continues in parallel. The result is a fully automated curriculum, from a single easy bot through scripted-bot mastery to self-play, without manual intervention. When enabled, this adaptive curriculum replaces the fixed baseline mix entirely; the two are mutually exclusive.

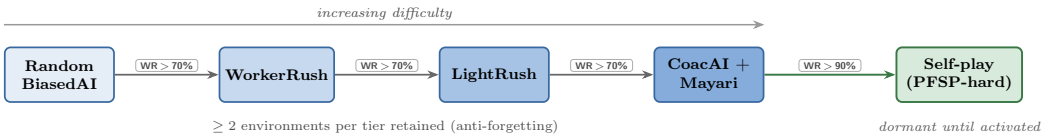


Figure 9.5: Automated opponent curriculum. Each bot environment is promoted to the next tier once its sliding-window win rate exceeds 70%, with at least two environments per tier kept to prevent forgetting. Self-play stays dormant until every terminal-tier opponent is mastered, then switches to PFSP-hard sampling against historical checkpoints.

Historical self-play and PFSP. Once scripted opponents are mastered, training against them yields diminishing returns. Self-play [104] exposes the agent to an opponent that adapts alongside it. A `CheckpointPool` stores every saved checkpoint and, at each refresh, builds a candidate set from all *recent* checkpoints (a First-In First-Out (FIFO) buffer of the last k) plus a uniform subsample of *historical* ones, balancing recency with diversity. A subset of environments plays against a checkpoint sampled from this set, refreshed every M updates. Pool sampling follows three modes inspired by PFSP [8]: `uniform` (random selection), `hard` ($P \propto (1-\mathrm{WR})^p$, prioritizing opponents the agent loses to), and `var` ($P \propto \mathrm{WR} \cdot (1-\mathrm{WR})$, prioritizing uncertain 50/50 matchups). In the feature ablation of Section 10.3, self-play uses `hard` mode with the default exponent $p = 1.0$ (reducing to $P \propto 1-\mathrm{WR}$); the sampled opponent is refreshed every $M = 50$ updates (≈600 k environment steps), drawn from a pool split evenly between the recent and historical buffers.

Matchup Competitiveness Weighting. Matchup Competitiveness Weighting (MCW) is a novel transition-level reweighting scheme operating within the PPO minibatch update. PFSP samples opponents by win-rate proximity [8], and Prioritized Experience Replay (PER) weights transitions off-policy by TD-error [105]. MCW combines both ideas but operates on-policy and at transition level, preserving the rollout distribution. The distinction from PFSP variants is deliberate: a competitiveness-weighted PFSP scheme ($P \propto \mathrm{WR}(1-\mathrm{WR})$) reweights the probability of sampling an opponent and thereby changes which data is collected, whereas MCW leaves the rollout distribution untouched and reweights only each transition's gradient contribution inside the update. The two act on orthogonal axes, data collection vs gradient weighting, and could in principle be combined. An `OpponentTracker` maintains per-opponent win rates over a sliding window (default: 100 games), and each transition receives weight

$$w_k = 1 - |\mathrm{WR}_k - 0.5|, \tag{9.8}$$

where $\mathrm{WR}_k \in [0,1]$ is the sliding-window win rate against opponent k. The weight peaks at $\mathrm{WR}_k = 0.5$ (value 1.0) and decays linearly to 0.5 at $\mathrm{WR}_k \in \{0,1\}$, a $2:1$ ratio between fully competitive and fully decided matchups (Figure 9.6). Weights are then renormalized so that $\bar{w} = 1$ across the rollout, preserving the gradient's overall magnitude.

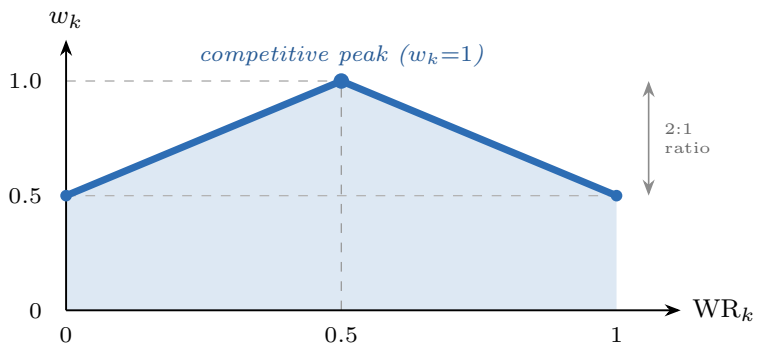


Figure 9.6: MCW transition weight $w_k = 1 - |\mathrm{WR}_k - 0.5|$ as a function of the sliding-window win rate against opponent k. The weight peaks at 1.0 for even matchups and decays to 0.5 for fully decided ones (a $2:1$ ratio), then is renormalized to mean 1 over the rollout.

MCW reweights *only the policy-gradient term* of the PPO objective; the entropy bonus and the value

losses of all critic heads (shaped, sparse, cost) remain uniform across transitions:

$$\mathcal{L} = \mathbb{E}_t\left[\, w_t\, \mathcal{L}^{\text{CLIP}}(t)\right] - \beta\, \mathbb{E}_t[\mathcal{H}(\pi_t)] + \sum_{h\in\{\text{sh,sp,co}\}} c_{\text{v},h}\, \mathbb{E}_t[\mathcal{L}^{\text{VF},h}(t)], \tag{9.9}$$

where w_t is the MCW weight assigned to transition t, $\mathcal{L}^{\text{CLIP}}$ the standard PPO clipped surrogate (Equation 3.16), $\mathcal{H}(\pi_t)$ the policy entropy with coefficient β (Equation 3.20), and $\mathcal{L}^{\text{VF},h}$ the value loss of critic head $h \in \{\text{shaped}, \text{sparse}, \text{cost}\}$ weighted by $c_{\text{v},h}$ (Equation 3.19). Only the surrogate term carries the per-transition weight w_t; all other terms aggregate uniformly over the batch. This decoupling is intentional: the value heads must produce unbiased estimates across the full opponent distribution for the GAE advantages to remain well calibrated, while the policy update concentrates its capacity on the most informative signal (near-50% matchups). The triangular weight itself is a design choice rather than a derived optimum: any profile that peaks at even matchups and decays toward decided ones would serve the same role, and tuning its exact shape is left to future work.

9.5 Auxiliary Representation Learning

The PPO reward signal alone may not push the encoder to capture all strategically relevant features. Four auxiliary heads share the encoder with the actor and critic, training jointly with the PPO objective to add gradient signal that shapes internal representations. Table 9.4 summarizes the four heads; architectural details and strategic rationale follow.

Table 9.4: Auxiliary prediction heads sharing the UECD encoder.

Head	Architecture	Target	Loss
Spatial ownership	1×1 conv	Per-cell P0 / P1 ownership map	Binary cross-entropy (per cell)
Unit count	AdaptiveAvgPool → MLP	12-d vector (6 types × 2 players)	MSE
Temporal contrastive	Pool → MLP → 128-d ℓ_2 embedding	Same vs other timesteps in batch	InfoNCE (Information Noise-Contrastive Estimation) [106]
Opponent action modeling	Conv head, per enemy cell	Enemy action type (6 classes)	Focal cross-entropy ($\gamma_f = 2$) [107]

(i) *Spatial ownership* forces the encoder to explicitly represent territorial control, a key strategic concept in RTS games. Targets are downsampled when the encoder resolution differs from the observation resolution. ***(ii)*** *Unit count* maintains a global summary of army composition, information otherwise distributed across spatial positions and essential for production and engagement decisions. ***(iii)*** *Temporal contrastive learning* pulls embeddings of consecutive timesteps t and $t{+}1$ together (positive pair) while pushing them away from other environments in the batch (negatives). In MicroRTS, consecutive states are highly correlated (units move one cell per action), so capturing this continuity helps the encoder distinguish meaningful state changes from noise. ***(iv)*** *Opponent action modeling* predicts each enemy unit's next action type (None, Move, Harvest, Return, Produce, Attack); the focal loss downweights easy predictions (e.g., idle units) and focuses on informative ones (e.g., attacking or producing units), encouraging the encoder to reason about opponent intentions.

The four heads differ in convergence behavior. Spatial ownership and unit count can in principle reach zero loss, since the agent observes the full game state and the targets are deterministic functions of the observation. Opponent action modeling and temporal contrastive learning are inherently noisy: the opponent's actions depend on its internal policy (unobservable), and consecutive states may differ unpredictably. For the latter two, the goal is not zero loss but a useful gradient signal that shapes the encoder representations throughout training.

The total loss combines the PPO objective with the per-task auxiliary losses:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{PPO}} + \sum_i c_i \cdot \mathcal{L}_{\text{aux},i}, \tag{9.10}$$

with per-task coefficients c_i (defaults: 0.1 for spatial ownership and unit count, 0.05 for temporal contrastive and opponent action modeling).

9.6 Generalization

Two mechanisms target generalization at different levels: *symmetry augmentation* exploits within-map invariances, while *multi-map training* exposes the policy to different maps in one run.

Symmetry augmentation. Most MicroRTS maps are spatially symmetric, yet a CNN trained on a fixed orientation may learn position-dependent biases (e.g., always expanding leftward). The `SymmetryAugmentation` wrapper (Section 7.6) assigns each environment one of four flip modes at episode start (identity, horizontal flip, vertical flip, or 180° rotation), consistently transforming observations, action masks, and the 7×7 attack target grid. Applied only during training, the augmentation quadruples positional diversity without extra rollouts. It is most natural on symmetric maps, where each flipped observation corresponds to an equivalent game state; on asymmetric maps (e.g., *BWDistantResources32x32*, *BloodBath.scmB*), it changes the orientation distribution and should be interpreted as data augmentation rather than an exact game symmetry.

Multi-map training and Prioritized Level Replay. The padded environment of Section 7.5.2 enables training across maps from 8×8 to 64×64 within a single run. Two scheduling strategies are available. *Cyclic rotation* rotates maps every M updates (default: 50) in round-robin order. *Prioritized Level Replay* (PLR) [108] samples the next map proportionally to learning difficulty, $P(\text{map}_i) \propto (1 - \text{WR}_i) + \epsilon$ (with $\epsilon = 0.1$), allocating more training time to maps where the agent currently struggles.

9.7 Architectural Refinements

The two final modifications operate on the activation function and the critic's pooling layer respectively. They affect the optimization landscape and value estimation rather than the network's representational capacity, so they are evaluated alongside the training features in Chapter 10 rather than the architecture ablation.

GELU activation. All ReLU activations in the encoder and decoder can be replaced with GELU (Gaussian Error Linear Unit) [109]:

$$\text{GELU}(x) = x \cdot \Phi(x), \tag{9.11}$$

where Φ is the standard Gaussian cumulative distribution function. Unlike ReLU, GELU provides a smooth, non-zero gradient for negative inputs. In the UECD architecture, which stacks 14 CBAM-ResBlocks, smooth gradients can reduce dead neurons and improve convergence.

SPP critic. The default critic uses adaptive average pooling to reduce the encoder output to a fixed-size vector. The Spatial Pyramid Pooling (SPP) [110] variant replaces this with multi-scale pooling: the feature map is pooled at three resolutions (1×1, 2×2, 4×4) and the results concatenated, providing global and local spatial information regardless of map size. This matters for multi-map training (Section 9.6), where a single critic must evaluate positions on maps from 8×8 to 64×64 [80].

Experimental Results and Analysis

10

This chapter presents the empirical evaluation of the contributions developed in Chapters 6 to 9. It addresses a single question in stages: *what is the strongest MicroRTS agent attainable under a realistic compute budget, and how does it perform against the established competition?*
The experiments build on one another. The network architecture is examined first (Section 10.2), followed by an ablation of each training feature (Section 10.3). The winning components of both studies feed into a full-scale single-map agent (Section 10.4), which is then evaluated against the full competition pool (Section 10.5) and probed for robustness under distribution shift (Section 10.6). Two further experiments follow: Section 10.7 reports a standalone behavior-cloning warmstart study (BC-PPO), run separately from the main agent as a compute-efficiency comparison, and Section 10.8 builds a multi-map agent and contrasts it with the single-map specialist across the open layouts of size at most 16×16.

10.1 Experimental Setup

All experiments share the same PPO training stack, built on the MicroRTS environment described in Chapter 7. The baseline training configuration appears in Table 10.1, and the corresponding hyperparameters in Table 9.1; together they serve as the reference for every subsequent experiment, and any deviation is stated explicitly. Evaluations use a fixed pool of scripted opponents with stochastic action sampling, under two protocols. Final checkpoint evaluations report win rates averaged over 1 000 games per opponent (500 as P0, 500 as P1), giving balanced position coverage and statistically reliable estimates. In-training evaluations, run periodically during training to track learning dynamics, use a lighter protocol of 10 games per opponent (5 as P0, 5 as P1).
Several experiments ran concurrently on the high-performance computing (HPC) cluster. In particular, the feature ablation (Section 10.3) overlapped with parts of the single-map training (Section 10.4), so the latter could not retroactively incorporate every ablation finding. Sections are therefore ordered by logical dependency rather than chronological execution.

Table 10.1: Baseline experiment setup used throughout Chapter 10 unless otherwise specified. PPO hyperparameters appear in Table 9.1.

Parameter	Value
Game environment	
Map	*basesWorkers16x16A* (16×16)
Max game length	4 000 steps
Training opposition	
Environment setup	24 bot envs, diverse opponents, alternate players
Opponent distribution	9 CoacAI, 9 Mayari, 2 RandomBiasedAI, 2 WorkerRush, 2 LightRush
Reward signal	
Reward weights	Default from Table 7.4
Value head	shaped only
Hardware	
GPU	RTX 6000 Ada (48 GB)

Reproducibility and ethics. All multi-seed experiments use the fixed seed set 1, 2, 3 (the per-seed columns of the ablation tables correspond to these values); single-seed runs use seed 1. Every run executes on a single RTX 6000 Ada GPU (48 GB) and saves a self-contained *config.json* for exact reconstruction, alongside the open-sourced codebase. The work involves no human subjects and no personal data; its only resource cost is compute, reported throughout in GPU-days.

10.2 Architecture Ablation Analysis

To isolate the effect of the network architecture, each candidate described in Table 8.1 is trained under identical hyperparameters for 100 M environment steps on the same map. No advanced features are enabled, so any performance difference reflects architecture capacity alone. The full sweep (7 architectures × 3 seeds at 100 M steps, 21 runs) consumed 13.6 GPU-days.
Table 10.2 summarizes the mean win rate across three seeds for each architecture, with per-seed matchup breakdowns in Tables 10.3a, 10.3b, and 10.3c.

Table 10.2: Architecture ablation summary across three seeds: mean overall win rate (%) ± standard deviation, and mean improvement over baseline ($\bar{\Delta}$). Architectures sorted by mean overall win rate.

Architecture	Params	SPS	Seed 1	Seed 2	Seed 3	Mean ± Std	$\bar{\Delta}$ (pp)
GridNet	838 k	2 688	61.9	70.2	62.4	64.8 ± 3.8	N/A
IMPALA-CNN	1.02 M	2 416	73.4	60.5	72.1	68.7 ± 5.8	+3.9
IMPALA-CNN-Entity	1.37 M	1 903	75.8	73.2	75.8	74.9 ± 1.2	+10.1
U-Net	2.59 M	2 121	77.9	78.4	74.8	77.0 ± 1.6	+12.2
U-Net-Entity-CBAM	2.90 M	1 289	88.1	70.5	78.8	79.1 ± 7.2	+14.3
U-Net-Entity	2.90 M	1 518	78.5	80.2	89.3	82.7 ± 4.7	+17.9
U-Net-Entity-CBAM-Deep	4.72 M	1 176	93.4	88.8	76.6	**86.3 ± 7.1**	**+21.5**

Cross-seed variance is substantial, up to ±7.2 for the larger models. With only three seeds, several of the gaps below sit within the noise floor and cannot be resolved at this budget; the discussion therefore focuses on means and treats differences smaller than the seed-to-seed spread as *inconclusive*. Rather than walking down the ranking row by row, the results are read through the three reasoning axes of Section 8.1.2 (local spatial, long-range relational, and global spatial), asking for each how much it contributes and how reliably.

10.2.1 Axis 1: Local Spatial Reasoning

The three convolutional refinements of Section 8.2 target the local axis, the backbone onto which the relational and global modules are later added. Together they account for the largest share of the gain over `GridNet`, though their individual contributions vary in magnitude and reliability.

Residual blocks (`IMPALA-CNN`, +3.9 pp). Replacing `GridNet`'s plain convolutions with `IMPALA-CNN` residual blocks shifts the mean from 64.8 to 68.7. The +3.9 pp gap overlaps with both seed-to-seed spreads (±3.8, ±5.8), leaving the comparison inconclusive at this budget. Seed 1 hints that better gradient flow sharpens rush defense (WorkerRush 27.6 / 25.0 → 99.2 / 79.8) without affecting relational reasoning (CoacAI stays erratic). The value of residual blocks lies less in this modest gain than in enabling the deeper backbones below without unstable gradients.

`U-Net` skips and SE attention (+12.2 pp over `GridNet`). Adding skip connections between encoder and decoder, plus SE channel attention, yields the largest local-axis jump (64.8 → 77.0). The

Table 10.3: Architecture ablation per-seed: 1000-game win rates. Each cell shows P0 / P1 win rates (%). Overall = mean across opponents and positions. $\bar{\Delta}$ = improvement over `GridNet`.

(a) Seed 1

Architecture	Random	WorkerRush	LightRush	CoacAI	Mayari	Overall	$\bar{\Delta}$ (pp)
GridNet	100.0 / 100.0	27.6 / 25.0	63.8 / 99.4	13.2 / 98.4	56.6 / 35.0	61.9%	N/A
IMPALA-CNN	100.0 / 100.0	99.2 / 79.8	98.6 / 67.6	18.8 / 30.6	74.6 / 64.4	73.4%	+11.5
IMPALA-CNN-Entity	100.0 / 100.0	92.6 / 40.4	85.4 / 77.0	12.2 / 80.2	79.8 / 90.4	75.8%	+13.9
U-Net	100.0 / 100.0	81.0 / 80.2	82.0 / 99.4	18.0 / 96.4	79.6 / 42.6	77.9%	+16.0
U-Net-Entity	100.0 / 100.0	35.2 / 99.6	63.0 / 49.8	72.4 / 73.4	98.0 / 94.0	78.5%	+16.6
U-Net-Entity-CBAM	100.0 / 100.0	71.0 / 70.4	97.4 / 95.2	64.2 / 96.2	94.6 / 92.4	88.1%	+26.2
U-Net-Entity-CBAM-Deep	100.0 / 100.0	97.8 / 98.6	99.6 / 99.8	54.6 / 93.4	94.8 / 95.0	**93.4%**	**+31.5**

(b) Seed 2

Architecture	Random	WorkerRush	LightRush	CoacAI	Mayari	Overall	$\bar{\Delta}$ (pp)
IMPALA-CNN	100.0 / 100.0	56.6 / 10.4	86.8 / 75.4	20.8 / 38.6	70.0 / 46.2	60.5%	−9.7
GridNet	100.0 / 100.0	57.2 / 40.0	84.6 / 99.8	6.0 / 97.6	76.0 / 41.0	70.2%	N/A
U-Net-Entity-CBAM	100.0 / 100.0	97.8 / 74.4	74.2 / 96.8	0.6 / 27.6	55.2 / 78.4	70.5%	+0.3
IMPALA-CNN-Entity	100.0 / 100.0	93.2 / 35.0	82.8 / 95.4	0.8 / 96.8	73.6 / 54.4	73.2%	+3.0
U-Net	100.0 / 100.0	85.4 / 97.0	91.6 / 96.4	15.4 / 65.4	76.0 / 56.8	78.4%	+8.2
U-Net-Entity	100.0 / 100.0	97.4 / 99.8	96.8 / 100.0	0.4 / 98.0	90.4 / 19.4	80.2%	+10.0
U-Net-Entity-CBAM-Deep	100.0 / 100.0	95.0 / 99.4	97.4 / 100.0	23.4 / 93.4	86.8 / 92.6	**88.8%**	**+18.6**

(c) Seed 3

Architecture	Random	WorkerRush	LightRush	CoacAI	Mayari	Overall	$\bar{\Delta}$ (pp)
GridNet	100.0 / 100.0	43.2 / 31.8	68.8 / 96.4	2.2 / 83.8	71.8 / 26.2	62.4%	N/A
IMPALA-CNN	100.0 / 100.0	83.2 / 36.4	87.0 / 94.8	5.8 / 92.6	83.8 / 37.6	72.1%	+9.7
U-Net	100.0 / 100.0	92.8 / 71.4	89.0 / 99.8	0.8 / 98.4	70.4 / 25.6	74.8%	+12.4
IMPALA-CNN-Entity	100.0 / 100.0	98.8 / 41.0	69.8 / 99.4	11.6 / 98.8	76.0 / 62.4	75.8%	+13.4
U-Net-Entity-CBAM-Deep	100.0 / 100.0	99.6 / 93.6	95.8 / 98.8	0.0 / 31.6	72.8 / 73.6	76.6%	+14.2
U-Net-Entity-CBAM	100.0 / 100.0	93.6 / 48.0	96.2 / 99.8	0.2 / 98.0	85.6 / 66.8	78.8%	+16.4
U-Net-Entity	100.0 / 100.0	99.8 / 97.4	92.4 / 87.0	42.2 / 85.2	98.6 / 90.8	**89.3%**	**+26.9**

+12.2 pp gain is several times the seed-to-seed spread and one of the most consistent of the sweep (±1.6), so this contribution is robust. Preserving full-resolution detail through the bottleneck, rather than rebuilding it from a compressed code, repairs most of the P0/P1 asymmetry seen in skip-less architectures (seed 1: CoacAI P1 30.6 → 96.4, LightRush P1 67.6 → 99.4).
The residual asymmetry against CoacAI is non-architectural: replays show CoacAI placing its barracks at the bottom regardless of starting position, giving the agent a structural advantage as P1 (bottom spawn). The opponent's long-horizon economic strategies further demand counter-plans that no architecture reliably learns within the 100 M-step budget, even with CoacAI in the opponent pool: a budget or curriculum problem rather than an architectural one.

CBAM spatial attention. CBAM adds a spatial gate on top of channel attention, selecting *where* to attend, not just *which* features matter. Its isolated effect is inconclusive at this scale: `U-Net-Entity` → `U-Net-Entity-CBAM` shifts the mean by −3.6 pp with the highest variance of the sweep (±7.2). Seed 1 is promising (+9.6 pp, removing the LightRush asymmetry 63.0 / 49.8 → 97.4 / 95.2), but seeds 2 and 3 fall back below `U-Net-Entity`. The gate seems constrained by limited capacity at this depth, with a clearer benefit emerging only once the network is deepened (Axis 3 below).

10.2.2 Axis 2: Long-Range Relational Reasoning

The Entity Transformer (Section 8.3) is the most reliable contribution in the sweep, adding value on top of *either* backbone with consistent gains on the long-horizon, coordination-heavy opponents (CoacAI, Mayari) that the local axis alone does not solve. Relational reasoning therefore appears to complement rather than substitute for spatial capacity.

On the `IMPALA-CNN` backbone (`IMPALA-CNN-Entity`, +6.2 pp). Adding the Transformer over extracted units to `IMPALA-CNN` lifts the mean from 68.7 → 74.9. The +6.2 pp gain exceeds both seed-to-seed spreads, and `IMPALA-CNN-Entity` is the most consistent architecture of the sweep (±1.2), making this contribution the most defensible of the study. All three seeds gain on the strategic bots (seed 1: CoacAI P1 30.6 → 80.2, Mayari 74.6 / 64.4 → 79.8 / 90.4), at the cost of throughput (2 416 → 1 903 SPS) and some rush defense (seed 1: WorkerRush P1 79.8 → 40.4): the shared encoder reallocates capacity from spatial to relational features.

On the `U-Net` backbone (`U-Net-Entity`, +5.7 pp). On `U-Net`, the same module yields a comparable +5.7 pp gain (77.0 → 82.7), the second-best mean of the sweep, though higher variance (±4.7) brings the gap closer to the noise floor than on `IMPALA-CNN`. The spatial / relational tension persists: seed 1 gains Mayari +18.4 / + 51.4 but loses WorkerRush P0 (81.0 → 35.2).

