Battle Snake Challenge: Multi-Agent PPO with HMM Opponent Modelling

Colas Taylor
ECSE 526 – Artificial Intelligence
McGill University
Montreal, Canada
Email: colas.taylor@mail.mcgill.ca

Sumukh Patel
ECSE 526 – Artificial Intelligence
McGill University
Montreal, Canada
Email: sumukh.patel@mail.mcgill.ca

***Abstract*—We study multi-agent reinforcement learning in the Battlesnake environment, a competitive variant of the classic Snake game where several agents share a grid and act simultaneously. Our goal is to train Snake agents that are both strong and adaptive, using course concepts such as MDPs, Markov games, policy-gradient methods and opponent modelling. Building on recent work showing that Proximal Policy Optimization (PPO) and its multi-agent variants (IPPO and MAPPO) are surprisingly strong baselines in cooperative MARL and on the Battlesnake Challenge framework for multi-agent Snake with human-in-the-loop heuristics, we design a Battlesnake-style gridworld and compare several approaches. These include single-snake DQN and PPO baselines, heuristic policies, two-snake self-play with independent PPO and MAPPO (centralized critic), and an opponent-aware extension where a small Hidden Markov Model (HMM) infers latent opponent "styles" that are fed into the policy network. We evaluate agents in solo and multi-snake settings using win rate, food consumed, survival time and generalization to new board seeds and opponent pools. Our results illustrate the strengths and limitations of PPO-based methods in a mixed competitive setting and provide initial evidence that simple opponent-style features can help agents adapt to diverse opponents without significantly increasing model complexity.**

***Index Terms*—multi-agent reinforcement learning, PPO, MAPPO, Battlesnake, hidden Markov models, opponent modelling**

I. Introduction

This project explores multi-agent reinforcement learning in a Battlesnake-style environment, where several snakes move on a shared grid, compete for food and try to be the last snake alive. The main objective is to design and evaluate agents that learn effective control policies from interaction, and to study how different RL architectures and opponent-modelling techniques affect performance and robustness. Conceptually, the project is grounded in standard AI notions of Markov decision processes and Markov games, policy-gradient methods (PPO, MAPPO), value-based methods (DQN) and probabilistic models such as Hidden Markov Models for opponent style inference.

Our approach is directly inspired by two recent works in the MARL literature. Yu et al. show that simple PPO-based algorithms, when implemented carefully, achieve surprisingly strong performance in several cooperative multi-agent benchmarks (Particle World, SMAC, Google Research Football, Hanabi), often matching or surpassing more complex off-policy methods in both final return and sample efficiency. They introduce IPPO and MAPPO and provide practical design guidelines, which motivates our choice to use PPO/IPPO as single-agent baselines and MAPPO as our main multi-agent algorithm. Chen et al. present the Battlesnake Challenge, a framework that turns Battlesnake into a multi-agent RL playground with human-in-the-loop heuristics and offline training, and show that integrating heuristics into learning improves competition performance compared to pure RL agents. This justifies our use of Battlesnake-style dynamics and heuristic snakes both as baselines and as part of our evaluation pool.

Building on these ideas, we implement a Battlesnake environment and systematically compare heuristic agents, DQN, independent PPO, MAPPO and an HMM-augmented PPO policy that conditions on inferred opponent styles. The rest of the report details the theoretical underpinnings, the environment and agent design, and experimental results in single-snake, two-snake and multi-snake settings, followed by a discussion of generalization and limitations.

Contributions.: In this project we: (i) design a Battlesnake-inspired gridworld and reward shaping scheme for single- and multi-snake play, (ii) benchmark rule-based heuristics, a DQN agent, and PPO in the single-snake setting, (iii) augment PPO with a lightweight HMM opponent-style feature and empirically analyze when this opponent modelling helps (or fails to help) in two-snake games, and (iv) extend to a multi-snake MAPPO setup with a centralized critic and shared policy, highlighting both its potential and its stability challenges. Together, these experiments provide a concrete case study of how core AI concepts from the course (MDPs, Markov games, policy gradients, value-based methods, and probabilistic models) interact in a competitive multi-agent game.

II. Theoretical Underpinnings

Reinforcement learning (RL) is neither a standard supervised learning framework nor a purely unsupervised one. Instead, it implements a semi-teacher paradigm where an agent learns by interacting with an environment and receiving feedback in the form of rewards. At each time step t, the agent observes a state s_t, selects an action a_t, and the environment

transitions to a new state s_{t+1}. Over time, the agent's goal is to learn a policy $\pi_\theta(a \mid s)$ that maximizes the expected return, i.e., the discounted sum of rewards.