This gain's reliability is structural, not a function of added capacity. Convolutions propagate interactions cell-by-cell through stacked receptive fields: two units far apart influence each other only after the signal traverses several layers of pooling and convolution. But in RTS, strategically meaningful interactions are entity-level: a worker and its base must coordinate as units, not spatial neighborhoods; a defender and the rusher it intercepts must reason about each other regardless of grid distance. Self-attention over entities bypasses spatial propagation: any two units are one attention hop apart. Hence a modest module (≈ 0.3 M extra parameters) delivers a more reliable gain than far larger convolutional additions: the contribution is mechanistic, addressing reasoning convolutional capacity provides only at much higher parameter cost. More convolutional depth or width does not replace it, only approximates it inefficiently.

10.2.3 Axis 3: Global Spatial Reasoning

The final step adds the two global-axis mechanisms of Section 8.4: bottleneck self-attention and a pyramidal depth profile $(2, 3, 4, 3, 2)$ that concentrates capacity where distant regions are summarized. Stacked onto `U-Net-Entity-CBAM`, they yield `U-Net-Entity-CBAM-Deep`, the top configuration by mean (79.1 → 86.3, +7.2 pp, best seed 93.4%). The +7.2 pp gap, however, sits at roughly the ±7.1 spread of the new architecture, so the improvement over `U-Net-Entity-CBAM` is suggestive rather than decisive. Two effects appear to combine. Depth supplies the capacity that the Axis 1 spatial gate lacked on its own, making CBAM effective in this deeper context. Global self-attention adds a complementary path, letting opposite ends of the map interact directly rather than through stacked receptive fields. Variance nonetheless stays high (±7.1; seed 3 CoacAI collapses to 0.0 / 31.6): the added capacity sharpens the ceiling without removing the strategic blind spots.

10.2.4 Final Architecture Choice

Despite the high seed variance, `U-Net-Entity-CBAM-Deep` remains the most defensible choice for subsequent experiments. It covers all three reasoning axes, and the ablation attributes a non-negative contribution to each axis in the deep configuration, though CBAM's effect in isolation is inconclusive. It achieves the highest mean win rate (86.3%) and the highest single-seed result (93.4%). The variance (±7.1) likely reflects training stochasticity at 100 M steps rather than an architectural

limitation, though the 3-seed budget cannot rigorously establish this. The throughput cost (1 176 vs 2 688 SPS for `GridNet`) is acceptable given the +21.5 pp improvement. The remainder of this chapter refers to this architecture as UECD.

10.3 Feature Ablation Analysis

Each feature is tested in isolation on top of the UECD baseline (Table 10.1) for 50 M environment steps. This shorter budget keeps the total ablation compute within the available cluster allocation. Three independent seeds are run per configuration, providing a minimal estimate of run-to-run variance and a means of flagging features whose effect is dominated by noise. Each ablation row enables exactly one feature, or a tightly coupled pair that cannot be cleanly separated (e.g., *filtered masks + reserved observations*). All other elements remain identical to the baseline: the network architecture, the optimization hyperparameters, the curriculum, and the evaluation protocol. The complete ablation consumed approximately 19.4 GPU-days of training compute.

Table 10.4 summarizes the mean win rate across the three seeds for each feature, with per-seed matchup breakdowns in Tables 10.5a, 10.5b, and 10.5c. Features are sorted by overall mean, and the baseline row separates features that help from those that hurt.

Seed variance is wide enough that the table cannot be read as a clean ranking. The baseline alone carries a ±7.5 standard deviation, and several features show ±10 to ±20 spreads, large enough that the gap between neighboring rows is usually smaller than the noise on each row.

Table 10.4: Feature ablation summary across three seeds: mean overall win rate (%) ± standard deviation, mean pp improvement over baseline ($\bar{\Delta}$), and additional parameters relative to the baseline.

Feature	**Add. Params**	**Seed 1**	**Seed 2**	**Seed 3**	**Mean ± Std**	**$\bar{\Delta}$ (pp)**
Extended Obs (73ch)	+25.8 k	83.6	86.7	82.7	84.3 ± 1.7	+26.3
Filt. Masks + Res. Obs	+560	85.9	83.1	72.8	80.6 ± 5.6	+22.6
MCW	+0	77.8	78.7	78.6	78.4 ± 0.4	+20.4
Opponent Modeling	+10.5 k	72.4	78.9	82.1	77.8 ± 4.0	+19.8
Triple Value Heads	+137.7 k	77.6	72.6	67.9	72.7 ± 4.0	+14.7
PAE (keep=95%)	+0	67.6	63.8	82.4	71.3 ± 8.0	+13.3
Hierarchical Mask	+0	59.9	66.4	86.8	71.1 ± 11.5	+13.1
Adaptive Opponents	+0	67.9	69.6	71.4	69.6 ± 1.4	+11.7
PopArt	+0	65.5	59.7	80.4	68.5 ± 8.7	+10.6
Aux Unit Count	+3.9 k	83.6	66.2	52.9	67.6 ± 12.6	+9.6
Bots + Self-Play + PFSP	+0	58.9	79.6	62.2	66.9 ± 9.1	+8.9
GELU	+0	69.6	74.3	55.0	66.3 ± 8.2	+8.3
Frame Stack (4)	+45.4 k	71.4	64.3	60.4	65.3 ± 4.5	+7.3
Autoregressive + Hier. Mask	+28.2 k	59.1	60.4	75.9	65.1 ± 7.6	+7.2
Autoregressive	+28.2 k	70.7	53.2	67.3	63.7 ± 7.6	+5.8
Aux Spatial	+1.2 k	56.1	68.3	65.4	63.3 ± 5.2	+5.3
SPP Critic	+81.9 k	74.8	58.6	56.2	63.2 ± 8.3	+5.2
Aux Contrastive	+22.8 k	70.6	54.8	59.5	61.6 ± 6.6	+3.7
Build-Time Rewards	+0	46.9	62.3	74.3	61.2 ± 11.2	+3.2
Baseline	N/A	50.4	55.3	68.2	58.0 ± 7.5	N/A
Augment Symmetry	+0	62.6	56.9	52.9	57.5 ± 4.0	−0.5
HL-Gauss	+16.5 k	62.2	21.2	35.0	39.5 ± 17.0	−18.5

Table 10.5: Feature ablation per-seed: 1 000-game win rates. Each cell shows P0 / P1 win rates (%). Overall = mean across opponents and positions. Δ = pp improvement over baseline.

(a) Seed 1

Feature	Random	WorkerRush	LightRush	CoacAI	Mayari	Overall	Δ (pp)
Filt. Masks + Res. Obs	100.0 / 100.0	67.2 / 88.8	99.4 / 66.0	59.4 / 89.0	89.4 / 99.6	85.9%	+35.5
Extended Obs (73ch)	100.0 / 100.0	67.6 / 75.2	99.2 / 100.0	22.8 / 81.2	95.6 / 94.6	83.6%	+33.2
Aux Unit Count	100.0 / 100.0	82.4 / 96.2	92.8 / 99.0	30.6 / 76.6	78.6 / 80.2	83.6%	+33.2
MCW	100.0 / 100.0	71.2 / 92.2	84.0 / 99.0	21.2 / 84.8	67.8 / 57.6	77.8%	+27.4
Triple Value Heads	100.0 / 100.0	41.8 / 76.2	98.4 / 98.4	28.4 / 80.6	94.0 / 58.0	77.6%	+27.2
SPP Critic	100.0 / 100.0	43.8 / 46.4	99.4 / 97.2	24.0 / 85.6	80.2 / 71.8	74.8%	+24.4
Opponent Modeling	100.0 / 100.0	98.2 / 37.4	96.2 / 88.0	13.2 / 60.6	79.6 / 50.4	72.4%	+22.0
Frame Stack (4)	100.0 / 100.0	97.8 / 55.6	90.6 / 93.8	0.0 / 53.8	70.8 / 51.2	71.4%	+21.0
Autoregressive	100.0 / 100.0	62.6 / 59.0	81.2 / 82.0	17.0 / 67.8	82.0 / 55.6	70.7%	+20.3
Aux Contrastive	100.0 / 100.0	81.8 / 39.0	59.8 / 45.6	49.8 / 64.0	88.4 / 77.4	70.6%	+20.2
GELU	100.0 / 100.0	46.4 / 31.8	93.4 / 97.6	40.4 / 85.2	73.6 / 28.0	69.6%	+19.2
Adaptive Opponents	100.0 / 100.0	63.4 / 55.8	85.8 / 90.8	10.8 / 70.6	59.8 / 41.8	67.9%	+17.5
PAE (keep=95%)	100.0 / 100.0	26.8 / 92.2	10.4 / 97.4	4.0 / 91.0	69.2 / 84.6	67.6%	+17.2
PopArt	100.0 / 100.0	70.8 / 62.0	53.0 / 99.4	0.0 / 77.0	33.4 / 59.2	65.5%	+15.1
Augment Symmetry	100.0 / 100.0	31.2 / 22.2	46.0 / 84.8	23.4 / 79.6	79.2 / 59.2	62.6%	+12.2
HL-Gauss	100.0 / 100.0	44.6 / 48.8	49.4 / 27.8	35.0 / 44.6	90.2 / 81.4	62.2%	+11.8
Hierarchical Mask	100.0 / 100.0	69.2 / 14.8	90.4 / 96.4	0.2 / 4.6	64.4 / 59.2	59.9%	+9.5
Autoregressive + Hier. Mask	100.0 / 100.0	83.8 / 24.0	78.2 / 55.4	8.0 / 27.4	66.0 / 48.2	59.1%	+8.7
Bots + Self-Play + PFSP	100.0 / 100.0	43.8 / 36.2	84.0 / 80.6	6.6 / 28.0	64.6 / 44.8	58.9%	+8.5
Aux Spatial	100.0 / 100.0	66.2 / 15.4	70.0 / 76.4	13.2 / 21.6	56.2 / 42.0	56.1%	+5.7
Baseline	100.0 / 100.0	42.4 / 23.4	96.8 / 51.2	0.0 / 0.0	75.4 / 14.6	50.4%	N/A
Build-Time Rewards	100.0 / 100.0	24.0 / 9.0	41.2 / 19.0	27.6 / 29.2	68.2 / 50.8	46.9%	−3.5

(b) Seed 2

Feature	Random	WorkerRush	LightRush	CoacAI	Mayari	Overall	Δ (pp)
Extended Obs (73ch)	100.0 / 100.0	100.0 / 71.0	100.0 / 100.0	45.8 / 79.2	84.0 / 87.4	86.7%	+31.4
Filt. Masks + Res. Obs	100.0 / 100.0	47.0 / 87.8	99.2 / 100.0	32.2 / 90.2	86.0 / 88.4	83.1%	+27.8
Bots + Self-Play + PFSP	100.0 / 100.0	90.2 / 98.0	62.6 / 51.6	30.8 / 75.8	94.4 / 92.8	79.6%	+24.3
Opponent Modeling	100.0 / 100.0	91.0 / 60.8	63.6 / 78.0	31.6 / 84.6	94.4 / 84.6	78.9%	+23.5
MCW	100.0 / 100.0	80.6 / 81.6	91.2 / 90.8	27.8 / 66.0	84.6 / 64.8	78.7%	+23.4
GELU	100.0 / 100.0	35.2 / 53.2	70.2 / 99.4	31.0 / 95.2	84.2 / 74.8	74.3%	+19.0
Triple Value Heads	100.0 / 100.0	90.4 / 40.8	92.2 / 86.2	39.8 / 39.0	88.0 / 49.4	72.6%	+17.3
Adaptive Opponents	100.0 / 100.0	74.6 / 66.6	91.8 / 79.8	20.2 / 53.8	65.8 / 43.6	69.6%	+14.3
Aux Spatial	100.0 / 100.0	92.2 / 43.8	98.4 / 81.6	19.6 / 37.0	82.6 / 27.6	68.3%	+13.0
Hierarchical Mask	100.0 / 100.0	91.8 / 43.4	84.0 / 91.6	0.2 / 13.8	66.4 / 73.0	66.4%	+11.1
Aux Unit Count	100.0 / 100.0	43.8 / 79.0	85.8 / 91.2	17.8 / 52.0	63.4 / 28.6	66.2%	+10.8
Frame Stack (4)	100.0 / 100.0	58.0 / 49.2	85.2 / 96.2	0.0 / 55.2	76.2 / 22.6	64.3%	+8.9
PAE (keep=95%)	100.0 / 100.0	12.8 / 73.6	91.0 / 100.0	22.6 / 42.0	52.8 / 42.8	63.8%	+8.4
Build-Time Rewards	100.0 / 100.0	36.8 / 36.6	68.6 / 36.2	34.6 / 44.8	88.6 / 77.2	62.3%	+7.0
Autoregressive + Hier. Mask	99.8 / 100.0	34.2 / 48.4	53.2 / 94.2	9.8 / 78.2	57.2 / 29.4	60.4%	+5.1
PopArt	100.0 / 100.0	79.6 / 80.8	50.6 / 5.2	13.0 / 21.8	60.8 / 85.2	59.7%	+4.4
SPP Critic	100.0 / 100.0	34.6 / 31.4	32.8 / 47.0	26.2 / 59.6	80.4 / 74.4	58.6%	+3.3
Augment Symmetry	100.0 / 100.0	33.0 / 73.8	64.6 / 75.2	8.8 / 46.0	40.6 / 27.2	56.9%	+1.6
Baseline	100.0 / 100.0	33.6 / 59.8	84.6 / 83.6	2.8 / 1.0	46.6 / 41.2	55.3%	N/A
Aux Contrastive	100.0 / 100.0	39.0 / 87.6	37.8 / 78.0	1.0 / 14.6	45.0 / 45.4	54.8%	−0.5
Autoregressive	100.0 / 100.0	64.2 / 41.8	75.8 / 60.2	6.4 / 6.8	49.8 / 26.8	53.2%	−2.1
HL-Gauss	93.4 / 99.2	0.2 / 19.4	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	21.2%	−34.1

(c) Seed 3

Feature	Random	WorkerRush	LightRush	CoacAI	Mayari	Overall	Δ (pp)
Hierarchical Mask	100.0 / 99.8	69.0 / 55.8	72.6 / 99.0	82.8 / 99.6	96.2 / 93.4	86.8%	+18.6
Extended Obs (73ch)	100.0 / 100.0	40.0 / 96.4	99.2 / 99.8	31.6 / 71.0	96.4 / 92.4	82.7%	+14.4
PAE (keep=95%)	100.0 / 100.0	93.2 / 99.2	99.4 / 98.8	18.2 / 67.6	69.2 / 78.8	82.4%	+14.2
Opponent Modeling	100.0 / 100.0	98.6 / 90.2	74.8 / 41.8	57.8 / 72.2	96.0 / 89.6	82.1%	+13.9
PopArt	100.0 / 100.0	83.8 / 99.2	90.6 / 100.0	1.0 / 99.8	73.2 / 56.6	80.4%	+12.2
MCW	100.0 / 100.0	90.4 / 78.0	98.0 / 97.4	21.0 / 70.2	72.6 / 58.2	78.6%	+10.3
Autoregressive + Hier. Mask	100.0 / 100.0	75.6 / 49.8	92.0 / 89.0	11.6 / 88.4	85.8 / 66.6	75.9%	+7.7
Build-Time Rewards	100.0 / 100.0	63.4 / 36.8	69.4 / 90.4	45.4 / 82.4	90.0 / 65.4	74.3%	+6.1
Filt. Masks + Res. Obs	100.0 / 100.0	32.6 / 45.2	98.0 / 99.8	12.4 / 92.6	75.0 / 72.4	72.8%	+4.6
Adaptive Opponents	100.0 / 100.0	47.2 / 76.0	94.4 / 98.8	2.2 / 77.4	72.0 / 45.8	71.4%	+3.2
Baseline	100.0 / 100.0	80.8 / 68.2	34.2 / 82.0	20.6 / 72.8	93.2 / 30.6	68.2%	N/A
Triple Value Heads	100.0 / 100.0	60.2 / 49.8	63.8 / 95.4	14.6 / 66.0	82.0 / 47.2	67.9%	−0.3
Autoregressive	100.0 / 100.0	29.4 / 39.4	88.0 / 99.2	15.0 / 79.8	65.8 / 56.4	67.3%	−0.9
Aux Spatial	100.0 / 100.0	95.8 / 84.8	81.2 / 91.8	0.4 / 10.6	48.2 / 41.0	65.4%	−2.9
Bots + Self-Play + PFSP	100.0 / 100.0	25.4 / 54.8	86.0 / 96.4	0.2 / 38.6	72.0 / 49.0	62.2%	−6.0
Frame Stack (4)	100.0 / 100.0	15.4 / 88.8	76.0 / 94.4	2.6 / 18.6	56.2 / 51.8	60.4%	−7.9
Aux Contrastive	100.0 / 100.0	34.0 / 49.6	87.0 / 92.0	15.0 / 23.2	61.8 / 32.6	59.5%	−8.7
SPP Critic	100.0 / 100.0	90.0 / 35.6	79.8 / 62.2	0.0 / 10.4	54.2 / 29.6	56.2%	−12.1
GELU	100.0 / 100.0	36.4 / 45.6	96.2 / 70.4	6.8 / 1.6	58.6 / 34.2	55.0%	−13.3
Aux Unit Count	100.0 / 99.8	70.4 / 49.4	62.0 / 79.6	2.4 / 1.2	51.4 / 13.2	52.9%	−15.3
Augment Symmetry	100.0 / 100.0	41.4 / 50.4	81.2 / 63.8	3.6 / 1.6	53.4 / 33.8	52.9%	−15.3
HL-Gauss	70.0 / 100.0	0.0 / 44.2	0.8 / 92.4	0.0 / 0.0	0.2 / 42.6	35.0%	−33.2

Only one separation is unambiguous: the four largest gains stand clearly apart, with a 5 pp drop from opponent modeling (+19.8 pp) to the fifth feature (+14.7 pp) and low variance throughout. Two more (*Triple Value Heads* and *PAE*) sit just behind: the former dips back to baseline on seed 3, the latter shows the wider spread; both still belong with the leaders, not the continuum down to the two below-baseline failures. Means in that continuum are often sizeable, several still above +10 pp; the obstacle to confident ordering is not that features fail to help but that cross-seed variance dwarfs the differences.

One fact cuts across the whole table: the gains that hold up cost almost nothing. The four most dependable features add only 36.9 k parameters against UECD's 4.72 M backbone, whereas the three largest parameter additions of the sweep (*Triple Value Heads* +137.7 k, *SPP Critic* +81.9 k, *Frame Stack* +45.4 k) all sit outside the leading group, despite carrying up to nearly four times the combined parameter count of the dependable four. Capacity is not the lever that pays at this scale.

10.3.1 Reliable Gains: Perception and Signal Allocation

The four most dependable gains split into two mechanisms, and neither touches model capacity: three change what the policy reads from the environment (*extended observations*, *filtered masks*, and *opponent modeling*), and one (*MCW*) changes which matchups its gradient focuses on. None adds meaningful capacity, yet each is stable enough across seeds to build on without reservation.

Extended observations (+26.3 pp, ±1.7, Section 7.2) is the largest and most consistent gain of the sweep, and its near-absent variance is the real signal: a result this stable across three independent runs reflects a structural change in the information available to the policy rather than a fortunate optimization path. With unit actions, remaining ticks to completion, and global resource counts made directly readable, the policy reads the present state instead of reconstructing it from a frame history. This is also why frame stacking lands far down the table (discussed below): it pays four times the channel cost for a strictly weaker form of the same temporal signal.

Filtered masks + reserved observations (+22.6 pp, ±5.6, Section 7.3) is one of the cheapest features

of the sweep (+560 parameters) for one of its largest gains. Its higher variance is itself informative: the size of the gain depends on how often the baseline happens to collide on a given seed, but the direction is positive on all three. Pruning invalid destinations upstream removes an entire failure class, the silent cancellation of concurrent moves into the same cell, that the policy gradient is slow to unlearn on its own.

MCW (+20.4 pp, ±0.4, Section 9.4) is the most stable feature of the entire sweep. Re-weighting opponents by recent win rate concentrates each batch's gradient on the matchups still unsolved instead of spending it on already-mastered bots. Because the effect is a pure reallocation of where gradient is spent and does not depend on initialization, the variance is near zero: the mechanism behaves identically regardless of seed.

Opponent modeling (+19.8 pp, ±4.0, Section 9.5) is nominally an auxiliary head but behaves like a perception feature rather than a learning trick. Forcing the encoder to predict the opponent's next actions constrains its representation toward features useful for both acting and anticipating, and the gains concentrate on the strategic bots (CoacAI, Mayari), the same opponents the Entity Transformer helps with in the architecture sweep (Section 10.2). Both close the relational gap from different angles, which is why they are kept together downstream.

10.3.2 Second Tier: Strong but Budget-Limited

Triple value heads (+14.7 pp, ±4.0, Section 9.3) and ***PAE*** (+13.3 pp, ±8.0, Section 9.3) rank fifth and sixth by mean and belong with the group above rather than with the noise below: PAE improves the baseline on all three seeds, and triple value heads on two of three (essentially flat on the third). Both are value-function mechanisms built for shifting-target regimes, so their less-than-perfect consistency at this flat-reward 50 M budget is expected rather than disqualifying.

10.3.3 The Middle Band: Deferred Judgment, Not Failure

The features ranked below this second tier and above the *HL-Gauss* collapse, more than half the table, all carry a positive mean gain, several reaching well above the baseline noise floor (up to +13 pp). For most, the 50 M budget denies not the effect but the consistency to certify how much each contributes; a few are reliable yet still set aside downstream for reasons other than performance. They are grouped by the condition each one needs and does not get.

Capacity, Gradient-Path, and Variance-Bound Features

This first group is unresolved at this budget, either it adds parameters or gradient-path length that 50 M updates cannot fit, or its signal is dominated by cross-seed variance. The ***SPP critic*** (+5.2 pp, ±8.3, Section 9.7) and ***autoregressive sampling*** (+5.8 pp, ±7.6, Section 9.2) both extend the model along axes whose benefit only appears once the added structure is well fitted, and at this scale neither separates from noise. ***Hierarchical sub-action masking*** (+13.1 pp, ±11.5, Section 9.2) adds no parameters but carries the third-highest variance of the sweep (seed 3 at 86.8%, seed 1 below 60%), and combining it with autoregressive sampling (+7.2 pp, ±7.6) improves on autoregressive sampling alone but lands well below hierarchical masking alone, a sign the two interact in ways the budget cannot resolve. Its strong but erratic signal is set aside in the final selection for timing reasons (Section 10.3.6). ***Frame stack*** (+7.3 pp, ±4.5, Section 7.6) belongs here too: as noted above, its temporal signal is largely subsumed by *extended observations* at 4× the channel cost.

Value-Function Machinery Awaiting Shifting Targets

The second group depends on shifting value targets that a flat 50 M-step reward never produces. Its strongest members, *triple value heads* and *PAE*, were discussed in the second tier above; ***PopArt*** (+10.6 pp, ±8.7, Section 9.3) is the remaining case, reaching its largest win rate on seed 3 with no

consistent cross-seed effect. As with the other two, its natural home is the reward-annealing and phased regimes of the long runs, where the value targets it normalizes actually move.

Opponent-Distribution Mechanisms

The third group changes the opponent distribution the agent trains against. ***Adaptive opponents*** (+11.7 pp, ±1.4, Section 9.4) is the most reliable feature in this band, with the second-lowest variance of the entire sweep: its gain is real, not a measurement artifact. It is nonetheless set aside in the later runs, again not for performance: those runs curate the opponent pool by hand for finer control over the training distribution than the automatic scheduler allows. ***Self-play with PFSP*** (+8.9 pp, ±9.1, Section 9.4) sits at the opposite end, operating on a volatile, evolving checkpoint pool refreshed every 50 updates; its variance reflects that non-stationarity, since 50 M steps is too short for the pool to mature, and it is kept as a conditional feature for the longer runs where it has room to do so.

Auxiliary Heads and Minor Principled Additions

The remaining middle features split into two threads. Among the auxiliary heads, the gain shrinks as the prediction target grows more complex and less aligned with the policy: ***Aux unit count*** (+9.6 pp, ±12.6, Section 9.5) targets the simplest signal (a 12-dimensional per-type count) yet carries the largest variance of the auxiliary group, dropping 15 pp below baseline on seed 3; ***Aux spatial*** (+5.3 pp, ±5.2, Section 9.5) reconstructs the per-cell ownership map but is deliberately down-weighted to avoid dominating the PPO objective, which caps its gain; and ***Aux contrastive*** (+3.7 pp, ±6.6, Section 9.5) is the weakest, its InfoNCE signal too faint to reshape the encoder against PPO's much stronger gradient while competing for the same weights. The other thread holds two within-noise changes kept for principled reasons rather than measured strength: ***GELU*** (+8.3 pp, ±8.2, Section 9.7) alters the optimization landscape rather than representational capacity (which is why it is evaluated here rather than in the architecture sweep) and has no clear downside for deeper networks, while ***build-time rewards*** (+3.2 pp, ±11.2, Section 9.3) rescales unit-production rewards to reflect actual build cost, removing an unjustified uniform-cost assumption at no apparent risk.

None of these mechanisms is malfunctioning: the 50 M ablation simply does not grant them the training time, the shifting targets, or the mature curriculum they were built to exploit. The selection policy below keeps most of this band as per-run candidates, dropping only the few made redundant by a stronger feature, superseded by hand-curated control, or implemented too late.

10.3.4 Negative Findings: Two Distinct Failure Modes

The two below-baseline features fail for opposite reasons, and the distinction determines what to do with each. ***Augment Symmetry*** (−0.5 pp, ±4.0, Section 9.6) is not broken: quadrupling the batch through the four map symmetries reduces gradient variance but adds no strategic diversity on a fixed single map. Augmentation is meant to aid generalization, which is simply not the bottleneck at 50 M steps on one map with such compute. ***HL-Gauss*** (−18.5 pp, ±17.0, Section 9.3) is genuinely broken in this configuration, collapsing to 21.2% and 35.0% on seeds 2 and 3. This is not a verdict on distributional value prediction, which has strong precedents [100, 101], but a failure of the specific pairing of 255 bins, a default bin range, and PPO's clipped value loss; a fair evaluation would demand dedicated tuning of all three. Absent that tuning, *PopArt* already covers the same value-target stabilization more cheaply and is incompatible with it.

10.3.5 Synthesis

Read as a whole, the sweep reduces to *three levers*. The features that reliably help change *what the policy perceives* (*extended observations*, *filtered masks + reserved observations*, and *opponent modeling*)

or *which matchups its gradient focuses on* (*MCW*). The wide middle mostly changes *how that gradient is computed*: value-function machinery whose targets do not shift in a flat-reward run, auxiliary losses that compete with PPO, and capacity additions that need more updates than 50 M provides. These still help on average; the budget cannot certify by how much. The two below-baseline features sit outside this picture entirely, one a sound mechanism on the wrong benchmark, the other a sound idea in a broken configuration.

10.3.6 Final Feature Selection

The ablation results guide the feature choices for all subsequent experiments. Features are sorted into three groups based on the magnitude and consistency of their effect, on whether theoretical considerations override a noisy empirical signal, and on whether the measurement itself is trustworthy at this budget.

Always enabled. Five features are kept in every subsequent run (the *top-5* configuration below): four on empirical grounds, one on theoretical. The first four are *extended observations (73ch)*, *filtered masks + reserved observations*, *MCW*, and *opponent modeling*; they post the four largest mean gains of the sweep (each ≥ 19.8 pp), each with low cross-seed variance and no seed below baseline, at negligible or zero parameter cost. The fifth, *GELU*, is kept on theoretical grounds: it removes ReLU's dying-neuron pathology through a smooth, non-zero gradient on negative inputs, and has become the standard activation in modern deep architectures.

Conditionally enabled. The remaining features have variance comparable to baseline noise or target use cases beyond the single-map benchmark, so they are enabled selectively rather than across the board. *Self-play + PFSP* and *Augment Symmetry* are generalization-oriented and enabled in long-horizon or multi-map runs where opponent diversity and map invariance matter. *SPP critic* and *Build-time rewards* are theoretically motivated with no clear downside, and are kept. *PAE* and *PopArt* stabilize the value function under shifting targets and are enabled under reward annealing and phased schedules. Finally, *Frame stack (4)* and *Autoregressive* remain promising but inconclusive at this budget and are kept as per-run candidates.

Excluded. Eight features are dropped from all subsequent experiments. *Aux contrastive*, *Aux unit count*, and *Aux spatial* all add a gradient pathway competing with PPO over encoder weights; once *opponent modeling* supplies a reliable representation prior, their marginal value does not justify the interference. *HL-Gauss* duplicates *PopArt*'s role (value-target stabilization), is incompatible with it, and strictly less reliable here. *Adaptive opponents* is reliable but superseded by hand-curated pools giving finer control over the training distribution. Finally, *Triple value heads*, *Hierarchical mask*, and *Autoregressive + Hierarchical mask* are excluded for timing, not empirical weakness: all carry positive signals but were implemented too late for the final agent.

10.3.7 Selection Validation

To validate the five-feature cutoff, two configurations are trained for 100 M steps on *basesWorkers16x16A* with the baseline hyperparameters of Table 10.1: the five always-enabled features alone, and the same set augmented with every conditionally enabled feature (excluding self-play and the symmetry augmentation, since the focus here is single-map skill). Their win-rate trajectories are reported in Figure 10.1.

The results confirm the ablation's prediction. The top-5 configuration matches the all-features asymptote by ~70 M steps and reaches the 95% win-rate milestone roughly 30 M steps earlier, a ~40% reduction in compute to the same target. Beyond 80 M, both regimes plateau in the 96–100% range, a spread narrow enough that the endpoint difference falls within single-run noise; the comparison

therefore turns on the convergence trend rather than the asymptote itself. Within that trend, the conditional features neither destabilize training nor lift the asymptote, but they do slow it: the added capacity (SPP critic, frame stack, autoregressive sampling) and value-function machinery (PAE, PopArt) lengthen the gradient path and widen the optimization surface without supplying new signal for the policy to exploit, and the optimizer cannot convert that overhead into asymptote gains within budget. Both configurations substantially outperform the architecture-only baseline (86.3% at the same budget, Table 10.2), confirming that perception and signal-allocation gains compound on top of architectural ones rather than substitute for them.

Taken together, the three configurations sharpen the conclusion of the ablation: within the 50–100 M-step regime, the bottleneck is the quality of the training signal, not the capacity of the network. The encoder already has enough representational room to reach the asymptote, but only the signal-shaping features deliver gradients informative enough to do so within budget.

The top-5 cutoff is retained as the default for all subsequent experiments, with conditional features re-introduced only when their motivating conditions are present.

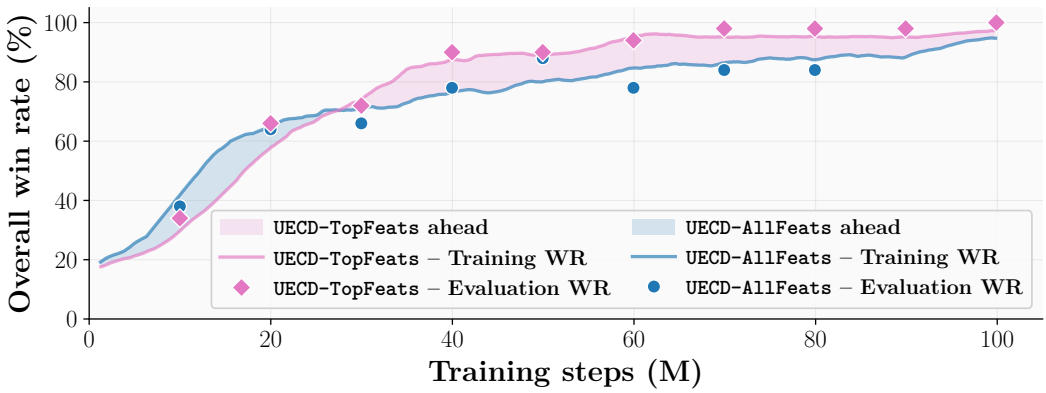


Figure 10.1: Win-rate trajectories over 100 M training steps on *basesWorkers16x16A* for the top-5 configuration and the all-features configuration (top-5 plus every conditionally enabled feature, excluding self-play and the symmetry augmentation). Bold markers show evaluation win rates; faint lines show smoothed in-training win rates over a sliding window of 50 matches.

10.4 Building the Single-Map Agent

With the UECD architecture and the top-5 feature set both fixed, full-scale training on *basesWorkers16x16A* can now target the strongest possible single-map agent. The process unfolds in successive stages, each responding to the shortcomings of the previous run.

10.4.1 The Sparse-Reward Trap: A Discount-Induced Rush Collapse

A first long-horizon training run is conducted to assess how the top-5 configuration of Section 10.3.7 extends from the dense shaping signal of the 50–100 M-step ablation regime to the sparse win/loss-only reward. The setup modifies the validated configuration along three axes: the training horizon is raised from 100 M to 300 M steps; the shaped-reward weight w_{shape} is annealed linearly from 1.0 to 0.0 over $[90, 210]$ M steps, transitioning the agent toward a pure terminal-reward regime; and self-play with PFSP is enabled for opponent diversity, pairing 12 self-play environments with 12 bot environments split across 3 CoacAI, 3 Mayari, 2 RandomBiasedAI, 2 POWorkerRush, and 2 POLightRush. The run is divided into three phases with decreasing learning rate and entropy coefficient, interpolated linearly across phase transitions (Figure 10.2).

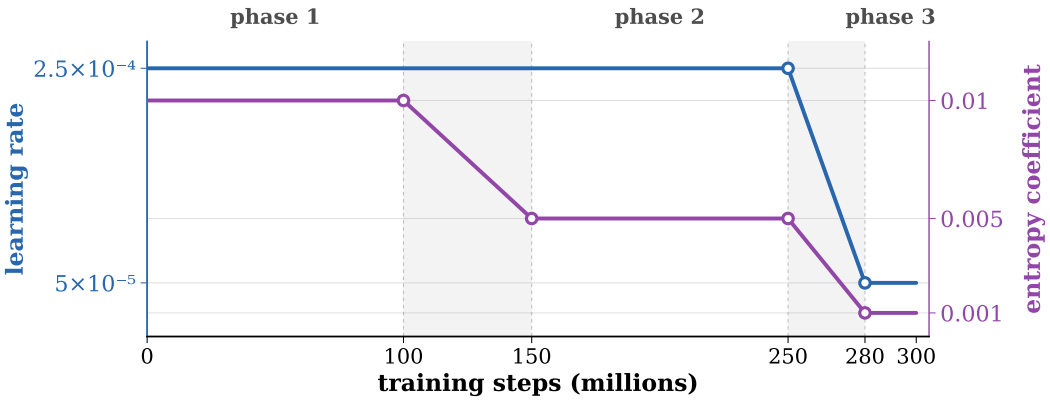


Figure 10.2: Phase schedule for the 300 M-step reward-scheduling run: learning rate α and entropy coefficient β across the three training phases, linearly interpolated between adjacent phases.

Figure 10.3 traces evaluation win rate, average episode length, and unweighted per-component returns over 300 M steps, highlighting the shaped-weight annealing window.

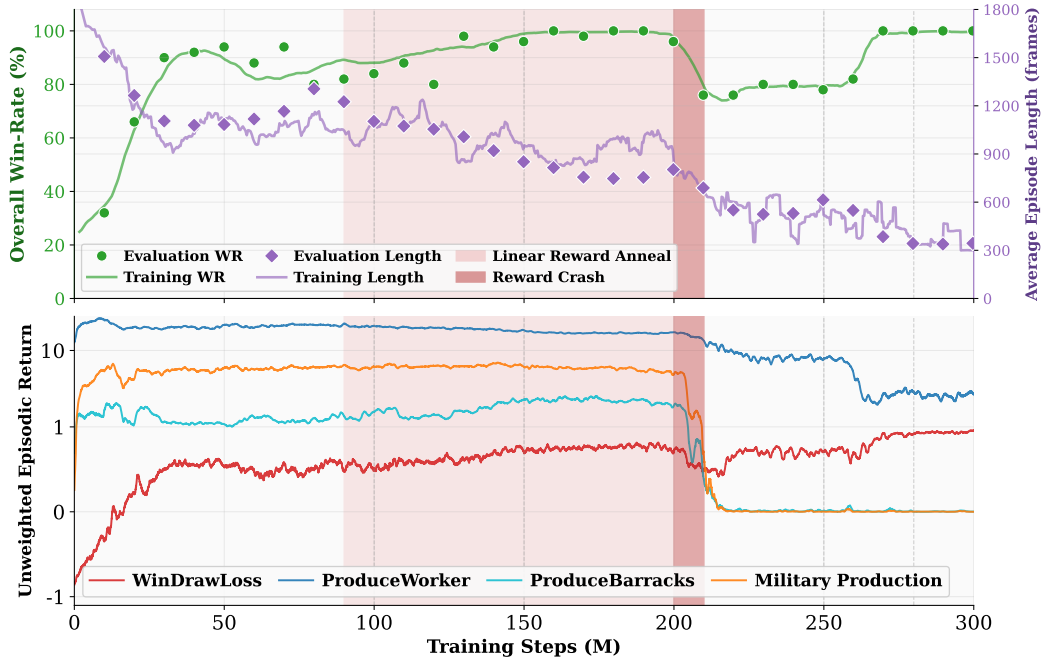


Figure 10.3: Rush collapse over the 300 M-step reward-scheduling run. *Top:* evaluation win rate (green) and average episode length (purple). *Bottom:* unweighted per-component returns on symlog scale (*WinDrawLoss*, *ProduceWorker*, *ProduceBarracks*, *MilitaryProduction*). Light red marks the shaped-weight annealing window $[90, 210]$ M steps, with a darker inner band at the crash.

These training dynamics separate into two regimes. Up to ≈200 M steps, the agent develops a balanced strategy: evaluation win rate climbs steadily toward 100%, episode length stabilizes around

750 frames, and all reward components rise in concert. After w_{shape} reaches zero near 210 M, evaluation win rate dips to ≈75% before recovering to near-100% by 280 M, while episode length collapses from ≈750 to ≈300 frames and the macroeconomic and military components of the return (*ProduceBarracks*, *MilitaryProduction*) fall to zero. The recovery is illusory. The agent has abandoned the strategic components of its policy and is winning by rushing the opponent with its starting workers, a strategy that exploits a short-horizon vulnerability of the training pool. The question is why a policy that had converged to a balanced strategy at 200 M steps abandons it within the following 90 M steps, precisely as the shaped reward is removed.

The Discount-Induced Shortcut

The collapse is mechanical rather than behavioral: it follows directly from the structure of the discounted-return objective once the shaping signal has been annealed away. Within a trajectory $\tau = (s_0, a_0, r_1, \ldots, s_T)$, the episodic return decomposes as

$$G(\tau) \;=\; w_{\text{shape}} \sum_{t=0}^{T-1} \gamma^t \, r_{t+1}^{\text{shape}} \;+\; w_{\text{term}} \, \gamma^{T-1} \, r_T^{\text{term}}, \tag{10.1}$$

where w_{shape} is the shaped-reward weight (annealed linearly), r_t^{shape} are the dense per-step shaping rewards, $w_{\text{term}} = 10$ is the terminal-reward weight, $r_T^{\text{term}} \in \{-1, 0, +1\}$ is the terminal win / draw / loss signal, T is the policy-dependent episode length, and $\gamma = 0.99$.

The collapsed objective. The agent enters the annealing window already proficient: at 90 M steps, before w_{shape} begins to drop, it wins between 80% and 90% of its games against the training pool, so r_T^{term} is strongly positive in expectation. As w_{shape} is driven to zero, the shaped sum in Equation 10.1 vanishes and the PPO objective collapses to

$$J(\pi) \;\xrightarrow{w_{\text{shape}} \to 0}\; w_{\text{term}} \, \mathbb{E}_{\tau \sim \pi}\left[\gamma^{T-1} \, r_T^{\text{term}}\right] \;\approx\; 10 \, \mathbb{E}_{\tau \sim \pi}\left[\gamma^{T-1}\right]. \tag{10.2}$$

Because $\gamma < 1$, the function $T \in \mathbb{N} \mapsto \gamma^{T-1}$ is strictly decreasing, so the optimal policy under Equation 10.2 is the one that minimizes $\mathbb{E}_\pi[T]$, independently of any quality-of-play criterion. Faster wins are not merely preferred; they become the only direction in which the surrogate objective can grow.

Transfer to PPO-Clip. Equation 10.2 is a statement about the abstract objective $J(\pi)$. The question is how this abstract bias gets transmitted to the actual policy parameters during training. The transmission goes through the empirical advantage. PPO does not optimize $J(\pi)$ directly; it optimizes the clipped surrogate $L^{\text{CLIP}}(\theta)$ introduced in Equation 3.16, whose two key quantities for the rush collapse are the empirical advantage $\hat{A}_t$, which carries the bias into the gradient, and the clip operator, which fails to suppress it.

The advantage carries the bias. Once $w_{\text{shape}} = 0$, the only non-zero reward in a trajectory is the terminal one, so the return-to-go from state s_t along a trajectory of length T simplifies to

$$G_t^{(T)} \;=\; 10 \, \gamma^{T-t-1} \, r_T^{\text{term}},$$

and the empirical advantage is $\hat{A}_t = G_t - V(s_t)$. The key point is that $V(s_t)$ is a function of state only: it does not depend on the trajectory that follows s_t. Comparing two rollouts that share state s_t but end at T and $T - \Delta$, the same $V(s_t)$ is subtracted from both returns, so the difference between their advantages is exactly the difference between their returns:

$$\hat{A}_t^{(T-\Delta)} - \hat{A}_t^{(T)} \;=\; G_t^{(T-\Delta)} - G_t^{(T)} \;=\; \left(\gamma^{-\Delta} - 1\right) G_t^{(T)} \;>\; 0. \tag{10.3}$$

Equation 10.3 is the bridge from the return ratio to the gradient: at every state visited by both trajectories, the action that led to the shorter rollout receives a strictly larger empirical advantage than the action that led to the longer one, irrespective of where $V(s_t)$ sits. The gap grows exponentially with Δ: a one-step shortening multiplies the base return by $\gamma^{-1} \approx 1.01$, and the effect compounds across the remaining horizon.

The clip does not suppress it. Inspecting Equation 3.16, the clip operator restricts ρ_t to $[1-\epsilon, 1+\epsilon]$, but $\hat{A}_t$ enters the loss as an unclipped multiplier. PPO's clipping therefore controls only how aggressively the policy is allowed to shift per minibatch in response to a given advantage; it places no bound on how large that advantage can be. As trajectories shorten and $\hat{A}_t$ inflates according to Equation 10.3, the gradient $\nabla_\theta L^{\text{CLIP}}$ grows accordingly, and each minibatch update moves the policy toward whatever actions produced the largest empirical advantages, which in the collapsed regime is mechanically the actions that led to the shortest episodes.

Numerical magnitude. Under the run's actual numbers, the amplification is substantial. Shortening the expected episode length from $T_{\text{eq}} \approx 750$ frames (the pre-collapse equilibrium) to $T_{\text{rush}} \approx 300$ frames (the post-collapse rush) amplifies the discounted terminal signal by

$$\frac{\gamma^{T_{\text{rush}}-1}}{\gamma^{T_{\text{eq}}-1}} = \gamma^{T_{\text{rush}}-T_{\text{eq}}} = \gamma^{-450} = 0.99^{-450} \approx 92. \tag{10.4}$$

A two-orders-of-magnitude differential between a normal-length win and a fast rush is more than enough to override whatever residual signal the value network had learned during the shaped phase. The per-component returns of Figure 10.3 make this explicit: *WinDrawLoss* keeps climbing as the agent gets better at the shorter game, while *ProduceBarracks* and *MilitaryProduction* fall to zero and *ProduceWorker* drops from ≈13.0 to ≈3.0, because the discounted objective no longer rewards them.

Conditions and mitigation. The pathology arises from three interacting conditions: (i) a discount factor $\gamma < 1$, which makes γ^{T-1} sensitive to episode length; (ii) a near-terminal-only reward regime, produced by the shaped weight reaching zero; and (iii) policy control over T, which gives the optimizer a route to amplify the terminal signal by choosing shorter trajectories. Mitigation can target any of the three: raising γ toward 1 flattens γ^{T-1} across feasible T; preserving a non-zero shaping floor maintains a dense per-step gradient that dominates the discounted-terminal one; and an undiscounted average-reward formulation removes the dependence entirely. The remedy adopted in Section 10.4.2 follows the second route: the shaped-weight floor is raised from 0 to 0.1, restoring a productive-action gradient strong enough to anchor the optimizer to balanced play.

Out-of-Distribution Cost

Whether the post-collapse policy is genuinely competent or merely exploits a training-pool weakness can be settled empirically by testing it against opponents the agent never saw. Figure 10.4 compares the agent at 150 M steps (the pre-collapse balanced policy) with the agent at 300 M steps (the post-collapse rush) against the two strongest in-training opponents (CoacAI, Mayari) and two scripted CoG competitors withheld from training (TMA, ObiBotKenobi). If the rush were a transferable strategy, it would hold against both; if it were a shortcut tied to the training distribution, it would hold on the first pair and collapse on the second.

On in-training opponents, the two checkpoints are indistinguishable on win rate (~98–100% for both); the only signature of the strategy change is the drop in average episode length from ~900 to ~300 frames. On held-out opponents, the picture inverts: WR collapses from 56–70% to near zero, while episode lengths fall from ~1 300 to ~850 frames, well above the ~300 frames of a successful rush. The rush is attempted but fails: early worker aggression that broke CoacAI and Mayari is dismantled by opponents with more robust early-game defenses.

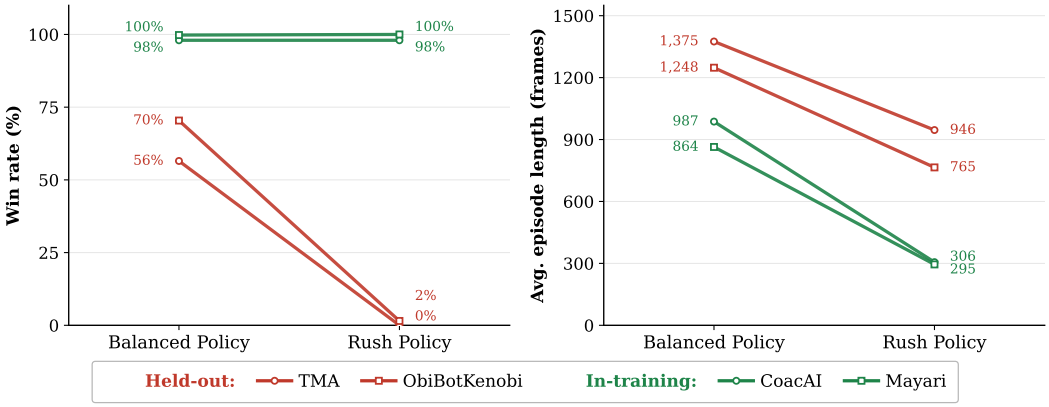


Figure 10.4: Win rate and average episode length (frames) for the 150 M (balanced policy) and 300 M (rush policy) checkpoints. The rush matches the balanced policy on in-training opponents (green) but collapses on held-out ones (red).

This is the empirical signature of the discount-induced shortcut. Against held-out opponents, neither pillar of the rush incentive holds: the rush does not produce a fast win, and many attempts terminate in losses. The shortcut is selected for by the training distribution and collapses against everything else. All subsequent runs therefore retain a 10% shaped-reward floor throughout training; this residual signal keeps the value function calibrated and the policy macro-oriented while the dominant gradient contribution still comes from the win / loss reward.

10.4.2 Two-Phase Opponent-Pool Fine-Tuning

Fine-tuning runs in two phases of 24 environments from the 150 M checkpoint, with progressively stronger opponents and annealed learning rate, entropy, and shaped weight (Table 10.6).

Table 10.6: Two-phase fine-tuning schedule from the 150 M checkpoint to the final UECD-`Best` agent. Values interpolate linearly between phase endpoints.

	150 M	*Phase 1*	**240 M**	*Phase 2*	**350 M**
Learning rate	5×10^{-5}	$\longrightarrow$	2×10^{-5}	$\longrightarrow$	9.5×10^{-6}
Entropy coef.	5×10^{-3}	$\longrightarrow$	2×10^{-3}	$\longrightarrow$	1.2×10^{-3}
Shaped weight	1.0	$\longrightarrow$	0.25	$\longrightarrow$	0.10

Phase 1: broaden the opponent pool (150 M → 240 M). The three weakest scripted bots (RandomBiasedAI, WorkerRush, LightRush) are replaced by two competition-level opponents, TMA and ObiBotKenobi, at 2 environments each. Self-play expands from 12 to 16 environments, evenly split between the latest checkpoint and a PFSP-hard sample from the historical pool. By 240 M, the agent reaches near-perfect win rate against all four scripted opponents in the updated pool while preserving the balanced, macro-oriented strategy that the 10% shaped-reward floor maintains.