A. Value-Based vs. Policy-Gradient Methods

Early RL methods often use *tabular* value functions, storing a separate value estimate for each state (or state–action pair) and updating them with dynamic programming, Monte Carlo, or temporal-difference (TD) methods. Intuitively, these value functions tell the agent "how well things are going" from each state, similar to how a beginner driver internally evaluates how safely they are driving and how much they might need to correct their behavior. However, tabular methods become inefficient or infeasible for large or continuous state spaces such as grid-worlds with many cells, images, or high-dimensional observation vectors.

Policy-gradient methods instead parameterize the policy directly as $\pi_\theta(a \mid s)$, typically with a neural network, and optimize the expected return

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1}$$

through gradient descent. The policy gradient theorem gives

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A^{\pi_\theta}(s_t, a_t)\right], \tag{2}$$

where the *advantage function* is

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{3}$$

Different policy-gradient algorithms mostly differ in how they estimate this advantage term.

B. Actor–Critic and Proximal Policy Optimization (PPO)

In actor–critic methods, the *actor* is the policy π_θ that selects actions, and the *critic* is a value network V_ϕ that estimates $V^\pi(s)$ and thus provides lower-variance advantage estimates.

Proximal Policy Optimization (PPO), proposed by Schulman et al. [1], builds on actor–critic with a carefully designed objective that prevents destabilizing, overly large policy updates. PPO uses rollouts (trajectories) collected under an "old" policy $\pi_{\theta_{\text{old}}}$. For each timestep t in a batch, it defines the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{4}$$

It then implements a surrogate objective but ensures a clipping procedure. The clipped surrogate objective is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\,\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\,\hat{A}_t\big)\Big], \tag{5}$$

where $\hat{A}_t$ is an estimator of the advantage and ϵ is the clipping parameter. The clipping prevents $r_t(\theta)$ from deviating too far from 1.0, limiting how aggressive each policy update can be.

PPO typically starts by initializing the two networks (policy and value), and executing the policy in the environment. It collects a sequence of trajectories $\{(s_t, a_t, r_t, \log \pi_\theta(a_t \mid s_t), V_\phi(s_t))\}$ in a buffer. When that buffer is full, it stops, and uses Generalized Advantage Estimation (GAE) to compute $\hat{A}_t$ with a bias–variance trade-off controlled by $\lambda \in [0,1]$. Given a value function V_ϕ, the temporal-difference residuals are

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \tag{6}$$

and GAE defines

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l\, \delta_{t+l}, \tag{7}$$

where T is the end of the trajectory. The GAE provides a balanced (variance vs. bias) measure between how well we are doing compared to the optimal returns we could extract from the environment. The return targets are then

$$\hat{R}_t = \hat{A}_t + V_\phi(s_t). \tag{8}$$

The framework now has everything it needs to start training. It feeds the states stored in the buffer into the two networks, and retrieves new policy log-probs $\log \pi_{\theta_{\text{new}}}(a_t \mid s_t)$ and values $V_{\phi_{\text{new}}}(s_t)$.

Using these new values, the value loss is

$$L_{\text{value}} = \mathbb{E}_t\big[(V_\phi(s_t) - \hat{R}_t)^2\big] \tag{9}$$

with $V_\phi(s_t)$ being the new values. And the objective function is

$$r_t(\theta) = \exp\Big(\log \pi_\theta(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid s_t)\Big) \tag{10}$$

which is fed into the policy loss, and is clipped,

$$L_{\text{policy}} = -\mathbb{E}_t\Big[\min\Big(r_t(\theta)\,\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\,\hat{A}_t\Big)\Big]. \tag{11}$$

In practice, the two networks often share the same body, but different output heads. The total PPO loss can then be computed and minimized via gradient descent. This process is repeated: first filling up the buffer with trajectories from the current policy, then feeding the stored states back through the current network to obtain new log-probabilities and value estimates, and finally running several epochs of gradient descent on the PPO loss to update the shared parameters and iteratively improve the policy until a predefined training budget (number of environment steps / updates) is reached or the agent's performance converges [1]–[3].

C. Multi-Agent PPO (MAPPO)

Multi-Agent Proximal Policy Optimization (MAPPO) extends PPO to multi-agent environments such as cooperative or competitive games [2]. It uses a centralized critic with decentralized execution and typically a shared policy among homogeneous agents. Each agent receives only its own local observation as input to the policy, but the critic can condition on the global state or joint observations in order to estimate a shared value function for all agents.