Phase 2: harden against RAISocketAI (240 M → 350 M). RAISocketAI joins the bot pool at 4 environments, with each of the four other opponents reduced to 1 environment; the self-play allocation is unchanged. This phase tightens the policy against the strongest available opponent and produces the final agent, UECD-`Best`.

Compute. Total training compute for UECD-Best amounts to 9.47 GPU-days of full-run wall-clock on a single RTX 6000 Ada, with a strict marginal cost (on-path segments only) of 7.21 GPU-days. Table 10.7 decomposes this budget by phase; the progressive slowdown it shows reflects heavier opponent inference as the pool broadens (Phase 1) and shifts to RAISocketAI (Phase 2). The gap between full-run and on-path totals captures wall-clock spent past the branch points at 150 M and 240 M on segments ultimately discarded from the final agent's training trajectory. RAISocketAI itself used ≈1.5 B steps over 70 GPU-days across 8 maps; of this, ≈500 M steps and 23.6 GPU-days were spent on the small-map subset (≤16×16, including *basesWorkers16x16A*). On this subset specifically, UECD-Best uses less than half the wall-clock budget (9.47 vs 23.6 GPU-days) and trades breadth for depth: ≈350 M steps on a single layout against ≈500 M split across the subset. This compute frugality is also an environmental consideration, keeping the energy footprint modest relative to large-scale RL efforts.

Table 10.7: Per-phase training budget for UECD-Best on *basesWorkers16x16A* (RTX 6000 Ada).

Phase	**Steps (M)**	**Eff. SPS**[†]	**On-path (hours)**[‡]	**Full run (hours)**[§]
From scratch	0–150	1 226	34.0	68.5
Phase 1 (broaden pool)	150–240	825	30.3	50.0
Phase 2 (vs RAISocketAI)	240–350	281	108.7	108.7
$\sum$	**350 M steps**		**173.0 hours** **7.21 days**	**227.2 hours** **9.47 days**

[†] Effective step rate over each segment (segment steps divided by on-path wall-clock).
[‡] Charges only the steps retained in the final agent.
[§] Actual wall-clock of each training run; the from-scratch and Phase 1 runs continue past their branch points (150 M and 240 M).

Why a 10% floor and not pure sparse. A small but persistent shaped signal pins the value function to the same scale throughout training, bypassing the discount-shortcut described in Section 10.4.1. The 10% floor is a pragmatic stand-in rather than the principled solution. Two alternatives exist. PopArt [99] could in principle absorb the magnitude shift, but the shaped→sparse transition is not merely a scale change: it removes entire reward components, fundamentally altering what the value function must predict. PopArt handles non-stationarity in magnitude, not in structure. The cleaner remedy is a multi-head value architecture, with separate heads for the shaped, win / loss, and cost components (as in RAISocketAI), which absorbs the structural shift by design rather than by floor. Triple value heads were in fact the intended solution here. They appear in the feature ablation under that name (Table 10.4), but the result understates their potential: the implementation in use at that time contained bugs that were only resolved after UECD-Best had been trained. The final configuration therefore falls back on the 10% floor: less elegant, but functional and stable.

10.4.3 The Emergent Ranged-Only Strategy

After 350 M cumulative steps, UECD-Best converges to a consistent *Ranged-only* composition. The agent assigns three workers to early-game resource gathering (RAISocketAI and most other strong bots use two), funding a faster Barracks followed by continuous Ranged-unit production, then exploits the unit's superior range to kite approaching attackers. Replays show this is the dominant tactic against every opponent in the pool, including those that deploy mixed compositions: Light or Heavy rush patterns get decimated before reaching the Ranged line. A *companion website* hosts 36 recorded UECD-Best matchups, the source code, the extended tournament metrics, and the CoG bots archive at https://mathisdelsart.github.io/microrts-drl-uecd-website.

This behavior is *emergent*. It is not encoded in the reward weights, the architecture, or any explicit shaping term. It is the agent's own answer to the joint optimization problem of (i) maximizing the win / loss signal, (ii) tolerating the 10% shaped floor that rewards macro activity, and (iii) staying robust against the diverse opponent pool seen during fine-tuning.

The single residual weakness. The *Ranged-only* build holds against RAISocketAI in the majority of games, but its fast Light rush occasionally pierces it before the Ranged line forms (Light > Ranged in close combat, Figure 2.4). Two interventions targeted a hybrid Light/Ranged composition, both applied over a transition window and annealed back to baseline: ***(i)*** reward weights re-shaped to elevate *ProduceLight* and penalize *ProduceRanged*, and ***(ii)*** the entropy coefficient held at an elevated constant ($\beta = 5 \times 10^{-3}$) to slow re-convergence on a single unit type. Neither shifted the policy meaningfully: the agent returned to its *Ranged-only* build once the modifications annealed away, suggesting that the *Ranged-only* optimum is a strong attractor under the current optimization landscape. A more thorough investigation of reward shaping, opponent curricula, or architectural changes promoting compositional flexibility is left for future work.

10.5 Evaluating the Single-Map Agent

A 19-agent round-robin tournament on *basesWorkers16x16A* (Chapter 6) compares the four RL agents developed in this chapter (`UECD-AllFeats` and `UECD-TopFeats` from Section 10.3.7, `UECD-Rushed` from Section 10.4.1, and `UECD-Best` from Section 10.4.2) against 15 competition bots. The 1 710 pairwise games (10 per matchup, 5 as P0, 5 as P1) follow the CoG deterministic-action protocol and completed without crash or timeout across all matchups.

10.5.1 Win Rate as the Primary Performance Signal

Figure 10.5 reports the overall ranking. `UECD-Best` achieves the top score with 174 points, 10 ahead of RAISocketAI (164). The lead is not an artifact of including multiple UECD variants: restricted to external (non-UECD) agents alone, `UECD-Best` still ranks first. The intermediate UECD checkpoints map directly onto each section's contribution. `UECD-TopFeats` (131 points), from the feature ablation of Section 10.3.7, already places fifth overall. `UECD-Best` adds 43 more points, the measurable gain from the two-phase fine-tuning of Section 10.4.2. `UECD-AllFeats` (110 points) ranks below `UECD-TopFeats`, confirming Section 10.3.7's finding: adding the conditional features on top of the top-5 does not improve performance and may slow convergence at this budget. `UECD-Rushed` (98 points) ranks last among the UECD variants but still ahead of GridNet (60.5 points), the published DRL baseline for MicroRTS [11]. This gap is partly inflated by an asymmetry in how GridNet was trained: it only learned to play as P0 and loses every game when forced into the P1 role, which roughly halves its effective tournament total. Even after doubling GridNet's score to compensate (an upper bound, since P0 skill need not transfer to P1), both `UECD-TopFeats` (131) and `UECD-Best` (174) still exceed it, confirming that the architectural and feature contributions of this chapter compound positively over the published baseline. Expressed as tournament win rates, this corresponds to a +63.06 pp improvement, from 33.61% for `GridNet` to 96.67% for `UECD-Best`[1].

The head-to-head matrix (Figure 10.6) refines this ranking. `UECD-Best` wins 100% of games against every opponent except WorkerRush (50%) and RAISocketAI (90%). The WorkerRush result is an artifact of the deterministic CoG protocol: when both policies are deterministic, the ten games against a single opponent collapse into two identical replays, one per starting position, so a single positional asymmetry produces the 5–5 split. Under the stochastic protocol used elsewhere in

[1]Tournament-point totals divided by the maximum (180): 60.5/180 = 33.61% for *GridNet* and 174/180 = 96.67% for *UECD-Best*. After symmetry correction for *GridNet*'s P0-only training (doubled score, ~67.2%), the gap narrows to ~29 pp but remains decisive.

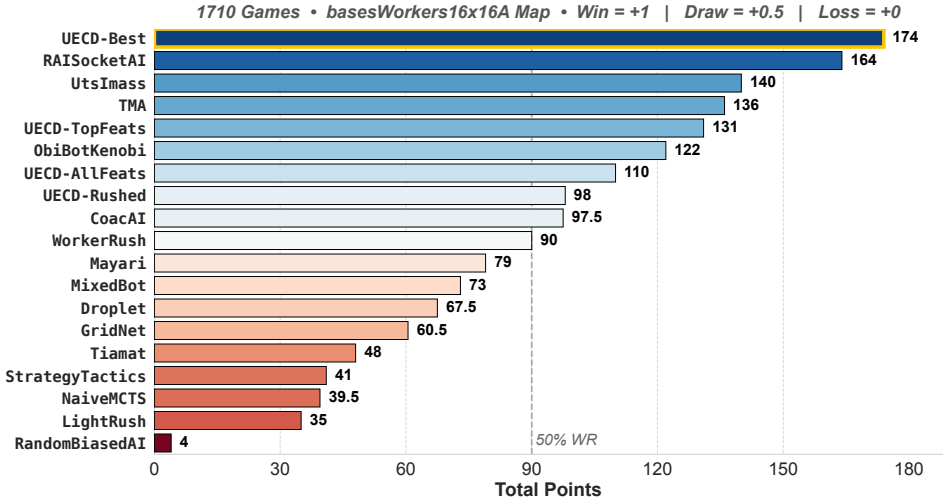


Figure 10.5: Final tournament standings on *basesWorkers16x16A* across 19 agents under the CoG deterministic-action protocol (1 710 pairwise games, 10 per matchup split as 5 P0 and 5 P1). Ranking is by total points (win = 1, draw = 0.5, loss = 0); the dashed line marks the 50% win-rate threshold.

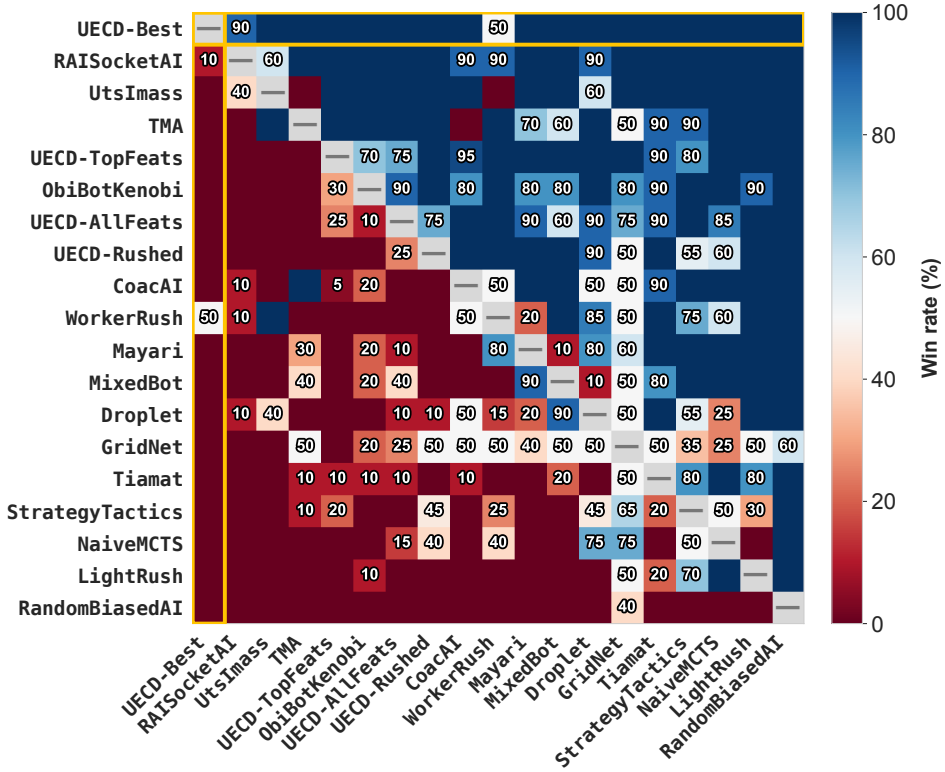


Figure 10.6: Head-to-head win-rate matrix, agents ordered by total points (matching Figure 10.5). Each cell shows the row agent's win rate (%) against the column agent over 10 games; the color scale runs from red (0%) to blue (100%). Diagonal cells are empty (no self-play).

this chapter (1000 games per matchup), UECD-Best wins 99.3% (±0.5 pp) against WorkerRush, ruling out a structural blind spot. The RAISocketAI head-to-head, re-evaluated under the stochastic protocol, settles at 65.7% (±2.9 pp): still a clear lead, but tighter than the 9–1 deterministic split suggests[2]. The intermediate UECD rows of the matrix show a consistent pattern: they win against the weaker scripted bots on the right side of the matrix but collapse against the strongest opponents on the left (TMA, ObiBotKenobi, RAISocketAI). This is the same fragility profile diagnosed in Section 10.4.1, now visible across the full opponent pool: the two-phase fine-tuning is what bridges the gap to fully competitive play, not the architecture or feature set alone.

10.5.2 Game-Theoretic Metrics as Robustness Checks

The point totals of Figure 10.5 can be sensitive to the composition of the opponent pool: adding clones of a weak agent, for instance, would inflate every other agent's score uniformly. To complement the standings, four game-theoretic metrics are computed on the 19×19 win-rate matrix of Figure 10.6, each capturing a distinct property of the agent's behavior across the pool.

The Copeland score [83] counts the signed pairwise wins minus losses, measuring dominance *breadth* regardless of margin; α-Rank [82] reports the long-run stationary mass of each agent under an evolutionary invasion dynamic, identifying the strategies hardest to displace; Nash averaging [81] solves the win-rate matrix as a zero-sum meta-game and reports the agent's expected payoff against the equilibrium mix, separating dominated strategies from the equilibrium frontier; Regret [81, 84] measures the gap between each agent's win rate and that of the pool's best response, reported both averaged across opponents (*average regret*) and on the single hardest matchup (*worst-case regret*).

The mathematical definitions and a four-agent worked example illustrating how the metrics complement one another on the same matrix are deferred to Appendix C. Table 10.8 reports the top-3 agents per metric, excluding Nash because its top-3 is a perfect tie at score 0 (discussed below).

Table 10.8: Top-3 agents per game-theoretic metric, computed on the 19×19 win-rate matrix. ↑ ≡ *"higher is better"* and ↓ ≡ *"lower is better"*.

Metric	Top #1		Top #2		Top #3	
Copeland (↑)	UECD-Best	+17	RAISocketAI	+16	TMA	+11
α-Rank† (↑)	UECD-Best	0.339	RAISocketAI	0.182	TMA	0.079
Avg. regret (↓)	UECD-Best	0.03	RAISocketAI	0.06	UtsImass	0.19
Worst-case regret (↓)	RAISocketAI	0.40	UECD-Best	0.50	*Tie*	N/A

† Selection intensity $\alpha = 0.1$, Moran population $m = 50$.

Dominance Breadth: Copeland Score

Figure 10.7 reports the Copeland score of every agent in the tournament. UECD-Best leads with +17, RAISocketAI follows at +16, and TMA is third at +11. Because the Copeland score reduces each matchup to a sign (+1 for a win, −1 for a loss) and discards margin, the metric reports *dominance breadth*: the number of opponents an agent beats more often than not, regardless of margin. UECD-Best's score of +17 corresponds to beating 17 of its 18 opponents head-to-head; the single missing point comes from the WorkerRush deterministic-protocol tie discussed in the head-to-head analysis above, which lands at exactly 50% and yields a Copeland contribution of 0. RAISocketAI's +16 comes from the same 17-win profile, with its single non-win being a pairwise loss to UECD-Best rather than a draw. The drop of 5 points down to TMA at the next position reflects the same

[2] Confidence intervals are 95% Wald intervals on a binomial proportion, $\pm 1.96\sqrt{p(1-p)/n}$ with $n = 1000$ games: ±0.5 pp at $p = 0.993$ and ±2.9 pp at $p = 0.657$. These capture sampling noise at a fixed checkpoint, not seed-to-seed variance, since *UECD-Best* is a single training run.

discontinuity already visible in the leaderboard: two agents sit at the top of the pool, the rest decline progressively from +11 down through −18 for RandomBiasedAI.

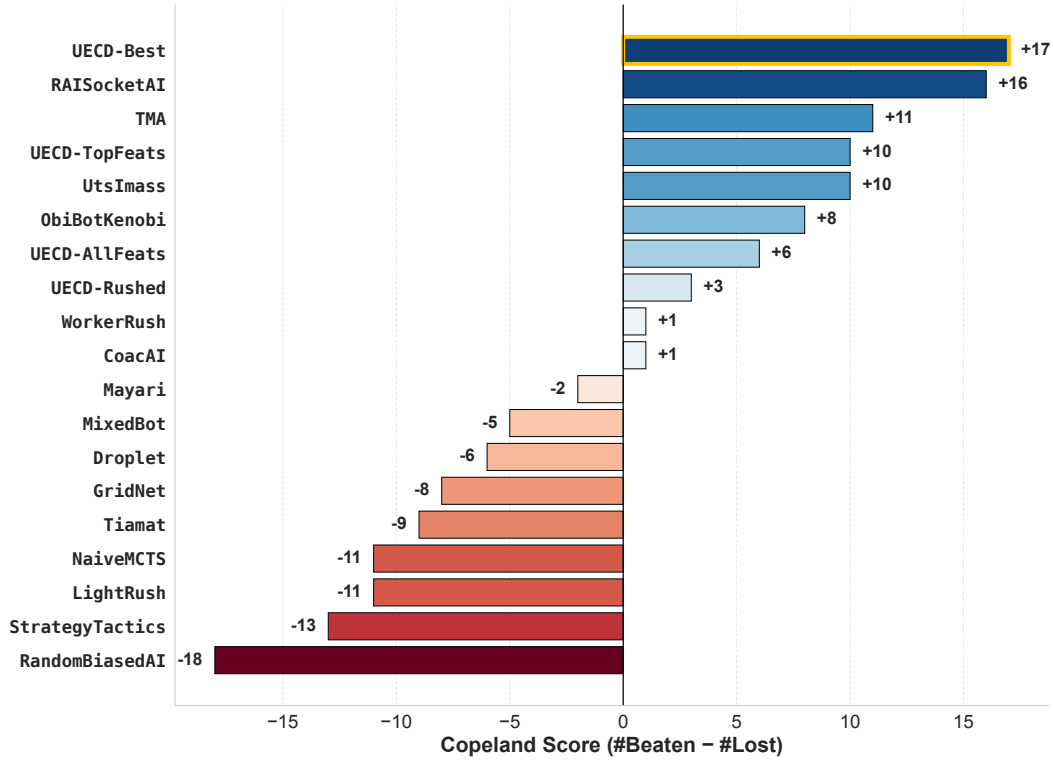


Figure 10.7: Copeland scores on the tournament's 19×19 win-rate matrix, agents sorted from highest to lowest score.

Evolutionary Stability: α-Rank

Figure 10.8 traces the α-Rank stationary mass of each agent as the selection intensity α varies from 10^{-3} to 10^{3}. Two curves dominate the picture entirely. UECD-Best's mass grows monotonically with α and rises toward 1.0 at high selection pressure, displacing every other strategy out of the population. RAISocketAI traces a single low bell that peaks just below 0.2 at moderate α, where UECD-Best already sits around 0.45 (read at a higher selection intensity than the $\alpha = 0.1$ used in Table 10.8), before collapsing back to zero as α grows further. No other agent ever separates from zero on the visible scale.

What this asymmetry reveals is precisely what the leaderboard alone cannot show. At low α, selection is too weak to distinguish strategies, and every agent receives roughly equal stationary mass ($1/19 \approx 0.053$). As α grows, the dynamic favors strategies that resist invasion. UECD-Best is the unique agent that resists invasion from *every* other strategy in the pool, so its mass grows monotonically toward the full population. RAISocketAI resists invasion from every agent *except* UECD-Best; at moderate α it accumulates non-trivial mass by dominating the rest of the pool, but at higher selection pressure UECD-Best invades it and the bell collapses. The seventeen other agents are invadable by multiple opponents from the start, and never accumulate measurable mass.

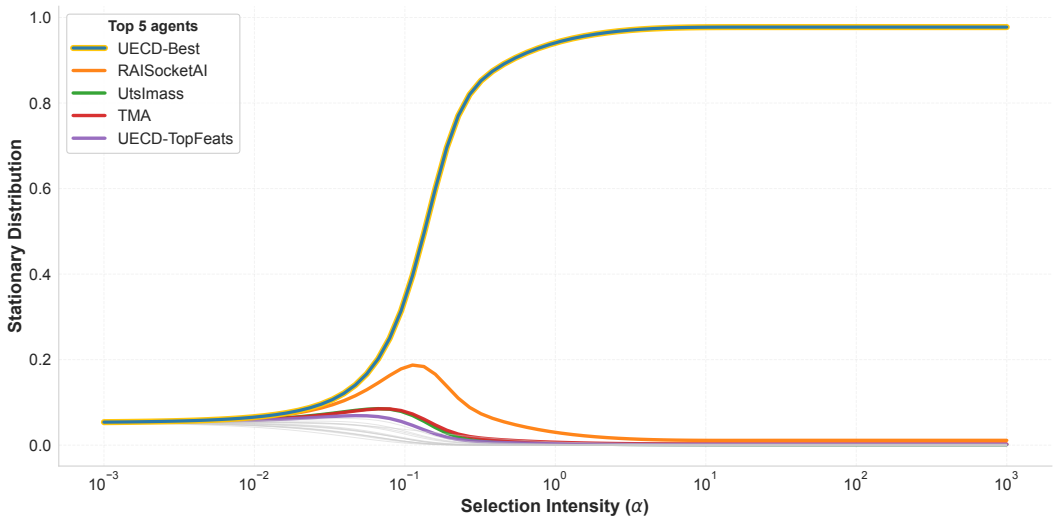


Figure 10.8: α-Rank stationary mass as a function of the selection intensity α. Each curve tracks an agent's long-run share of the population under the evolutionary invasion process; only the top-5 agents are highlighted, the rest grouped at the bottom.

The Non-Exploitable Frontier: Nash Averaging

Figure 10.9 shows a sharply bimodal distribution. Eight agents tie at exactly 0 (UECD-`Best`, RAISocketAI, TMA, ObiBotKenobi, WorkerRush, and the three other UECD variants), while the remaining eleven sit strictly below, from −0.100 for Mayari down to −0.500 for the weakest. No agent occupies the intermediate range: the split at zero is clean and binary.

The ties at 0 are equal best responses to the equilibrium mix $\mathbf{p}^*$: against an opponent playing optimally, none of them can be exploited. The negative-score agents are dominated: the equilibrium mixture beats them in expectation, so they are never played at equilibrium. All four UECD variants sit on the non-exploitable side, including UECD-`Rushed` despite its mid-tournament ranking.

Nash averaging therefore reads the tournament as a structural partition rather than a continuous ranking: a small frontier of non-exploitable strategies separated by a clean discontinuity from the dominated rest of the pool. The other metrics produce a smooth ordering, whereas this one produces a binary one, and both draw the boundary at the same agents.

Robustness to the Best Response: Average and Worst-Case Regret

Figure 10.10 shows two complementary views of the regret metric, both flipped to robustness (robustness = 1− regret) so higher is better in both panels. The worst-case view (left) is striking: only RAISocketAI (0.60) and UECD-`Best` (0.50) reach non-zero scores; every other agent sits at exactly 0. The labels indicate where each agent loses the most ground to the best response in the pool. UECD-`Best`'s worst-case is WorkerRush, the deterministic-protocol tie analyzed earlier; RAISocketAI's worst-case is UECD-`Best` itself, the matchup where its win rate falls furthest below the best response. The bottom seventeen all hit the floor against UECD-`TopFeats`, TMA, or UECD-`Rushed`, the opponents the best response decisively beats. The mean-case view (right) recovers a continuous ranking that closely tracks the headline leaderboard, with UECD-`Best` (0.97) and RAISocketAI (0.94) at the top followed by a monotone decline.

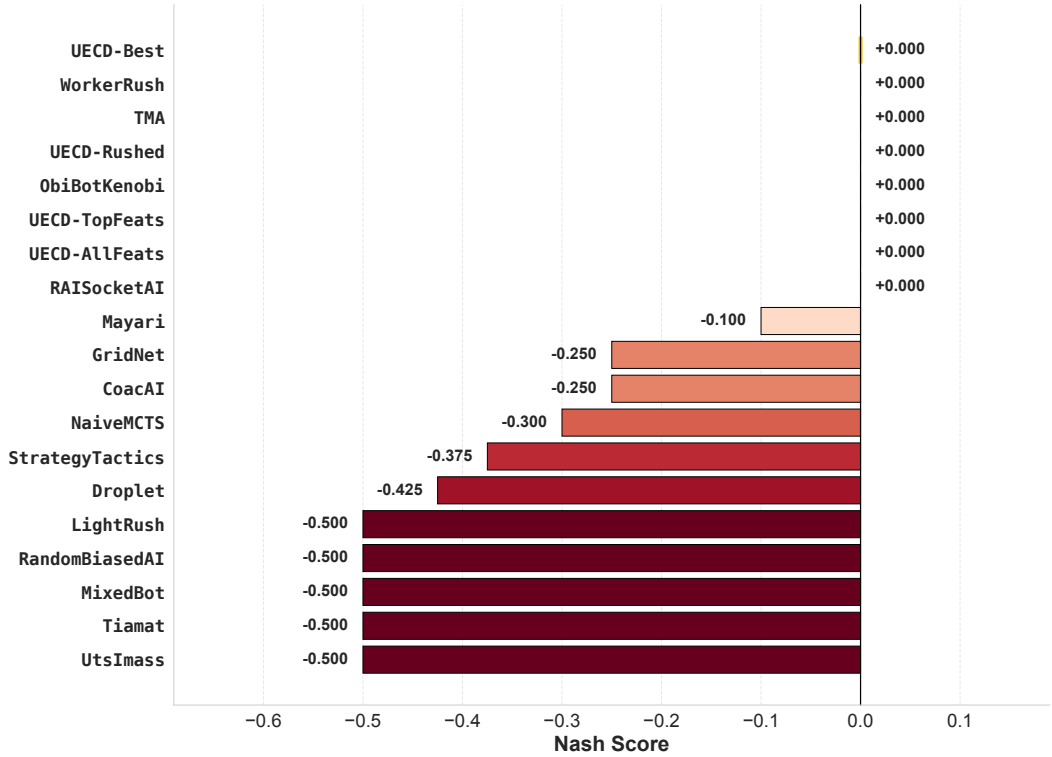


Figure 10.9: Nash equilibrium scores on the tournament's 19×19 win-rate matrix. Eight agents tie at exactly 0 (the value of the game), forming the equal-best-response frontier; agents with strictly negative scores are dominated strategies, never selected at equilibrium.

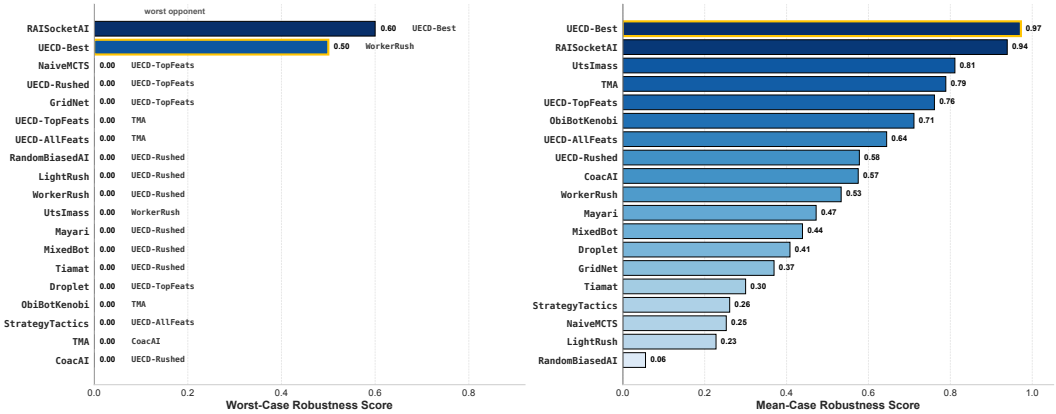


Figure 10.10: Robustness scores on the tournament's 19×19 win-rate matrix. Left: worst-case robustness. Right: mean-case robustness. Both quantities equal $1 - r$ where r is the corresponding regret in Table 10.8; higher is better in both panels.

10.6 Generalization Probes

Single-map dominance does not imply broader competence. UECD-Best's single-map specialization isolates architectural and training contributions for a clean per-component signal, but leaves open how the resulting policy behaves once the training distribution shifts. Two probes give a first cut at this: a *layout shift* (same size, different topology: *TwoBasesBarracks16x16*) and a *scale shift* (same layout family, larger size: *basesWorkers32x32A*). Each pairing plays 100 stochastic-sampling games.

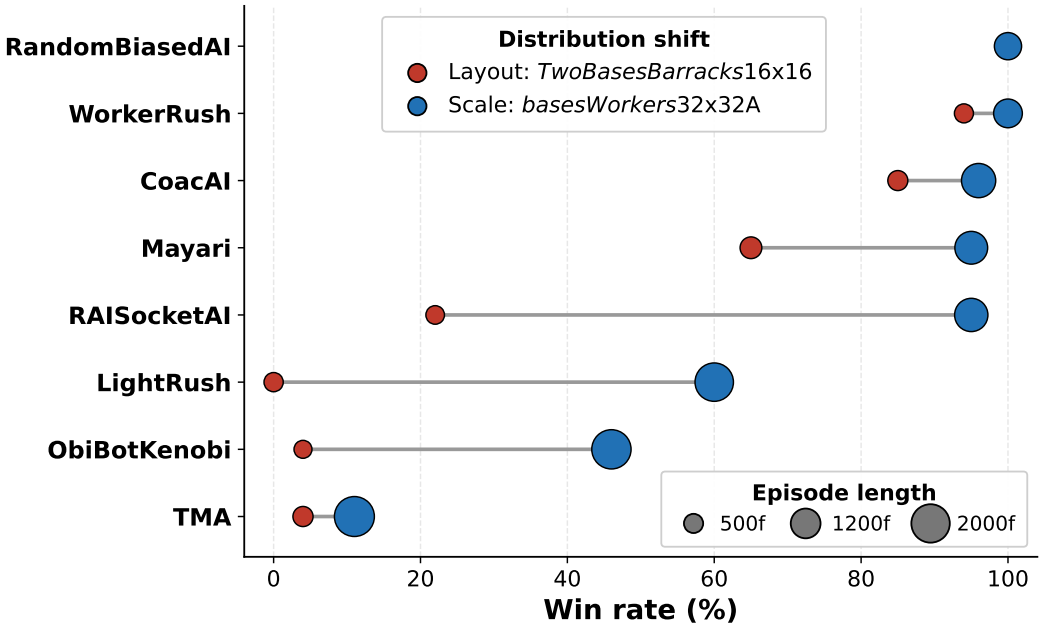


Figure 10.11: Win rate and mean episode length of UECD-Best on the two distribution-shift probes against 8 opponents, over 100 stochastic-sampling games per matchup.

Figure 10.11 reports a striking asymmetry: scale-shift win rate sits well above layout-shift win rate for every opponent, and the gap widens against the strongest agents. The two probes also produce qualitatively different failure modes:

Scale shift is tolerated. On *basesWorkers32x32A*, the agent retains $\geq 95\%$ win rate against 5 of 8 opponents (Random, WorkerRush, Mayari, CoacAI, RAISocketAI). The Ranged-only strategy transfers across map size: unit relationships, action semantics, and reward components are unchanged. Where win rate drops (LightRush, ObiBotKenobi, TMA), episode length grows in proportion: the agent loses, but slowly, and still resists instead of collapsing.

Layout shift is brittle. On *TwoBasesBarracks16x16*, win rate against the strongest opponents collapses (LightRush at 0%, ObiBotKenobi at 4%, TMA at 4%, RAISocketAI at 22%). Episode lengths remain short and uniform (430 to 650 frames), with no correlation to win rate: the agent commits early to a strategy that does not match the new topology. Pre-existing Barracks and the second base alter the optimal opening, and UECD-Best has never seen this configuration.

Taken together, the probes are consistent with a learned representation that encodes map-specific geometric and topological priors rather than transferable strategic invariants: the agent is not perceptually disoriented under either shift, just executing a plan tuned to a different distribution. Several broader conclusions follow plausibly: that single-map training overfits to topology more

than to size, that competitive play on a new layout would benefit from per-map fine-tuning or training on diverse layouts, and that scale generalization may emerge more readily from size-aware architectural choices (padded observations, scale-invariant critics). These claims rest on a single test map per shift type and a single trained agent, however, so they remain working hypotheses rather than established results.

10.7 Behavior-Cloning Warmstart for PPO

RAISocketAI [12] reported that a single DoubleCone PPO model reached ~72% win rate in the tournament round-robin, while a BC-PPO variant (behavior cloning from bot replays as a warmstart for PPO fine-tuning) reached ~88% on the same benchmark without map-specific fine-tuning. This sizable gap motivates a reproduction of the BC-PPO recipe on the architecture and training pipeline of this thesis to test two hypotheses: ***(i)*** BC pre-training accelerates convergence during the early steps of PPO, and ***(ii)*** a BC-initialized PPO reaches a higher final plateau on win rate than from-scratch PPO given equal compute. This section reports a controlled experiment on the single map *basesWorkers16x16A* at 100 M steps.

10.7.1 BC+VF Pre-Training

A script records $\left(\mathbf{o}_t, \mathbf{a}_t, r_{t+1}\right)$ tuples from *bot-vs-bot* games, observed from one designated bot's perspective. Here, the perspective bot is RAISocketAI and the opponent set is composed of RAISocketAI itself, CoacAI, and Mayari, with 100 games against each opponent split evenly between positions P0 and P1. This yields 300 games totaling ~313 000 state-action transitions on *basesWorkers16x16A* (RAISocketAI: ~161 000, CoacAI: ~86 000, Mayari: ~67 000); the imbalance reflects game length, since mirror matches against RAISocketAI last longer.
Following RAISocketAI's formulation, the policy head (cross-entropy on bot actions) and the value head (MSE on discounted reward-to-go) are jointly trained with total loss

$$\mathcal{L}_{\text{BC+VF}} = \underbrace{\frac{1}{|\mathcal{U}_t|} \sum_{k=1}^{7} \sum_{u \in \mathcal{U}_t} -\log \pi_k\left(a_{k,u} \mid \mathbf{o}_t\right)}_{\mathcal{L}_{\text{BC}}} + c_1 \cdot \underbrace{\left(V_\theta(\mathbf{o}_t) - G_t\right)^2}_{\mathcal{L}_{\text{VF}}}, \tag{10.5}$$

where $\mathcal{U}_t$ is the set of active units at step t (the BC loss is per-unit-averaged so that maps with varying army sizes are treated uniformly) and G_t is the reward-to-go of the episode containing t. Despite operating on $(\mathbf{o}_t, \mathbf{a}_t, r_{t+1})$ tuples, this stage is not *offline RL*: it involves no Bellman bootstrapping and no policy improvement, and r_{t+1} enters only through the critic's regression target G_t. The reward is used as a learning signal only later, by PPO. Pre-training the critic this way, rather than from a random initialization, leaves the value head already informative at the start of fine-tuning and avoids a costly re-bootstrap phase.

10.7.2 Comparison Setup

Both runs use the UECD architecture and share the same PPO hyperparameters: seed, learning rate, entropy coefficient, parallel environments, total budget (100 M steps), and map (*basesWorkers16x16A*). They differ only in initialization: "BC+VF → PPO" starts from a BC+VF checkpoint trained for 30 epochs on the dataset above (with a short 100-update learning-rate warmup to protect the pre-trained weights), whereas "From-scratch PPO" starts from random weights. The from-scratch run is the corresponding architecture-ablation run from Section 10.2 for this network.
In-training evaluations run every 10 M steps against five reference opponents (RandomBiasedAI,

WorkerRush, LightRush, CoacAI, Mayari), with 10 games per bot. It is a deliberately light protocol intended to track learning trends, not to support fine pointwise comparisons. The BC+VF checkpoint itself (before any PPO update) is also evaluated against the same five opponents (here with 20 games per bot) to provide a "BC-only" reference.

10.7.3 Results and Discussion

Figure 10.12 summarizes the overall win-rate trajectories.

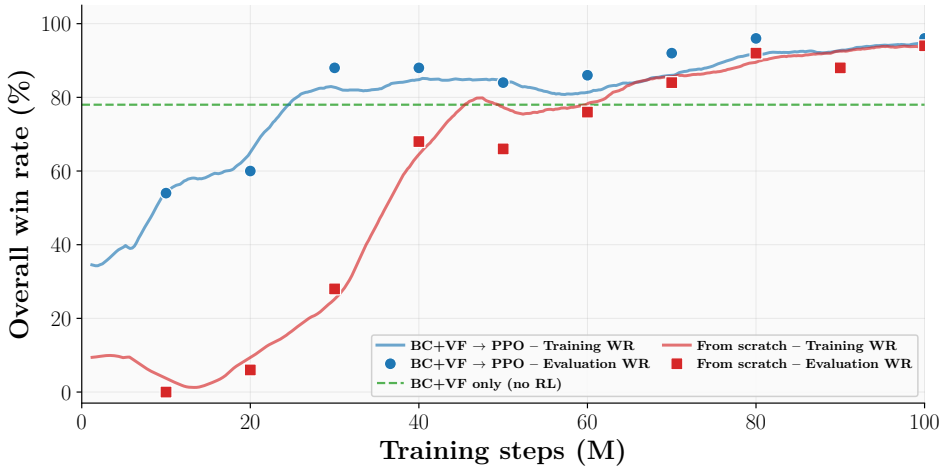


Figure 10.12: Overall in-training win rate against five evaluation bots for BC+VF → PPO (blue) and from-scratch PPO (red) on *basesWorkers16x16A* over 100 M steps. The dashed green line marks the BC+VF-only reference. Solid lines show the (smoothed) per-episode training WR; markers show evaluation WR at 10 M-step intervals (points only, not interpolated).

Three observations stand out. ***(i)*** "BC-only" is already strong: the BC+VF checkpoint alone, trained purely by supervised learning on bot replays, reaches 78% overall win rate against the five evaluation bots, confirming that imitating competent bots produces a non-trivial baseline policy. ***(ii)*** BC strongly accelerates early PPO convergence: at 10 M steps "BC+VF → PPO" has already recovered 54% while "from-scratch PPO" sits at 0% (the from-scratch run has not yet learned basic resource harvesting); the gap is still +54 pp at 20 M and +60 pp at 30 M (88% vs 28%), and over the first 50 M steps the BC run dominates by an average of 41 pp. ***(iii)*** No measurable difference at convergence: beyond 70 M steps both regimes settle in the 84% to 96% range. The 96% vs 94% split at 100 M is well within the binomial noise of a 50-game evaluation ($\sim \pm 6$ pp near 95% win rate) and should not be read as an asymptotic advantage in either direction[3].

The experiment supports hypothesis ***(i)***: BC pre-training drastically accelerates convergence over the first 50 M steps. Hypothesis ***(ii)*** cannot be cleanly evaluated at this protocol's resolution: at convergence the two regimes sit within the noise of the 50-game in-training evaluation, so the result is consistent with parity, with a small residual BC advantage, or with a small from-scratch advantage, but not with a large asymptotic gap. A definitive answer would require the 1 000-game protocol of Section 10.5, which is not run here. The takeaway is therefore that the value of BC-PPO is *compute-efficiency*: under an unconstrained training budget the from-scratch run has time to

[3]With $n = 50$ games per evaluation point, the standard error (SE) of a binomial proportion is $\mathrm{SE} = \sqrt{p(1-p)/n} \approx 0.031$ at $p = 0.95$, giving a 95% CI of $\pm 1.96\,\mathrm{SE} \approx \pm 6$ pp. This reflects the sampling noise of one estimate, not a seed-to-seed variance: each run uses a single seed.

catch up, while under tight compute (e.g. 100 M steps on a single GPU) BC produces a significantly stronger agent at equal wall-clock. BC-PPO is reported here as a standalone experiment and is not used by the single-map UECD agent of Section 10.4, which is trained from random initialization.

10.8 Multi-Map Agent

The generalization probes of Section 10.6 showed that a single-map specialist overfits to one topology. The obvious remedy is to train across several maps at once, but before treating that as a result it has to be shown to work at all on the environment stack built for this thesis. This section serves that narrower purpose. Its goal is not to produce a robust or competition-strong generalist, which the remaining compute budget did not allow, but to establish three things: that a single network can be trained across maps of different sizes through the padded environment of Section 7.5.2, that PLR supplies a usable map curriculum, and that the resulting policy spreads its skill across the pool instead of collapsing onto one layout.

10.8.1 Setup

This experiment produces `UECD-MultiMap`, trained from scratch for 200 M steps on a single RTX 6000 Ada (≈2.3 GPU-days), reusing the UECD architecture and the always-on top-5 feature set of Section 10.3.6 (extended observations, filtered masks + reserved observations, MCW, opponent modeling, GELU) without change. Five conditional features are added: *SPP critic*, *frame stack (4)*, *autoregressive* action sampling, *PAE*, and *PopArt*. The map pool is the five open-competition layouts of size at most 16×16 (*basesWorkers8x8A*, *FourBasesWorkers8x8*, *NoWhereToRun9x8*, *basesWorkers16x16A*, *TwoBasesBarracks16x16*) (Table 6.1, Appendix B). Every 50 updates, the training map is resampled by PLR [108], which weights each of the five maps by $(1 - \mathrm{WR})$ so the agent spends more time on the layouts where it is currently weakest and less on those it already wins (Section 9.6). Self-play, PFSP, and the shaped-to-sparse reward annealing of Section 10.4.2 are all left off. All remaining configuration and hyperparameters follow the baseline of Table 10.1.

10.8.2 Evaluation

The evaluation is a round-robin over the five-map training pool against the same competition field as Section 10.5, joined by the single-map champion `UECD-Best` of Section 10.4, relabeled `UECD-SingleMap` below for the specialist-vs-generalist contrast. The tournament runs 6 000 games across 16 agents and five layouts: the fifteen benchmark agents of Tables 6.2 and 6.3 (except `GridNet`, whose fixed 16×16 critic cannot process the pool's smaller and differently shaped layouts) joined by `UECD-SingleMap` and `UECD-MultiMap`. The analysis goes from the per-map breakdown carrying the main result, to the home-layout reference exposing the trade behind it, and finally to head-to-head behavior aggregated over the pool.

Per-Map Coverage

The per-map win rates (Figure 10.13) carry the main result. Forced across maps it never trained on, `UECD-SingleMap` collapses on the small layouts, 15% on *basesWorkers8x8A* and 26% on *NoWhereToRun9x8*, yet peaks at 94% on its home *basesWorkers16x16A*, a cross-map standard deviation of ≈28 pp. `UECD-MultiMap` stays between 49% and 78% on every map, a spread of ≈11 pp, with no per-map collapse, and finishes well ahead on overall win rate (64% vs 47%). Training on the pool converts the specialist's single tall peak into an even, moderate plateau, which is exactly the behavior the padded environment and PLR were meant to enable.
The collapse on *basesWorkers8x8A* is the most informative single data point. There `UECD-SingleMap` ranks fourteenth of the sixteen agents, with only LightRush and RandomBiasedAI below it. This

nuances the earlier reading of the generalization probes (Section 10.6). There, scale shift looked tolerable because padding preserved the input geometry. What padding cannot supply is strategy: the near-optimal line on 8×8 is an immediate worker rush, a behavior the single-map agent never developed on 16×16. The transferred policy is representationally compatible with the smaller map but strategically unsuited to it, exactly the failure mode the multi-map pool is meant to remove.

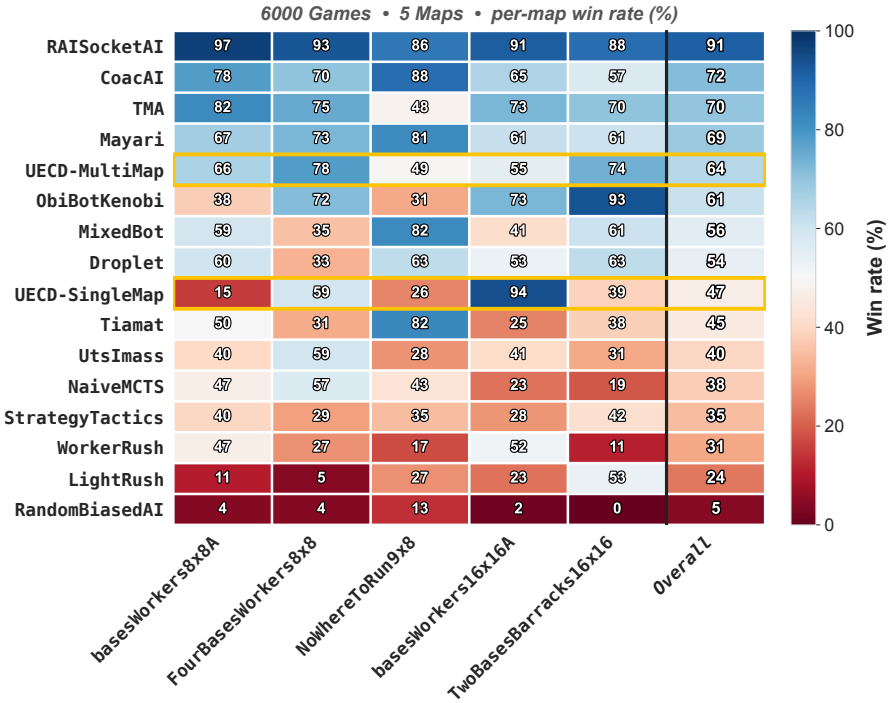


Figure 10.13: Per-map win rate (%) over the five-map pool, agents ordered by overall standing with the *Overall* column set apart on the right.

Peak vs Coverage

The trade behind that plateau is visible on the home map alone (Figure 10.14). Restricted to *basesWorkers16x16A*, UECD-`SingleMap` tops the entire 16-agent field with 141 points, ahead of RAISocketAI (137), confirming that its specialization is genuine peak strength rather than an artifact of the pooled scoring. UECD-`MultiMap` gives up that peak, reaching 82 points for seventh place on the same map, in exchange for the uniform competence it shows everywhere else. The sixth-place finish is itself a reasonable outcome given that the multi-map run was capped at 200 M steps split across five layouts, against 350 M steps spent by UECD-`SingleMap` on this single layout.

Head-to-Head over the Pool

Aggregated over the pool (Figure 10.15), UECD-`MultiMap` wins the majority of its games against every opponent except four: it loses to the three strongest competition bots, RAISocketAI (2%), ObiBotKenobi (22%), and TMA (40%), and splits evenly with CoacAI (50%). It also edges out UECD-`SingleMap` head-to-head (56%). UECD-`SingleMap`, by contrast, only beats the four weakest agents in the field and trails everyone else. What UECD-`MultiMap` gains is robustness of *coverage*: it loses to the same elite bots as the single-map agent, but no longer hands free games to weaker opponents on unfamiliar maps.

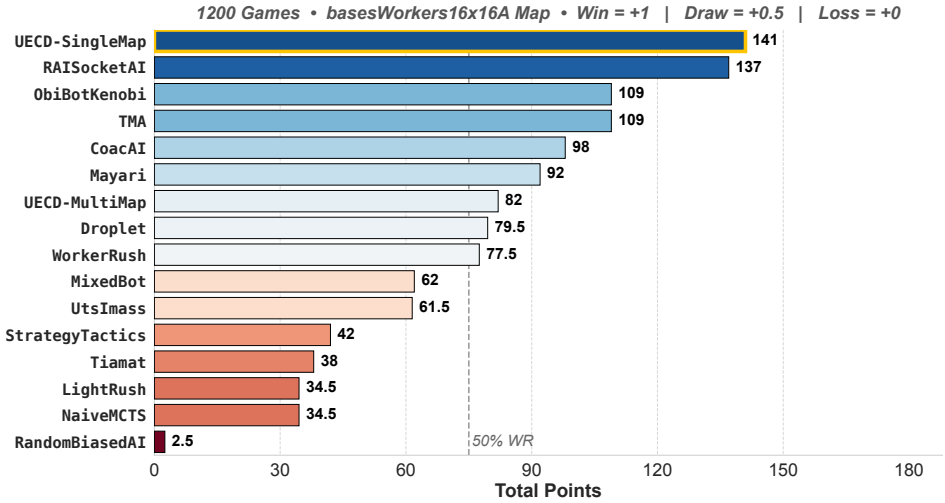


Figure 10.14: Final standings on *basesWorkers16x16A* alone (1 200 games), the single layout UECD-SingleMap (UECD-Best) was trained on.