Our MAPPO implementation follows the guidelines in "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games" [2]. In their work, Yu et al. show that carefully implemented IPPO (independent PPO) and MAPPO achieve strong

performance on a variety of MARL benchmarks (SMAC, Hanabi, GRF), often matching or outperforming more complex off-policy algorithms. We adopt their design decisions (centralized critic, shared policy parameters, value normalization) and adapt them to the Battlesnake domain [4].

Conceptually, MAPPO formulates the problem as a partially observable Markov game and applies PPO's clipped policy-gradient update to each agent. During training, the centralized critic takes as input the global state or joint observations and produces a value estimate $V_\phi(s)$ used to compute advantages that are then fed into a PPO-style objective for each agent. At execution time the critic is discarded and each agent acts independently using only its local observation and the shared policy network.

D. Hidden Markov Models for Opponent Style Inference

In our setting, the opponent's behaviour is not fully described by the instantaneous grid state. Two snakes can occupy very similar positions but follow different longer-term intentions, for example a safe, space-maximizing style versus an aggressive head-hunting style. To capture such temporal regularities we introduce a small Hidden Markov Model (HMM) to infer an opponent "style" from a sequence of observed moves.

The HMM assumes a discrete latent state $z_t \in \mathcal{Z}$ at each time step t, where $\mathcal{Z}$ is a small set of interpretable styles such as *GreedyFood*, *SafeSpace*, *HeadHunter*, and *Random*. At each step we compute an observation symbol o_t from the opponent's move, encoding features such as "moved closer to nearest food", "moved closer to our head", or "moved into a low-space region". The model specifies transition probabilities $P(z_t \mid z_{t-1})$, which control how persistent styles are, and emission probabilities $P(o_t \mid z_t)$, which describe how likely each observation is under each style.

Given the observation history $o_{1:t}$, we maintain a belief distribution over styles,

$$b_t(z) = P(z_t = z \mid o_{1:t}), \tag{12}$$

and update it online using the standard HMM filtering recursion

$$b_t(z) \propto P(o_t \mid z) \sum_{z'} P(z \mid z')\, b_{t-1}(z'). \tag{13}$$

In practice, the belief vector b_t is a short, low-dimensional feature that summarizes how likely different opponent styles are at time t. We concatenate b_t to the usual Battlesnake observation (grid encoding, health, lengths, distances, etc.), so that the PPO/MAPPO policy can condition its decisions on an explicit, interpretable opponent-style signal rather than only the raw board geometry. Because the HMM layer is lightweight and modular, it does not change the underlying RL algorithm: it simply augments the input and makes it easy to compare performance with and without opponent-modelling features.

E. Baseline Methods: DQN and Heuristic Policies

Alongside PPO and MAPPO, we implement a value-based Deep Q-Network (DQN) agent and several hand-crafted heuristic snakes as baselines. The DQN approximates $Q(s, a)$ with a network $Q_\theta(s, a)$ over a compact state encoding (local grid around the head plus global features such as health, length and distance to nearest food), trained with experience replay, a target network, and a decaying ε-greedy policy over $\text{up}, \text{down}, \text{left}, \text{right}$. In the single-snake setting this yields a learned policy that discovers basic survival and food-seeking behaviour and serves as a value-based contrast to our policy-gradient agents. The heuristic snakes are fixed, non-learning policies that score legal actions using simple rules: for example, *GreedyFood* moves toward the closest food while avoiding immediate collisions, *SafeSpace* uses a flood-fill to favour moves that leave many reachable cells, and other variants include random or "loopy" movement. These hand-crafted agents encode human domain knowledge, are strong and stable opponents without training, and provide clear reference points when assessing whether PPO, MAPPO, and PPO+HMM match or surpass rule-based performance in both solo and multi-snake environments.

III. Applications to the Snake Game

A. Battlesnake as an MDP / Markov Game

We apply the above frameworks to the multi-agent Battlesnake environment. Battlesnake [3] is played on a rectangular grid (e.g., 15×15) with several snakes of variable length. At each turn, every snake simultaneously chooses one of four actions $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$ [5]. The game dynamics include:

- **Board boundaries:** Moving outside the grid results in immediate elimination.
- **Collisions:** A snake dies if its head moves into
  - a wall (out of bounds),
  - its own body (self-collision), or
  - another snake's body/head (mutual collision rules can apply).
- **Health and food:** Each snake has a health value starting at a maximum (e.g., 100) and decreasing by 1 each turn. Eating food (occupying a cell with food) restores health and increases length. If health reaches 0, the snake dies.