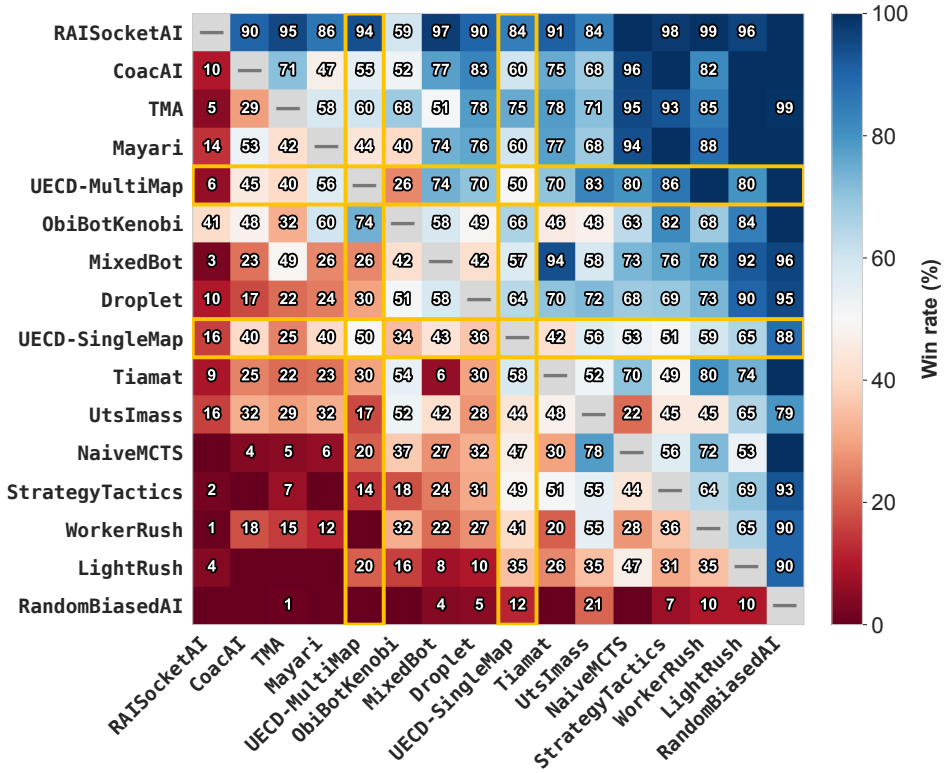


Figure 10.15: Global head-to-head win rates over the five-map pool (6 000 games): cell (i, j) is row i's win rate against column j, ordered by overall standing, diagonal masked (self-play), UECD agents bordered in gold.

10.8.3 Hypothesis Assessment

The hypothesis was threefold: that a single network can be trained across maps of different sizes through the padded environment of Section 7.5.2, that PLR supplies a usable map curriculum, and that the resulting policy spreads its skill across the pool rather than collapsing onto one layout. The evidence above supports each part. A single unchanged network trained across three map sizes (8×8, 9×8, 16×16) absorbed the layout diversity without instability; PLR produced a policy with no per-map blind spot; and `UECD-MultiMap` spreads competence evenly across the pool, in clear contrast to `UECD-SingleMap`. The experiment does not reach competitive peak strength: `UECD-MultiMap` still loses to the strongest bots on every map, so this remains a feasibility-and-validation result, not a competition-grade generalist. The price of coverage is a moderate drop on the single-map reference, in line with the reduced budget and the broader objective. The single-map `UECD-Best` therefore stands as the stronger headline contribution; the multi-map experiment establishes that broadening the training distribution, not changing the architecture, is the route to generalization, the direction taken up in Chapter 13.

11 Discussion

This chapter steps back from Chapter 10's per-experiment analysis to draw out what its results mean once assembled. It reads them along four angles. The first two situate the work against the literature: how the findings depart from prior results on the benchmark, and where they instead confirm established observations (Section 11.1). The last two turn to consequences: implications for practitioners building an RTS agent under a constrained budget (Section 11.2), and what they imply for researchers beyond the specific case of this agent (Section 11.3). Two further angles are addressed elsewhere: the internal correlation across the result sections is already built into the staged structure of Chapter 10, where each experiment feeds the next, and the threats to the validity of the findings are gathered in Chapter 12.

11.1 Findings in Relation to Prior Work

11.1.1 Where the Findings Differ

UECD-`Best` beats RAISocketAI [12], the strongest DRL MicroRTS agent to date, on *basesWorkers16x16A*. The more interesting point is not the ranking but where the gain comes from. In the ablation (Section 10.3), the features that deliver a reliable gain are the ones that change what the policy perceives (extended observations, filtered masks, opponent modeling) or which matchups its gradient focuses on (MCW). They cost almost no parameters. At this scale the lever is not network capacity but the quality of the training signal.

The clearest difference concerns behavior cloning. RAISocketAI [12] reported a BC warmstart raising tournament win rate from ~72% to ~88%. On a single map at 100 M steps, BC accelerates early training but produces no detectable change in the final plateau at this evaluation resolution: both agents converge to the same range (Section 10.7). This is not a contradiction but a different setting: RAISocketAI measures the effect across several maps, whereas the experiment here isolates a single map at a fixed budget. In that setting, BC buys training time, not a higher ceiling.

Finally, the discount-induced rush collapse of Section 10.4.1 has no counterpart in the MicroRTS literature. The formal account of why annealing the shaped reward all the way to zero pushes the policy toward the shortest games extends the classical reward-shaping analysis [87]. The discount-induced preference for shorter episodes is itself classical; the contribution here is its concrete identification, quantification (the ≈92× amplification of Section 10.4.1), and mitigation in the shaped-to-sparse annealing regime, together with the out-of-distribution fragility test.

11.1.2 Where the Findings Agree

Several results confirm known observations. BC speeding up early training matches [111] and RAISocketAI [12]; only the conclusion about the final level differs. The value of relational attention for multi-unit control is in line with the StarCraft II agents that use it [8, 67]. Self-play with PFSP being useful but budget-sensitive echoes the large-scale work [8, 66]: its instability at 50 M steps comes from a checkpoint pool that has no time to mature. Overfitting to topology rather than to map size (Section 10.6) mirrors what the RL generalization literature describes [112, 113]. And the game-theoretic metrics surface structure that a plain win rate hides, which supports their use as evaluation tools [81, 82].

11.2 Implications for Practitioners

For anyone building an RTS agent under limited compute, the results translate into concrete rules:

- Spend the budget on perception and training signal, not on network capacity: the features that deliver a stable gain change what the policy perceives or which matchups it weights, at near-zero parameter cost (Section 10.3); enlarging the backbone does not.
- Do not anneal the shaped reward all the way to zero while $\gamma < 1$ and the policy controls episode length, or it triggers the rush collapse of Section 10.4.1. The clean remedy is a properly trained multi-head value function, like the one in RAISocketAI [12]: implemented correctly, it handles the structural shift by design.
- Use a BC warmstart when wall-clock is the binding constraint: at equal compute it yields a stronger agent, though not a higher ceiling once the budget is large [111].
- With few seeds, choose features on their reliability, not their mean: a steady, low-variance gain is worth more than a larger but unstable one.
- For robustness across maps, broaden the training distribution with a map-and-matchup curriculum [108] rather than redesign the architecture (Section 10.8).

11.3 Implications for Researchers

Several of these rules generalize beyond the particular agent built here:

- The reward collapse is domain-independent: any shaped-to-terminal annealing schedule with $\gamma < 1$ and a policy-controlled horizon is exposed to the same discount-induced shortcut, so the characterization and the three mitigations of Section 10.4.1 transfer to long-horizon RL beyond MicroRTS [87].
- Capacity is not the bottleneck at this scale: the practitioner rule above questions the reflex of enlarging networks for small-to-medium RTS, since where compute is scarce the training signal, not width or depth, is the variable that pays.
- Topology overfitting is a property of the data, not the model: it originates in the training distribution rather than the architecture (Section 10.8), pointing to environment design (PCG and regret-based adversaries [114, 115], or PLR [108]) as the principled route to generalization, not a better encoder.
- Multi-metric evaluation should be the default: each game-theoretic metric exposes structure the others hide (Section 10.5.2), so reporting win rate alone can understate or flatter an agent [81, 82].

12 Limitations

The contributions of Chapters 6 to 10 sit within a defined scope. Section 12.1 enumerates the trade-offs adopted by design to keep the work tractable within an academic budget; Section 12.2 catalogs the residual gaps observed in the experimental results, which the budget allowed neither to explain nor to close, and whose corresponding research directions are developed in Chapter 13. In the language of experimental validity, Section 12.1 concerns internal validity (the controlled, single-run conditions under which each effect is attributed), whereas Section 12.2 concerns external validity (how far the findings generalize beyond the studied setting).

12.1 Deliberate Methodological Choices

MicroRTS rather than a commercial RTS. The choice of MicroRTS was justified in Chapter 1 by the 384 TPUs and 44 training days that AlphaStar required on StarCraft II [8]. Every conclusion of this work is therefore validated on a minimalist abstraction, and the transfer of UECD and its training pipeline to a commercial-scale environment remains an open empirical question.

Full observability. Experiments disable MicroRTS's fog-of-war and stochastic damage (Section 2.2), which isolates the combinatorial and relational difficulty of RTS from the state-estimation problem that partial observability introduces. The architectural axes of Chapter 8 have not been stress-tested against belief-update mechanisms that commercial RTS games require.

Pure reinforcement learning. The agent is trained end-to-end with PPO and never queries a symbolic planner, a search routine, or a language model, which isolates how far DRL can go on MicroRTS as a sole learning signal. Strategy classes naturally expressible as compact rules (rush counters, unit-counter cycles, build-order priors) are therefore rediscovered from scratch by gradient descent rather than injected as priors.

Per-cell action representation. GridNet emits one action vector per occupied cell at every step, which keeps the action space tractable but pins the policy at the unit level: no native mechanism for multi-unit macros, multi-step plans, or hierarchical sub-goals. Temporally extended control would require an additional abstraction layer that this thesis does not investigate.

Single-map focus for the final agent. UECD-`Best` is trained, fine-tuned, and tournament-evaluated on *basesWorkers16x16A* (Section 10.4) to attribute each architectural and training contribution causally through controlled ablation, rather than to maximize breadth at the expense of interpretability. The headline tournament result therefore characterizes one specific topology.

Padded multi-map training. Where RAISocketAI rewrote the Java–Python bridge [12] to dispatch natively on each map's dimensions, this thesis pads every observation to the largest map size (Section 7.5.2). The padded scheme preserves the per-cell GridNet action representation but wastes most of the convolutional work on empty cells (quantified in Section 7.5.2). The decision was driven by the JVM single-startup constraint (Section 7.1.1) and by implementation cost, not by a measured throughput comparison.

Three-seed, 50 M-step ablation. The feature ablation (Section 10.3) tests features at 50 M steps with three seeds each (~19.4 GPU-days total), balancing feature discrimination against compute cost. The per-feature standard deviation often matches the baseline's own (±7.5 pp), making the ranking of moderate-effect features statistically weak; a 5- to 10-seed protocol at 100 M steps would have been more conclusive but exceeded the budget. The same applies to the final agent: UECD-`Best` is reported from a single training run, so the 96.67% headline of Section 10.5 has no seed band either.

12.2 Open Residual Limitations

Sparse reward triggers rush collapse. Removing the shaped reward entirely while keeping $\gamma = 0.99$ amplifies the win signal of a fast rush by a factor of ~92 (Section 10.4.1), and the policy reorganizes around this shortcut at the cost of out-of-distribution robustness. The remedy retained in UECD-`Best` floors the shaped weight at 10% instead of annealing it to zero, which pins the value function to a stable scale but leaves the policy reliant on a hand-tuned residual signal rather than on a clean sparse objective.

Monotone emergent strategy. After fine-tuning, UECD-`Best` converges to a Ranged-only composition (Section 10.4.3) that dominates the tournament pool. The RAISocketAI matchup is still won (65.7% stochastic win rate, 9–1 deterministic) but by a narrower margin, since its fast Light rush exploits the rock-paper-scissors counter (Light beats Ranged at melee range) before the Ranged line is built. Late-stage attempts to diversify the composition (elevated *ProduceLight* reward, Ranged production penalty, higher entropy) failed to shift the policy: the Ranged-only attractor was too entrenched for fine-tuning to dislodge without destroying the learned behavior. Closing the margin would require retraining from an earlier checkpoint, before the composition locks in.

Topology-shift collapse. Following established generalization-probing methodology [112, 113], the probes of Section 10.6 show that UECD-`Best` retains ≥ 95% win rate against five of eight opponents under a scale shift (16×16 → 32×32) but collapses to 4–22% against the strongest opponents under a layout shift (*TwoBasesBarracks16x16*). Single-map training overfits to the specific layout far more than to the specific size. The multi-map study (Section 10.8) shows that broadening the training distribution removes the per-map collapse, but the resulting generalist does not reach competition-grade strength against the strongest bots, so the gap is narrowed rather than closed.

Generalization probed on two shifts only. The probes (Section 10.6) cover one layout shift (*TwoBasesBarracks16x16*) and one scale shift (*basesWorkers32x32A*) at 100 games per opponent, enough to establish a qualitative asymmetry between scale and layout robustness but not to delineate the failure mode systematically: the maps of Chapter 6 beyond *basesWorkers16x16A* and the two shift maps are not probed with UECD-`Best`; three of them are instead used to train UECD-`MultiMap` on its five-map pool (Section 10.8), leaving six entirely unused, and the probes do not control for terrain density, resource asymmetry, or chamber structure independently.

Replay-bound BC warmstart. The BC-PPO experiment (Section 10.7) confirms that supervised pre-training on bot replays [111] accelerates early PPO without lifting the final plateau. The approach does not transfer to the rest of the work: the BC dataset was generated on *basesWorkers16x16A* only, in the standard 29-channel format, without additional features. Extending BC to the multi-map, multi-feature setting would require regenerating replays per map and retraining BC per feature combination, an effort beyond the thesis's time budget.

13 Future Work

This chapter outlines directions for future work, building on the limitations identified in Chapter 12. Section 13.1 proposes four short-term research directions that draw directly on the artifacts of Chapters 7 to 10. Section 13.2 then steps back to a longer horizon, drawing on a cross-game research line on transferable representations and general game systems to outline a program that uses MicroRTS as a stepping stone toward general RTS AI.

13.1 Short-Term Directions

The four directions below each augment the artifacts of this thesis with one additional mechanism: an external language model, a hierarchical decomposition, a symbolic prior module, or an adversarial map generator. The common thread is tractability: each direction prototypes on top of the existing codebase rather than rebuilding it, and each closes at least one limitation named in Chapter 12, from the pure-RL scope to the topology-shift collapse.

13.1.1 Hybrid DRL and LLM Agents

The first direction pairs UECD with an LLM that handles strategic decisions, keeping UECD as the per-tick executor [16, 17]. Three concrete instantiations are immediate. ***(i)*** An *LLM planner* would emit a game plan from the initial map and unit table (rush, expand, or a specific hybrid Light/Ranged composition), which UECD then executes. ***(ii)*** An *LLM opponent modeler* [116, 117] would read the first N frames of play through the per-component statistics exposed by the environment stack of Chapter 7, classify the opponent's strategy in natural language, and condition UECD via an embedding [118]. ***(iii)*** An *LLM curriculum generator* would produce new training scenarios on demand, replacing the fixed scripted pool with a constantly refreshed distribution.
What makes this actionable now is the modular Java–Python bridge of Chapter 7: the structured observations, per-opponent statistics, and replay machinery are exactly the surface a language model needs. The fit with the community is direct as well: the 2026 MicroRTS competition has pivoted to LLM-based agents, so a hybrid LLM-DRL stack turns the agent of this thesis into a reusable building block rather than a final destination. Voyager [119] has shown in Minecraft that an LLM-driven agent can produce coherent long-horizon behavior in a complex game environment, suggesting that the approach extends to RTS. This direction also relaxes the pure-RL scope of Section 12.1: strategy classes naturally expressible as compact priors come from an external language model instead of being rediscovered from scratch by gradient descent.

13.1.2 Hierarchical Policy Decomposition

The second direction questions the assumption that a single network should decide, at every tick and from a global view, what every unit on the map does. A more faithful split mirrors how human players, and real organizations, operate. A *strategic policy* runs on a slow tick, setting army composition and the macro objective (push left, defend the home base, harass the opponent's expansion), while a set of *per-unit micro policies* executes that objective at every tick from local observation only. Each unit gets its own perception and tactical autonomy, as an employee retains

autonomy over the means while the manager fixes the goal. Half of the substrate is already built: the UECD encoder produces per-cell features usable as per-unit observations, and the entity-attention block of Section 8.3 already reasons over units rather than over a flat grid. The missing piece is a principled coupling between the two levels, for which FeUdal Networks [120] and the Options framework [121] provide concrete starting points [122], with macro-action selection in StarCraft [123] the closest RTS precedent. This addresses two limitations of Chapter 12 at once. GridNet's per-cell action representation (Section 12.1) constrains the policy to operate at the single-unit level, leaving no room for multi-unit macros or temporally extended sub-goals. The proposed strategic-tick policy supplies precisely that missing abstraction layer. The same decomposition closes the topology-shift collapse of Section 10.6: a monolithic policy with a global view is structurally biased to overfit to the geometry of its training map, while per-unit policies with local observations generalize more naturally because their input is less correlated with the global topology.

13.1.3 Neuro-Symbolic Integration

The third direction couples UECD with a symbolic reasoning module: a small set of compact, human-readable rules induced from tournament replays, of the form "if the opponent has produced more Heavy than Ranged in the last 60 ticks, prioritize Light." The rules condition the policy either through additional input channels or through a soft gating mechanism on the action head, so the neural network keeps its differentiable expressiveness while the symbolic part contributes interpretability [124] and sample-efficient priors. The artifacts of this thesis make the direction immediately tractable: the tournament replay corpus of Chapter 6 provides the data source for rule induction, and the modular per-component reward function of Chapter 7 offers a natural insertion point for rule-derived priors. The direction also belongs to a broader line of work on the formal description of game knowledge through ludemes [125] and on the use of explicit knowledge to constrain neural learners [126]. It addresses two limitations of Chapter 12: the vulnerability of pure end-to-end policies under distribution shift, which symbolic priors absorb; and the statistical uncertainty imposed by the limited training budget, which knowledge injection offsets.

13.1.4 Adversarial Co-Evolution and PCG

The fourth direction replaces the fixed map pool by a procedural content generator (PCG) [127] that produces maps, starting conditions, and unit-cost variants targeted at the difficulty frontier of the current agent. The agent and the generator co-evolve, with the generator rewarded for finding configurations that are challenging but solvable, following the open-ended-evolution scheme of POET [115] and the regret-based environment design of PAIRED [114]. What makes this tractable in the present codebase is that the padded-observation pipeline and the SPP critic of Chapters 8 and 9 already support variable-size maps without retraining: the only missing component is the generator itself. The limitation this direction targets is the most explicit of the four, the topology-shift collapse of Section 10.6. Instead of hoping that a manually curated map curriculum eventually covers enough geometry to generalize, the agent trains against an adversary whose explicit job is to find topologies it fails on, and robustness becomes part of the training objective rather than a hoped-for side effect.

13.2 Long-Term Vision

Section 12.1 identified the choice of MicroRTS over a commercial RTS as the first deliberate scope limitation, leaving the transfer of UECD and its training pipeline to commercial-scale environments as an open question. Having shown that DRL can produce a competitive agent on this abstraction, this section reframes that limitation as an opportunity: "what can be transferred from MicroRTS

toward larger RTS games, and how?" The four subsections below each adapt one tool from a cross-game research line on transferable representations and general game systems, addressing a specific gap that this thesis raised but could not close. Together they sketch a longer-term program.

13.2.1 Spatial State-Action Features

The first tool comes from [128], which learns local spatial patterns around game actions and uses them as features that generalize across structurally different board games. These features sit deliberately between hand-crafted heuristics and end-to-end deep networks: simple enough to transfer across games with different boards and pieces, rich enough to capture meaningful tactical structure. The analog for RTS is direct, since the convolutional encoder of UECD already encodes spatial-action features implicitly: the composition of units in a 5×5 window around a move, the presence of enemy units within an attack pattern, the proximity of friendly resources. Making these features explicit and transferable would yield an RTS counterpart to those board-game features. The gap this would close is the topology brittleness identified in Chapter 12: a representation that captures local tactical structure independently of the global map geometry is one route to RTS competence that survives a layout change, and the convolutional substrate of UECD is precisely what makes a formal extraction protocol tractable.

13.2.2 Cross-Variant Policy and Value Transfer

The second tool is the transfer protocol of [129], which demonstrates zero-shot and fine-tuning transfer of fully convolutional policy-value networks across structurally different board games. The protocol relies on the semantic alignment provided by Ludii [125]: the source and target games share enough descriptive structure for the same network to be reused or fine-tuned across them. UECD already satisfies the architectural half of the prerequisites: the encoder is fully convolutional, and the SPP critic is scale-invariant, so the same protocol applies in principle. The experiments this enables are concrete and progressively ambitious: pre-train on the open-map pool of Chapter 6 and fine-tune on the closed maps; pre-train on the standard unit table and fine-tune on variants where unit costs or hit points are perturbed; in the longer term, pre-train on MicroRTS and fine-tune on Wargus-style environments[1] with different unit semantics. This direction answers the BC-portability bottleneck of Chapter 12: instead of regenerating supervised data for every new map, the transfer protocol reuses what the policy already knows [130].

13.2.3 Self-Play Distribution Shaping

The third tool is the self-play distribution manipulation of [131], which shows that reweighting episodes by length and injecting exploratory policies in Expert Iteration produces faster and more diverse self-play training. The self-play stack of Chapter 9, built on PFSP-hard sampling over a checkpoint pool, collects a largely undirected mix of trajectories and offers no lever to densify the regions of the trajectory space that matter most for learning. Applying the same mechanism would reweight the pool by episode length, opponent diversity, or terminal-reward variance, spending more of the self-play budget where the gradient signal is most informative. This becomes essential when self-play must span multiple maps or game variants, which is exactly the regime that the previous two subsections target. It also offers a direct lever on the monotone-strategy attractor of Section 12.2: reweighting episodes by opponent diversity disrupts the homogeneous self-play distribution that lets a Ranged-only composition dominate the gradient signal during fine-tuning.

[1] Open-source RTS engine based on Stratagus, GitHub: https://github.com/Wargus/wargus.

13.2.4 A General RTS Game System

The fourth direction is a program rather than a single experiment. Ludii [125] covers turn-based games through ludemes, declarative high-level concepts that make rules concise and shared semantics explicit, which enables the transfer protocol of Section 13.2.2. No equivalent exists for RTS games: real-time semantics such as simultaneous actions, resource economies, and fog-of-war sit outside the current ludeme grammar. This is one reason MicroRTS, Wargus, and the various StarCraft research environments [132, 133] have remained isolated codebases rather than instances of a common formalism. The long-horizon goal is to define an RTS counterpart of Ludii: a declarative game-description language able to describe these environments and future variants in a single substrate, with shared evaluation infrastructure and transfer protocols. The deliverables of this thesis, namely a competitive agent on one specific RTS abstraction, a reproducible tournament framework, and an open-source training pipeline, supply one of the concrete instances such a system would need to support. Such a substrate would be positioned to resolve the limitations identified in Chapter 12: single-map specialization, topology brittleness, and replay-bound warmstart.

"One manuscript completed, countless ideas remaining..." illustration generated with ChatGPT.

14 Conclusion

MicroRTS has been an academic benchmark for RTS AI since 2017. It isolates the core RTS difficulties (combinatorial action space, multi-unit coordination, sparse long-horizon planning) on small grids. Its compute requirements remain tractable compared to full-scale environments such as StarCraft II, where AlphaStar required 384 TPUs over 44 days to reach Grandmaster level [8]. Within this scope, this work targets a DRL agent capable of competing with the state of the art on MicroRTS.

Building on the Gym-μRTS baseline, this work delivers six contributions: ***(i)*** a reproducible CoG-style tournament framework over twelve maps and fifteen reference agents, ***(ii)*** an extended Java–Python bridge with composable wrappers, vectorized self-play, and padded observations for multi-map training, ***(iii)*** the UECD architecture combining multi-scale convolution, entity-level Transformer reasoning, and bottleneck self-attention, ***(iv)*** a modular training pipeline with flag-gated PPO additions, each ablated in isolation on a single map, ***(v)*** a formal analysis of discount-induced reward collapse that explains, and provides mitigations for, the instability of shaped-to-sparse reward annealing, and ***(vi)*** the final `UECD-Best` agent, which integrates the above under a two-phase opponent-curriculum fine-tuning schedule. A paper based on these contributions has been submitted to IEEE CoG 2026 (currently under review).

On the *basesWorkers16x16A* map, `UECD-Best` tops a 19-agent round-robin tournament, finishing first on four of the five ranking metrics: overall win rate (**96.67%**), Copeland score, α-Rank, and average regret. Against RAISocketAI, the primary competitive reference, `UECD-Best` holds a **65.7%** head-to-head win rate under the 1 000-game stochastic protocol (9–1 under the deterministic CoG format) while consuming **9.47** GPU-days and roughly **350 M** environment steps, below the ~500 M steps and ~23.6 GPU-days reported by RAISocketAI for its small-map subset ($\leq$ 16×16), trading that agent's multi-layout breadth for single-layout depth. Against the `GridNet` baseline, `UECD-Best` improves the win rate by **+63.06** pp (33.61% → 96.67%), driven by the combined effect of the UECD architecture, the retained PPO additions, and the two-phase fine-tuning curriculum. Among the retained features, the ablation study identifies four leading individual contributors: extended observations (**+26.3** pp), filtered masks + reserved observations (**+22.6** pp), Matchup Competitiveness Weighting (**+20.4** pp), and opponent modeling (**+19.8** pp).

By bringing per-unit Transformer reasoning, systematic feature ablation, and game-theoretic evaluation together on a controlled MicroRTS map, this work demonstrates that a carefully designed DRL agent can outperform the strongest existing competitor on a single fixed layout, at an academic compute budget and at the cost of cross-layout robustness, and isolates the design decisions that drive this performance, among them the mitigation of a discount-induced reward collapse. Together, these results answer the two research questions raised in Chapter 1.

Beyond the agent itself, this work arrives at a turning point in the MicroRTS competition: after a trajectory from early rule-based agents through search-based and hybrid methods to the first DRL winner in 2023, the focus is now shifting toward LLM-based agents for the 2026 cycle. The open-source pipeline released alongside this thesis is designed to bridge that transition, providing a tested DRL substrate for future hybrid DRL/LLM systems, multi-map extensions, or any of the other directions outlined in Chapters 12 and 13.

APPENDICES

Supplementary materials referenced throughout this master's thesis.

Derivations Underlying PPO

A

This appendix provides two derivations underlying PPO. Section A.1 derives the *policy gradient theorem* (Equation 3.9), showing why $\nabla_\theta J(\boldsymbol{\theta})$ can be estimated from sampled trajectories without differentiating through the environment. Section A.2 then re-expresses this gradient as the *importance-sampled surrogate loss* (Equation 3.15), establishing the equivalence $\nabla \mathcal{L}^{\mathrm{IS}}(\boldsymbol{\theta}) = \nabla J(\boldsymbol{\theta})$ in expectation, which justifies PPO's multiple gradient steps per batch.

A.1 Policy Gradient Theorem

Recall the policy-gradient objective (Equation 3.8):

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0] = \int p_\theta(\tau)\, G(\tau)\, d\tau, \tag{A.1}$$

where $p_\theta(\tau)$ is the probability of trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T)$ under policy π_θ. Differentiating directly yields

$$\nabla_\theta J(\boldsymbol{\theta}) = \int \nabla_\theta p_\theta(\tau)\, G(\tau)\, d\tau, \tag{A.2}$$

which is no longer an expectation and cannot be estimated by Monte Carlo sampling. The *log-derivative identity*

$$\nabla_\theta f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) \cdot \nabla_\theta \log f(\boldsymbol{\theta}) \tag{A.3}$$

restores the expectation form when applied to $f = p_\theta(\tau)$:

$$\nabla_\theta J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log p_\theta(\tau)\, G(\tau)\right]. \tag{A.4}$$

The gradient is again an expectation, and therefore estimable by sampling.

It remains to evaluate $\nabla_\theta \log p_\theta(\tau)$. The trajectory probability factorizes into the initial-state distribution, the policy, and the environment transitions:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)\, p(s_{t+1} \mid s_t, a_t). \tag{A.5}$$

Taking the logarithm turns the product into a sum:

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1} \log p(s_{t+1} \mid s_t, a_t). \tag{A.6}$$

The initial-state and transition terms do not depend on $\boldsymbol{\theta}$ and vanish under ∇_θ:

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t). \tag{A.7}$$

Substituting Equation A.7 into Equation A.4:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G(\tau)\right]. \tag{A.8}$$

A standard causality argument [23] replaces G_0 with the future-only return G_t in each summand, since rewards $r_1, \ldots, r_t$ predate a_t and only add variance to the estimator, yielding Equation 3.9:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t\right]. \tag{A.9}$$

A.2 Importance-Sampled Surrogate Loss

The policy gradient above requires sampling from π_θ. After each update, θ changes and previously collected data is no longer on-policy. To enable multiple gradient steps per batch, the gradient is re-expressed as the gradient of an *importance-sampled surrogate loss.*

Provided $q(x) > 0$ wherever $p(x) > 0$:

$$\mathbb{E}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)} f(x)\right]. \tag{A.10}$$

Applying this to Equation A.9 with $p = \pi_\theta$ and $q = \pi_{\theta_{\text{old}}}$ gives:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[\sum_{t=0}^{T-1} \rho_t(\theta) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t\right], \tag{A.11}$$

with $\rho_t(\theta) = \pi_\theta(a_t \mid s_t) / \pi_{\theta_{\text{old}}}(a_t \mid s_t)$ the importance sampling ratio.

The corresponding surrogate objective is

$$\mathcal{L}^{\text{IS}}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[\sum_{t=0}^{T-1} \rho_t(\theta) \cdot G_t\right]. \tag{A.12}$$

Differentiating with respect to θ (with $\pi_{\theta_{\text{old}}}$ fixed) yields

$$\nabla_\theta \mathcal{L}^{\text{IS}}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[\sum_{t=0}^{T-1} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \cdot G_t\right]. \tag{A.13}$$

Multiplying and dividing by $\pi_\theta(a_t \mid s_t)$ recovers the log-derivative form:

$$\nabla_\theta \mathcal{L}^{\text{IS}}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[\sum_{t=0}^{T-1} \rho_t(\theta) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t\right], \tag{A.14}$$

matching Equation A.11, so $\nabla\mathcal{L}^{\text{IS}} = \nabla J$ in expectation. The surrogate remains differentiable for nearby θ, enabling multiple gradient steps per batch as long as ρ_t stays close to 1, which motivates the PPO clipping mechanism (Section 3.4.3).

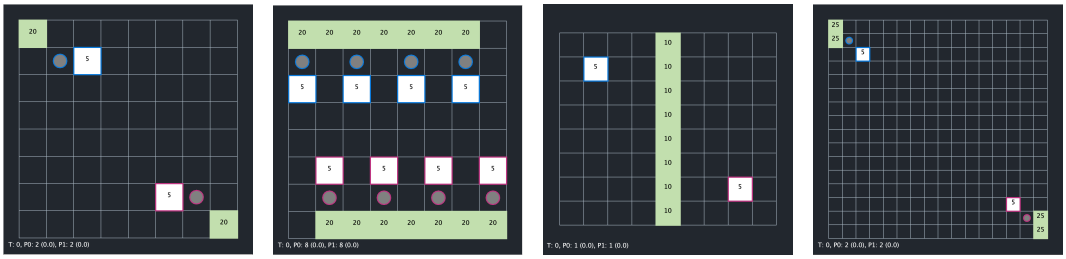
In practice, G_t is replaced by the lower-variance advantage estimator $\hat{A}_t$ (Equation 3.13); the IS argument applies identically since the change of measure does not depend on the integrand. The result is Equation 3.15, the form used in the body[1].

[1]The body uses the PPO shorthand $\mathbb{E}_t[\cdot] \equiv \frac{1}{T}\mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[\sum_{t=0}^{T-1} \cdot\right]$; the constant T is absorbed in the learning rate.

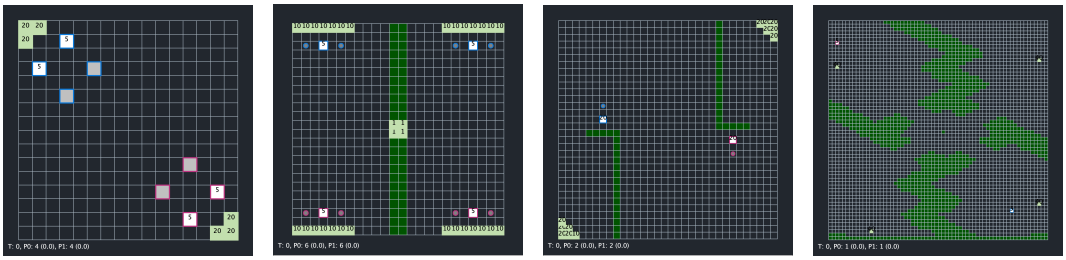
Evaluation Map Visualizations

B

This appendix presents the initial game states of the twelve evaluation maps used in the benchmark tournament (Table 6.1): eight open maps (Figure B.1) and four closed maps (Figure B.2). Maps are ordered by increasing grid size within each category. Visual conventions for units, resources, and terrain follow Section 2.2. All screenshots were captured from the MicroRTS game interface.

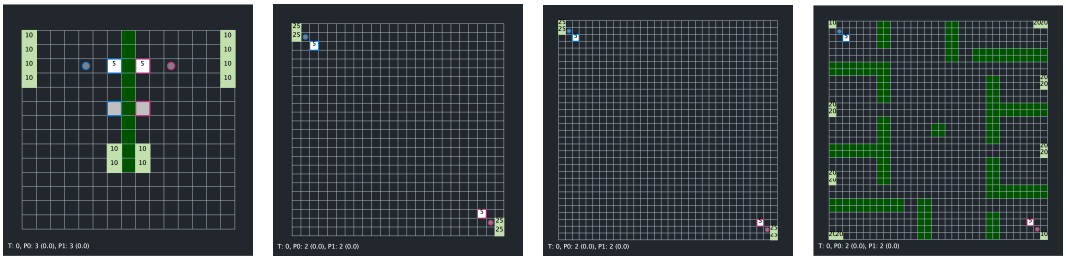


(a) *basesWorkers8x8A* (b) *FourBasesWorkers8x8* (c) *NoWhereToRun9x8* (d) *basesWorkers16x16A*



(e) *TwoBasesBarracks16x16* (f) *DoubleGame24x24* (g) *BWDistantResources32x32* (h) *BloodBath.scmB*

Figure B.1: Initial game states of the eight open evaluation maps, ordered by grid size.



(a) *itsNotSafe* (b) *basesWorkers24x24A* (c) *basesWorkers32x32A* (d) *chambers32x32*

Figure B.2: Initial game states of the four closed evaluation maps, ordered by grid size.

C Game-Theoretic Evaluation Metrics

This appendix details the four game-theoretic metrics introduced in Section 6.4.2 and summarized in Table 6.4. All operate on the same pairwise win-rate matrix $W \in [0,1]^{n \times n}$, where W_{ij} is the win rate of agent i against agent j and n is the number of agents. Draws count as half a win for each side, and the diagonal is set to 0.5 (no self-play). A four-agent worked example in Section C.5 illustrates how the four metrics complement one another on the same matrix.

C.1 Nash Averaging

Nash averaging [81] answers the question: "If an adversary could pick the worst-case opponent, which mix of agents minimizes the damage?"

Pairwise results are treated as a two-player zero-sum meta-game, where each "player" is not an agent but a *selector* choosing which agent to deploy. The payoff matrix $A \in [-0.5, 0.5]^{n \times n}$ centers the win-rate matrix around zero:

$$A_{ij} = W_{ij} - 0.5 \tag{C.1}$$

so that $A_{ij} > 0$ means agent i beats j more than half the time.
The Nash equilibrium is the probability distribution $\mathbf{p}^* \in \Delta^{n-1}$[1] over the agent pool that guarantees the best possible payoff against the worst-case opponent choice. It is found by solving:

$$\max_{v, \mathbf{p} \in \Delta^{n-1}} v \quad \text{s.t.} \quad \sum_i p_i A_{ij} \geq v \;\; \forall j \tag{C.2}$$

Equivalently, $\mathbf{p}$ achieves expected payoff at least v against every opponent j, with v maximized.
The solution $\mathbf{p}^*$ gives the *Nash weights*: agents with $p_i^* > 0$ form the *equilibrium support*, while agents with $p_i^* = 0$ are redundant.
The *Nash score* of each agent measures how well it performs against this optimal mix:

$$s_i = \sum_j A_{ij} p_j^* \tag{C.3}$$

An agent that only beats weak opponents (those with $p_j^* = 0$) scores low, while one that holds up against the strongest mix scores high. This makes Nash averaging robust to pool composition: adding clones or weak agents does not inflate scores.

C.2 α-Rank

α-Rank [82] answers the question: "If agents competed in an evolutionary population, which strategies would survive and dominate over time?"

α-Rank models a finite population of m individuals where, at any moment, all play the same strategy (an agent from the pool). A random mutant appears playing a different strategy and either takes over the population or dies out. The ranking reflects which strategies are hardest to displace and most likely to invade others.

[1] The probability simplex $\Delta^{n-1} = \{\mathbf{p} \in \mathbb{R}^n : p_i \geq 0, \sum_i p_i = 1\}$.

Fitness. Each individual's fitness is its average win rate against the rest of the population. When the population contains k individuals playing strategy j and $(m-k)$ playing i:

$$f_j(k) = \frac{(k-1)\,W_{jj} + (m-k)\,W_{ji}}{m-1}, \qquad f_i(k) = \frac{k\,W_{ij} + (m-k-1)\,W_{ii}}{m-1} \tag{C.4}$$

Fixation probability. Under Fermi selection dynamics, the probability that a single j-mutant takes over a population of i-players is:

$$\rho_{i\to j} = \left(1 + \sum_{k=1}^{m-1} \exp\left(\sum_{\ell=1}^{k} -\alpha\big(f_j(\ell) - f_i(\ell)\big)\right)\right)^{-1} \tag{C.5}$$

where α controls selection pressure. When $\alpha \to 0$, mutations reach fixation at random ($\rho_{i\to j} = 1/m$); when $\alpha \to \infty$, only strictly superior strategies invade.

Markov chain. With uniformly random mutants, the transition matrix between strategies is:

$$T_{ij} = \begin{cases} \dfrac{\rho_{i\to j}}{n-1} & \text{if } i \neq j \\ 1 - \displaystyle\sum_{k\neq i} T_{ik} & \text{if } i = j \end{cases} \tag{C.6}$$

Each off-diagonal entry T_{ij} is the probability that strategy j invades and replaces i. The diagonal T_{ii} is the probability that no mutation succeeds and strategy i persists.

Ranking. The α-Rank score of each agent is its entry π_i in the stationary distribution $\boldsymbol{\pi}$ satisfying $\boldsymbol{\pi}^\top T = \boldsymbol{\pi}^\top$: the long-run fraction of time the population spends playing strategy i. The agent with the largest π_i is ranked first and dominates the evolutionary dynamics. Unlike Nash averaging, α-Rank always produces a unique ranking, even when multiple equilibria exist.

C.3 Copeland Score

The Copeland score [83] answers the question: "How many opponents does the agent beat head-to-head, regardless of margin?"

Each pairwise matchup is reduced to a sign:

$$c_{ij} = \begin{cases} +1 & \text{if } W_{ij} > 0.5 \quad \text{(win)} \\ 0 & \text{if } W_{ij} = 0.5 \quad \text{(draw)} \\ -1 & \text{if } W_{ij} < 0.5 \quad \text{(loss)} \end{cases} \tag{C.7}$$

The Copeland score of agent i is then:

$$C_i = \sum_{j\neq i} c_{ij} \tag{C.8}$$

Scores range from $-(n-1)$ to $+(n-1)$. An agent achieving $C_i = +(n-1)$ beats every opponent and is a *Condorcet winner*. Since margin is discarded (both 51% and 99% wins yield $c_{ij} = +1$), the Copeland score measures dominance breadth.

C.4 Regret

Regret [84] answers the question: "For each opponent, how much worse is this agent compared to the best agent in the pool?"

Adapted from the minimax regret criterion, the regret of agent i against opponent j is the gap between i's win rate and the pool's best win rate against j:

$$r_{ij} = \max_k W_{kj} - W_{ij} \tag{C.9}$$

Two summaries are reported:

$$\text{Average regret}(i) = \frac{1}{n-1} \sum_{j \neq i} r_{ij} \tag{C.10}$$

$$\text{Worst-case regret}(i) = \max_{j \neq i} r_{ij} \tag{C.11}$$

Lower values are better in both cases. An agent with zero average regret is the best choice against every opponent. An agent with high worst-case regret has a single matchup where it performs much worse than the best available alternative, a weakness an adversary could exploit.

C.5 An Illustrative Example

Figure C.1 illustrates this complementarity on a four-agent example whose matrix W contains a non-transitive cycle (A beats B, B beats C, C beats A) together with a fourth agent D whose only victory is over C. The four metrics yield four complementary perspectives on the same matrix.*(i)* **Copeland** ties A and B at the top (+1) and C and D at the bottom (−1), too coarse to separate within either tier despite A beating B and D beating C head-to-head. *(ii)* **Nash** scores each agent by its expected payoff against the equilibrium mixture: A, B, and C tie at exactly 0, the value of the game, forming the non-exploitable frontier, while D falls to −0.08 as a dominated strategy never played at equilibrium. Nash thus separates C from D where Copeland lumps them together, but leaves the frontier itself unranked. *(iii)* **α-Rank** resolves that frontier through simulated evolutionary invasion, ranking $A > B > C$ ($\pi = 0.41, 0.32, 0.21$ for A, B, C; $\alpha = 1$, $m = 50$), as a population of A resists invading mutants more reliably than one of B or C. *(iv)* **Average regret**, the gap to the best response against each opponent, leaves A and B nearly identical (0.12 vs 0.13): B's two suboptimal matchups have small gaps (0.30 and 0.10), whereas A's single one against C is larger (0.35). Each panel thus exposes something the others miss, only α-Rank resolving A's dominance and only Nash distinguishing the non-exploitable C from the dominated D, which motivates reporting the four jointly.

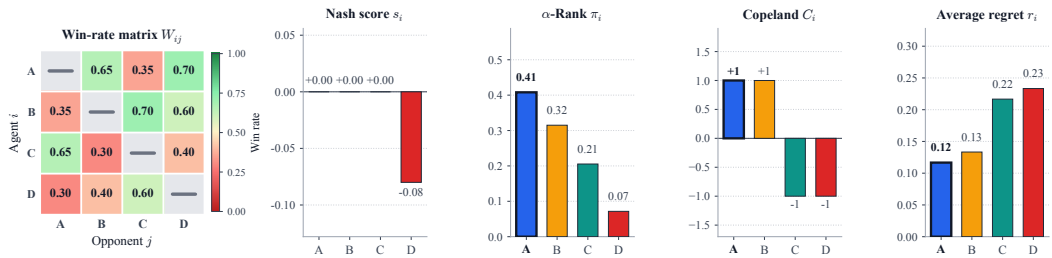


Figure C.1: Complementarity of the four tournament metrics on a small 4-agent example. The pairwise win-rate matrix W (left) is the input to all four metrics (right): Nash score s_i, α-Rank π_i, Copeland score C_i, and average regret r_i.

UECD Architecture: Full Specification

The simplified UECD diagram in Chapter 8 (Figure 8.3) conveys the overall idea of the architecture (modules, axes addressed, scatter targets) but omits channel widths and intermediate tensor shapes for readability. This appendix provides the fully annotated counterpart (Figure D.1), making every block, channel count, and tensor dimension explicit. The intent is to allow the UECD network to be reproduced exactly from the diagram alone, without referring to the source code.

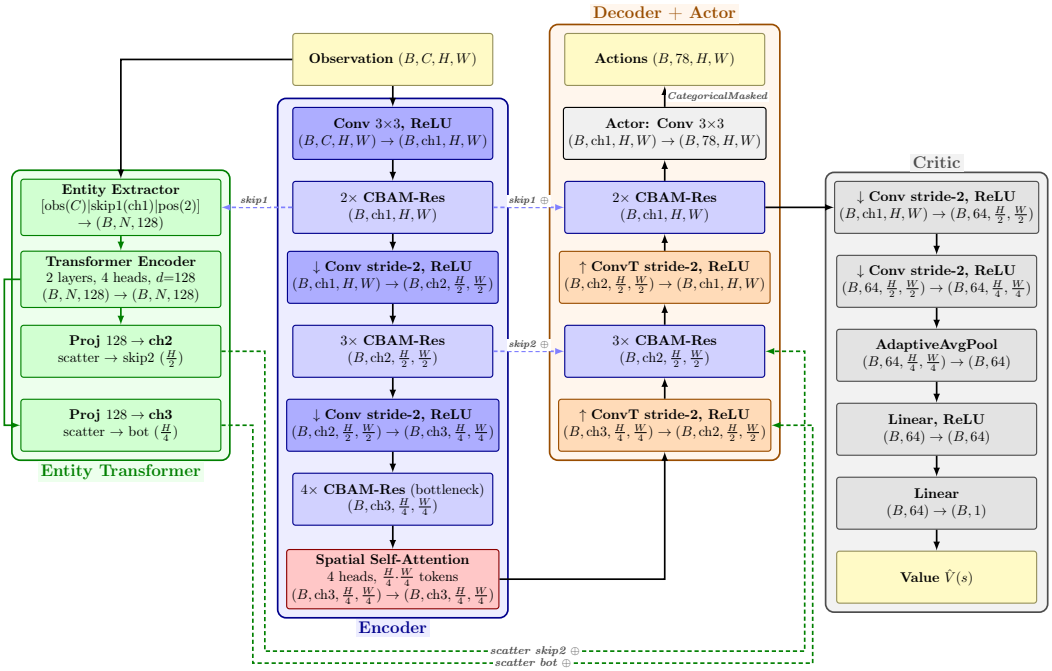


Figure D.1: UECD architecture (annotated, ≈4.7 M parameters at $ch1$=48). Encoder (blue): (2, 3, 4) CBAM-ResBlocks with spatial self-attention at the bottleneck (red). Entity Transformer (green): enriches per-unit representations via self-attention, scatters into the bottleneck and *skip2* (dashed green). Decoder (orange): (3, 2) CBAM-ResBlocks with U-Net skip connections (⊕, dashed blue). All tensor dimensions are given; the architecture is map-size agnostic via the critic's adaptive pool.

Bibliography

[1] D. Aschu, R. Peter, S. Karaf, A. Fedoseev, and D. Tsetserukou, "MARLander: A Local Path Planning for Drone Swarms using Multiagent Deep Reinforcement Learning," in *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Kuching, Malaysia: IEEE, 2024, pp. 2943–2948. DOI: 10.1109/SMC54092.2024.10831294.

[2] A. Krnjaic, R. D. Steleac, J. D. Thomas, G. Papoudakis, L. Schäfer, A. W. K. To, *et al.*, "Scalable Multi-Agent Reinforcement Learning for Warehouse Logistics with Robotic and Human Co-Workers," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2024, pp. 677–684. DOI: 10.1109/IROS58592.2024.10802813.

[3] A. P. Pope, J. S. Ide, D. Micovic, H. Diaz, D. Rosenbluth, L. Ritholtz, *et al.*, "Hierarchical Reinforcement Learning for Air-to-Air Combat," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2021, pp. 275–284. DOI: 10.1109/ICUAS51884.2021.9476700.

[4] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games (T-CIAIG)*, vol. 5, no. 4, pp. 293–311, 2013. DOI: 10.1109/TCIAIG.2013.2286295.

[5] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, *et al.*, "Outracing champion Gran Turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022. DOI: 10.1038/s41586-021-04357-7.

[6] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022. DOI: 10.1038/s41586-021-04301-9.

[7] M. Buro, "Real-time strategy games: A new AI research challenge," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003, pp. 1534–1535. [Online]. Available: https://skatgame.net/mburo/ps/RTS-ijcai03.pdf.

[8] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. DOI: 10.1038/s41586-019-1724-z.

[9] S. Ontañón, "The Combinatorial Multi-Armed Bandit Problem and Its Application to Real-Time Strategy Games," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 9, no. 1, pp. 58–64, 2013. DOI: 10.1609/aiide.v9i1.12681.

[10] S. Ontañón, N. A. Barriga, C. R. Silva, R. O. Moraes, and L. H. S. Lelis, "The First MicroRTS Artificial Intelligence Competition," *AI Magazine*, vol. 39, no. 1, pp. 75–83, 2018. DOI: 10.1609/aimag.v39i1.2777.

[11] S. Huang, S. Ontañón, C. Bamford, and L. Grela, "Gym-μRTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning," in *2021 IEEE Conference on Games (CoG)*, IEEE, 2021, pp. 1–8. DOI: 10.1109/CoG52621.2021.9619076.

[12] S. Goodfriend, "A Competition Winning Deep Reinforcement Learning Agent in microRTS," in *2024 IEEE Conference on Games (CoG)*, Milan, Italy: IEEE, 2024, pp. 1–8. DOI: 10.1109/CoG60054.2024.10645610.

[13] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, *et al.*, *StarCraft II: A New Challenge for Reinforcement Learning*, 2017. DOI: 10.48550/ARXIV.1708.04782.

[14] F. Richoux, "microPhantom: Playing microRTS under uncertainty and chaos," in *2020 IEEE Conference on Games (CoG)*, Osaka, Japan: IEEE, 2020, pp. 670–677. DOI: 10.1109/CoG47356.2020.9231653.

[15] V. Antuori and F. Richoux, "Constrained optimization under uncertainty for decision-making problems: Application to Real-Time Strategy games," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand: IEEE, 2019, pp. 458–465. DOI: 10.1109/CEC.2019.8789922.

[16] S. Cui, S. Xu, A. He, Y. Wang, and B. Xu, "Empowering LLMs with Parameterized Skills for Adversarial Long-Horizon Planning," in *2025 International Joint Conference on Neural Networks (IJCNN)*, Rome, Italy: IEEE, 2025, pp. 1–10. DOI: 10.1109/IJCNN64981.2025.11227831.

[17] W. Ma, D. Xu, S. Lin, H. Zhang, and J. Wang, "Adaptive Command: Real-Time Policy Adjustment via Language Models in StarCraft II," in *Proceedings of the 2024 Sixth International Conference on Distributed Artificial Intelligence (DAI)*, ACM, 2024, pp. 22–30. DOI: 10.1145/3719545.3721029.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[19] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research (JMLR)*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

[20] M. Towers, A. Kwiatkowski, J. K. Terry, J. U. Balis, G. Cola, T. Deleu, *et al.*, *Gymnasium: A Standard Interface for Reinforcement Learning Environments*, Zenodo, 2025. DOI: 10.5281/ZENODO.8127025.

[21] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 157–163. DOI: 10.1016/B978-1-55860-335-6.50027-1.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998. DOI: 10.1016/S0004-3702(98)00023-X.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning), Second edition. Cambridge, Massachusetts London, England: The MIT Press, 2018, ISBN: 978-0-262-03924-6.

[24] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley Series in Probability and Statistics), 1st ed. Wiley, 1994. DOI: 10.1002/9780470316887.

[25] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *Proceedings of the International FLAIRS Conference (FLAIRS)*, vol. 35, 2022. DOI: 10.32473/flairs.v35i.130584.

[26] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957, ISBN: 0-691-07951-X.

[27] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992. DOI: 10.1007/BF00992698.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[29] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, vol. 12, 1999, pp. 1057–1063. [Online]. Available: https://proceedings.neurips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

[30] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992. DOI: 10.1007/BF00992696.

[31] V. R. Konda and J. N. Tsitsiklis, "On Actor-Critic Algorithms," *SIAM Journal on Control and Optimization (SICON)*, vol. 42, no. 4, pp. 1143–1166, 2003. DOI: 10.1137/S0363012901385691.

[32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html.

[33] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," in *International Conference on Learning Representations (ICLR)*, 2016. [Online]. Available: https://arxiv.org/abs/1506.02438.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, 2017. DOI: 10.48550/ARXIV.1707.06347.

[35] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html.

[36] G. Robertson and I. Watson, "A Review of Real-Time Strategy Game AI," *AI Magazine*, vol. 35, no. 4, pp. 75–104, 2014. DOI: 10.1609/aimag.v35i4.2478.

[37] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "RTS AI Problems and Techniques," in *Encyclopedia of Computer Graphics and Games*, N. Lee, Ed., Cham: Springer International Publishing, 2015, pp. 1–12. DOI: 10.1007/978-3-319-08234-9_17-1.

[38] N. A. Barriga, M. Stanescu, F. Besoain, and M. Buro, "Improving RTS Game AI by Supervised Policy Learning, Tactical Search, and Deep Reinforcement Learning," *IEEE Computational Intelligence Magazine (CIM)*, vol. 14, no. 3, pp. 8–18, 2019. DOI: 10.1109/MCI.2019.2919363.

[39] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, *et al.*, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, 2017. DOI: 10.48550/ARXIV.1712.01815.

[40] P. Sweetser and J. Wiles, "Current AI in games: A review," *Australian Journal of Intelligent Information Processing Systems*, vol. 8, no. 1, pp. 24–42, 2002. [Online]. Available: https://eprints.qut.edu.au/45741/.

[41] C. E. Shannon, "XXII. Programming a computer for playing chess," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 41, no. 314, pp. 256–275, 1950. DOI: 10.1080/14786445008521796.

[42] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975. DOI: 10.1016/0004-3702(75)90019-3.

[43] M. Campbell, A. Hoane, and F.-h. Hsu, "Deep Blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002. DOI: 10.1016/S0004-3702(01)00129-1.

[44] S. Ontañón, *Experiments with Game Tree Search in Real-Time Strategy Games*, 2012. DOI: 10.48550/arXiv.1208.1940.

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. DOI: 10.1038/nature16961.

[46] S. Ontañón, "Combinatorial Multi-armed Bandits for Real-Time Strategy Games," *Journal of Artificial Intelligence Research (JAIR)*, vol. 58, pp. 665–702, 2017. DOI: 10.1613/jair.5398.

[47] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games (T-CIAIG)*, vol. 4, no. 1, pp. 1–43, 2012. DOI: 10.1109/TCIAIG.2012.2186810.

[48] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo Tree Search: A review of recent modifications and applications," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023. DOI: 10.1007/s10462-022-10228-y.

[49] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies For Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 04, no. 03, pp. 343–357, 2008. DOI: 10.1142/S1793005708001094.

[50] S. Ontañón, "Informed Monte Carlo Tree Search for Real-Time Strategy games," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Santorini, Greece: IEEE, 2016, pp. 1–8. DOI: 10.1109/CIG.2016.7860394.

[51] R. Moraes, J. Mariño, L. Lelis, and M. Nascimento, "Action Abstractions for Combinatorial Multi-Armed Bandit Tree Search," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 14, no. 1, pp. 74–80, 2018. DOI: 10.1609/aiide.v14i1.13018.

[52] J. R. H. Mariño, R. O. Moraes, C. Toledo, and L. H. S. Lelis, "Evolving Action Abstractions for Real-Time Planning in Extensive-Form Games," *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 33, no. 01, pp. 2330–2337, 2019. DOI: 10.1609/aaai.v33i01.33012330.

[53] N. A. Barriga, M. Stanescu, and M. Buro, "Combining Strategic Learning with Tactical Search in Real-Time Strategy Games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, AAAI Press, 2017, pp. 9–15. DOI: 10.1609/aiide.v13i1.12922.

[54] N. A. Barriga, M. Stanescu, and M. Buro, "Game Tree Search Based on Nondeterministic Action Scripts in Real-Time Strategy Games," *IEEE Transactions on Games (T-G)*, vol. 10, no. 1, pp. 69–77, 2018. DOI: 10.1109/TCIAIG.2017.2717902.

[55] S. Ontañón and M. Buro, "Adversarial Hierarchical-Task Network Planning for Complex Real-Time Games," in *IJCAI'15: Proceedings of the 24th International Conference on Artificial Intelligence*, Buenos Aires, Argentina: AAAI Press, 2015, pp. 1652–1658. [Online]. Available: https://www.ijcai.org/Proceedings/15/Papers/236.pdf.

[56] G. Synnaeve and P. Bessière, "A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 7, AAAI Press, 2011, pp. 79–84. DOI: 10.1609/aiide.v7i1.12429.

[57] B. Weber, M. Mateas, and A. Jhala, "Applying Goal-Driven Autonomy to StarCraft," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 6, no. 1, pp. 101–106, 2010. DOI: 10.1609/aiide.v6i1.12401.

[58] J. Hagelbäck and S. J. Johansson, "The Rise of Potential Fields in Real Time Strategy Bots," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 4, no. 1, pp. 42–47, 2008. DOI: 10.1609/aiide.v4i1.18670.

[59] G. Robertson and I. Watson, "Building behavior trees from observations in real-time strategy games," in *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, Madrid, Spain: IEEE, 2015, pp. 1–7. DOI: 10.1109/INISTA.2015.7276774.

[60] A. Gajurel, S. J. Louis, D. J. Mendez, and S. Liu, "Neuroevolution for RTS Micro," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2018, pp. 1–8. DOI: 10.1109/CIG.2018.8490457.

[61] D. Churchill and M. Buro, "Build Order Optimization in StarCraft," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 7, no. 1, pp. 14–19, 2011. DOI: 10.1609/aiide.v7i1.12435.

[62] D. Harabor and A. Grastien, "Online Graph Pruning for Pathfinding On Grid Maps," *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 25, no. 1, pp. 1114–1119, 2011. DOI: 10.1609/aaai.v25i1.7994.

[63] K. Adil, F. Jiang, S. Liu, W. Jifara, Z. Tian, and Y. Fu, "State-of-the-Art and Open Challenges in RTS Game-AI and Starcraft," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 8, no. 12, 2017. DOI: 10.14569/IJACSA.2017.081203.

[64] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, *A Survey of Deep Reinforcement Learning in Video Games*, 2019. DOI: 10.48550/arXiv.1912.10944.

[65] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, *et al.*, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020. DOI: 10.1038/s41586-020-03051-4.

[66] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, *et al.*, *Dota 2 with Large Scale Deep Reinforcement Learning*, 2019. DOI: 10.48550/ARXIV.1912.06680.

[67] X. Wang, J. Song, P. Qi, P. Peng, Z. Tang, W. Zhang, *et al.*, "SCC: An efficient deep reinforcement learning agent mastering the game of StarCraft II," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021, pp. 10 905–10 915. [Online]. Available: http://proceedings.mlr.press/v139/wang21v.html.

[68] M. Mathieu, S. Ozair, S. Srinivasan, C. Gulcehre, S. Zhang, R. Jiang, *et al.*, *AlphaStar Unplugged: Large-Scale Offline Reinforcement Learning*, 2023. DOI: 10.48550/ARXIV.2308.03526.

[69] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[70] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," in *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, 2015, pp. 2692–2700. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/29921001f2f04bd3baee84a12e98098f-Abstract.html.

[71] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, *et al.*, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 1406–1415. [Online]. Available: http://proceedings.mlr.press/v80/espeholt18a.html.

[72] G. C. Barros E Sá and C. A. G. Madeira, "Deep reinforcement learning in real-time strategy games: A systematic literature review," *Applied Intelligence*, vol. 55, no. 4, p. 243, 2025. DOI: 10.1007/s10489-024-06220-4.

[73] P. Endla, F. J. P, T. V, D. D. Bhavani, M. Tiwari, and T. T. V, "Deep Reinforcement Learning for Real-Time Strategy Games Techniques and Open Challenges," *ITM Web of Conferences*, vol. 76, S. Kannadhasan, P. Sivakumar, T. Saravanan, and S. Senthil Kumar, Eds., p. 01 006, 2025. DOI: 10.1051/itmconf/20257601006.

[74] S. Huang and S. Ontañón, *Comparing Observation and Action Representations for Deep Reinforcement Learning in μRTS*, 2019. DOI: 10.48550/ARXIV.1910.12134.

[75] R. Kelly and D. Churchill, "Transfer Learning between RTS Combat Scenarios Using Component-Action Deep Reinforcement Learning," in *CEUR Workshop Proceedings*, vol. 2862, 2020. [Online]. Available: https://ceur-ws.org/Vol-2862/paper28.pdf.

[76] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, AAAI Press, 2016. DOI: 10.1609/aaai.v30i1.10295.

[77] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 1995–2003. [Online]. Available: http://proceedings.mlr.press/v48/wangf16.html.

[78] S. Ontañón, "Experiments on Learning Unit-Action Models from Replay Data from RTS Games," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 12, no. 2, pp. 9–14, 2016. DOI: 10.1609/aiide.v12i2.12888.

[79] S. Manandhar and B. Banerjee, "Reinforcement actor-critic learning as a rehearsal in MicroRTS," *The Knowledge Engineering Review*, vol. 39, e6, 2024. DOI: 10.1017/S0269888924000092.

[80] M. L. H. D. Lemos, R. E. S. Vieira, A. R. Tavares, L. S. Marcolino, and L. Chaimowicz, "Enhancing deep reinforcement learning for scale flexibility in real-time strategy games," *Entertainment Computing*, vol. 52, p. 100 843, 2025. DOI: 10.1016/j.entcom.2024.100843.

[81] D. Balduzzi, K. Tuyls, J. Perolat, and T. Graepel, "Re-evaluating Evaluation," in *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, Montréal, Canada: Curran Associates, Inc., 2018, pp. 3272–3283. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/cdf1035c34ec380218a8cc9a43d438f9-Abstract.html.

[82] S. Omidshafiei, C. Papadimitriou, G. Piliouras, K. Tuyls, M. Rowland, J.-B. Lespiau, *et al.*, "α-Rank: Multi-Agent Evaluation by Evolution," *Scientific Reports*, vol. 9, no. 1, p. 9937, 2019. DOI: 10.1038/s41598-019-45619-9.

[83] D. G. Saari and V. R. Merlin, "The Copeland method: I.: Relationships and the dictionary," *Economic Theory*, vol. 8, no. 1, pp. 51–76, 1996. DOI: 10.1007/BF01212012.

[84] L. J. Savage, "The Theory of Statistical Decision," *Journal of the American Statistical Association (JASA)*, vol. 46, no. 253, pp. 55–67, 1951. DOI: 10.1080/01621459.1951.10500768.

[85] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, *et al.*, *OpenAI Gym*, 2016. DOI: 10.48550/ARXIV.1606.01540.

[86] H. van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid Reward Architecture for Reinforcement Learning," in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 2017, pp. 5392–5402. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/1264a061d82a2edae1574b07249800d6-Abstract.html.

[87] A. Y. Ng, D. Harada, and S. Russell, "Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping," in *Proceedings of the 16th International Conference on Machine Learning (ICML)*, Morgan Kaufmann, 1999, pp. 278–287. [Online]. Available: https://dl.acm.org/doi/10.5555/645528.657613.

[88] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement Learning with Augmented Data," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/e615c82aba461681ade82da2da38004a-Abstract.html.

[89] M. Stanescu, N. A. Barriga, A. Hess, and M. Buro, "Evaluating real-time strategy game states using convolutional neural networks," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Santorini, Greece: IEEE, 2016, pp. 1–7. DOI: 10.1109/CIG.2016.7860439.

[90] L. Han, P. Sun, Y. Du, J. Xiong, Q. Wang, X. Sun, *et al.*, "Grid-Wise Control for Multi-Agent Reinforcement Learning in Video Game AI," in *International Conference on Machine Learning (ICML)*, 2019. [Online]. Available: https://proceedings.mlr.press/v97/han19a.html.

[91] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., vol. 9351, Cham: Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[92] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, *et al.*, *Attention U-Net: Learning Where to Look for the Pancreas*, 2018. DOI: 10.48550/ARXIV.1804.03999.

[93] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT: IEEE, 2018, pp. 7132–7141. DOI: 10.1109/CVPR.2018.00745.

[94] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional Block Attention Module," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11211, Cham: Springer International Publishing, 2018, pp. 3–19. DOI: 10.1007/978-3-030-01234-2_1.

[95] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, *et al.*, "Attention Is All You Need," in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[96] N. Zwingenberger, *Transformers as Policies for Variable Action Environments*, 2023. DOI: 10.48550/ARXIV.2301.03679.

[97] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)*, 2016, pp. 4898–4906. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/c8067ad1937f728f51288b3eb986afaa-Abstract.html.

[98] S. Huang and S. Ontañón, *Action Guidance: Getting the Best of Sparse Rewards and Shaped Rewards for Real-time Strategy Games*, 2020. DOI: 10.48550/ARXIV.2010.03956.

[99] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)*, 2016, pp. 4287–4295. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/5227b6aaf294f5f027273aebf16015f2-Abstract.html.

[100] E. Imani and M. White, "Improving Regression Performance with Distributional Losses," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, ser. PMLR, vol. 80, 2018, pp. 2157–2166. [Online]. Available: https://proceedings.mlr.press/v80/imani18a.html.

[101] J. Farebrother, J. Orbay, Q. Vuong, A. Ali Taiga, Y. Chebotar, T. Xiao, *et al.*, "Stop Regressing: Training Value Functions via Classification for Scalable Deep RL," in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, ser. PMLR, vol. 235, 2024, pp. 13 049–13 071. [Online]. Available: https://proceedings.mlr.press/v235/farebrother24a.html.

[102] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, vol. 640, pp. 647–653, 2025. DOI: 10.1038/s41586-025-08744-2.

[103] Y. Jin, X. Song, G. Slabaugh, and S. Lucas, "Partial Advantage Estimator for Proximal Policy Optimization," *IEEE Transactions on Games (T-G)*, vol. 17, no. 1, pp. 158–166, 2025. DOI: 10.1109/TG.2024.3408298.

[104] R. Zhang, Z. Xu, C. Ma, C. Yu, W.-W. Tu, W. Tang, *et al.*, *A Survey on Self-play Methods in Reinforcement Learning*, 2025. DOI: 10.48550/arXiv.2408.01072.

[105] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *International Conference on Learning Representations (ICLR)*, 2016. [Online]. Available: https://arxiv.org/abs/1511.05952.

[106] A. van den Oord, Y. Li, and O. Vinyals, *Representation Learning with Contrastive Predictive Coding*, 2018. DOI: 10.48550/ARXIV.1807.03748.

[107] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

[108] M. Jiang, E. Grefenstette, and T. Rocktäschel, "Prioritized Level Replay," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021, pp. 4940–4950. [Online]. Available: http://proceedings.mlr.press/v139/jiang21b.html.

[109] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, 2016. DOI: 10.48550/ARXIV.1606.08415.

[110] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 37, no. 9, pp. 1904–1916, 2015. DOI: 10.1109/TPAMI.2015.2389824.

[111] D. J. Foster, A. Block, and D. Misra, "Is Behavior Cloning All You Need? Understanding Horizon in Imitation Learning," in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. [Online]. Available: https://papers.nips.cc/paper_files/paper/2024/hash/da84e39ae51fd26bb5110d9659c06e13-Abstract-Conference.html.

[112] E. Korkmaz, *A Survey Analyzing Generalization in Deep Reinforcement Learning*, 2024. DOI: 10.48550/ARXIV.2401.02349.

[113] S. Witty, J. K. Lee, E. Tosch, A. Atrey, K. Clary, M. L. Littman, *et al.*, "Measuring and Characterizing Generalization in Deep Reinforcement Learning," *Applied AI Letters*, vol. 2, 2021. DOI: 10.1002/ail2.45.

[114] M. Dennis, N. Jaques, E. Vinitsky, A. Bayen, S. Russell, A. Critch, *et al.*, "Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, Curran Associates, Inc., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

[115] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, *Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions*, 2019. DOI: 10.48550/ARXIV.1901.01753.

[116] J. Guo, B. Yang, P. Yoo, B. Y. Lin, Y. Iwasawa, and Y. Matsuo, *Suspicion-Agent: Playing Imperfect Information Games with Theory of Mind Aware GPT-4*, 2023. DOI: 10.48550/ARXIV.2309.17277.

[117] S. Mallampati, R. Shelim, W. Saad, and N. Ramakrishnan, *Dynamic Strategy Adaptation in Multi-Agent Environments with Large Language Models*, 2025. DOI: 10.48550/ARXIV.2507.02002.

[118] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "FiLM: Visual Reasoning with a General Conditioning Layer," *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 32, no. 1, 2018. DOI: 10.1609/aaai.v32i1.11671.

[119] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, *et al.*, *Voyager: An Open-Ended Embodied Agent with Large Language Models*, 2023. DOI: 10.48550/ARXIV.2305.16291.

[120] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, *et al.*, "FeUdal Networks for Hierarchical Reinforcement Learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017, pp. 3540–3549. [Online]. Available: http://proceedings.mlr.press/v70/vezhnevets17a.html.

[121] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999. DOI: 10.1016/S0004-3702(99)00052-1.

[122] H. Jiang, G. Wang, S. Li, J. Zhang, L. Yan, and X. Xu, "Hierarchical reinforcement learning based on macro actions," *Complex and Intelligent Systems*, vol. 11, no. 6, p. 247, 2025. DOI: 10.1007/s40747-025-01895-9.

[123] S. Xu, H. Kuang, Z. Zhi, R. Hu, Y. Liu, and H. Sun, "Macro Action Selection with Deep Reinforcement Learning in StarCraft," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 15, AAAI Press, 2019, pp. 94–99. DOI: 10.1609/aiide.v15i1.5230.

[124] L. Bae, "Towards Interpretable Reinforcement Learning in Real-Time Strategy Games," *Computational Research Progress in Applied Science and Engineering (CRPASE)*, vol. 8, no. 3, pp. 1–5, 2022. DOI: 10.52547/crpase.8.3.2486.

[125] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne, "Ludii – The Ludemic General Game System," in *ECAI 2020 - 24th European Conference on Artificial Intelligence*, ser. Frontiers in Artificial Intelligence and Applications, vol. 325, IOS Press, 2020, pp. 411–418. DOI: 10.3233/FAIA200120.

[126] M. Silvestri, M. Lombardi, and M. Milano, "Injecting Domain Knowledge in Neural Networks: A Controlled Experiment on a Constrained Problem," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, P. J. Stuckey, Ed., vol. 12735, Cham: Springer International Publishing, 2021, pp. 266–282. DOI: 10.1007/978-3-030-78230-6_17.

[127] K. Sorochan and M. Guzdial, *Generating Real-Time Strategy Game Units Using Search-Based Procedural Content Generation and Monte Carlo Tree Search*, 2022. DOI: 10.48550/ARXIV.2212.03387.

[128] D. J. N. J. Soemers, É. Piette, M. Stephenson, and C. Browne, "Spatial state-action features for general games," *Artificial Intelligence*, vol. 321, p. 103 937, 2023. DOI: 10.1016/j.artint.2023.103937.

[129] D. J. N. J. Soemers, V. Mella, É. Piette, M. Stephenson, C. Browne, and O. Teytaud, "Towards a General Transfer Approach for Policy-Value Networks," *Transactions on Machine Learning Research (TMLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=vJcTm2v9Ku.

[130] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, *et al.*, "A Comprehensive Survey on Transfer Learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021. DOI: 10.1109/JPROC.2020.3004555.

[131] D. J. N. J. Soemers, É. Piette, M. Stephenson, and C. Browne, "Manipulating the Distributions of Experience used for Self-Play Learning in Expert Iteration," in *2020 IEEE Conference on Games (CoG)*, Osaka, Japan: IEEE, 2020, pp. 245–252. DOI: 10.1109/CoG47356.2020.9231589.

[132] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, *et al.*, *TorchCraft: A Library for Machine Learning Research on Real-Time Strategy Games*, 2016. DOI: 10.48550/ARXIV.1611.00625.

[133] P.-A. Andersen, M. Goodwin, and O.-C. Granmo, "Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Maastricht: IEEE, 2018, pp. 1–8. DOI: 10.1109/CIG.2018.8490409.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**