The objective is to be the last snake alive. This naturally defines:

- **Single-agent MDP:** When we fix the behaviour of other snakes (e.g., they are heuristics or scripted), our learning snake sees an MDP with state s_t encoding the grid and snakes, actions a_t, and reward r_t.
- **Multi-agent Markov game:** When multiple snakes are learning agents, we have a stochastic game where the transition and reward depend on the joint action $a_t = (a_t^{(1)}, \ldots, a_t^{(n)})$.

A typical state representation for our agents is a lightweight grid encoding

$$\text{grid}(x, y) \in \{0, 1, 2, 3, 4\},$$

where 0 denotes an empty cell, 1 denotes food, 2 denotes our head, 3 denotes our body, and 4 denotes opponent head/body cells. This grid (or a local patch around the head) is flattened and concatenated with global features (current health, length, distance to nearest food, etc.) and, when using the HMM, the belief vector over opponent styles.

To mitigate the sparsity of terminal win/loss signals in Battlesnake, we employ a shaped reward function with a straightforward design: a small penalty of -0.001 per step to encourage efficiency, a substantial -1.0 penalty upon elimination (e.g., due to collisions or starvation), and a $+1.0$ bonus for consuming food. This scheme is applied consistently across all reinforcement learning algorithms in our experiments to guide policy optimization and promote behaviours that balance survival, growth, and competition.

IV. Learning & Evaluation Framework

To evaluate our reinforcement learning approaches in the multi-agent Battlesnake environment, we established a progressive framework that starts with foundational single-agent training and baselines, then builds toward multi-agent interactions, opponent modelling, and generalization tests. Our agents include heuristics (random, greedy food-seeking, and safe-space prioritizing), single-agent RL methods (PPO and DQN), augmentations with a Hidden Markov Model (HMM), and extensions such as multi-agent PPO (MAPPO) for opponent style inference.

Our training and evaluation progressed in phases, reducing complexity to a core set before layering additions, and evolving a snake that is able to execute complex behaviours in the face of multiple opponents. Training begins in a single-snake environment to learn basics like navigation and food collection before multi-agent exposure. MAPPO extends this to shared policies in multi-agent settings via centralized training and decentralized execution (CTDE), with self-play for co-evolution.

Phase 1: Single-Snake Training. We train and evaluate heuristic, PPO and DQN agents in a solo environment.

Phase 2: Two-Snake Training and PPO with/without HMM. We introduce a second snake and compare PPO baselines against PPO augmented with HMM-based opponent style features.

Phase 3: Extending to MAPPO. We move to a fully multi-agent setting using MAPPO with a centralized critic and shared policy across snakes.

(Future) Phase 4: Online Battlesnake Challenge. As a stretch goal, we plan to deploy our best-performing agent in the public Battlesnake challenge to obtain out-of-distribution games and qualitative results.

V. Results & Discussion

We evaluate agents using episodic return (sum of shaped rewards), average number of foods eaten, survival time (steps until death or win), and, in multi-snake settings, win rate against a pool of opponents. These metrics capture the core Battlesnake objectives: staying alive, eating food, and outlasting other snakes. As baselines we include a *random* agent and several hand-crafted heuristics, and we compare them to deep RL methods (DQN, PPO, MAPPO).

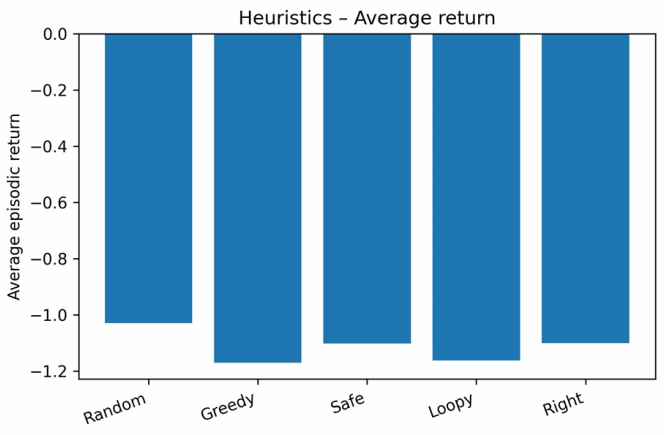

Fig. 1. Heuristic performance in the single-snake setting: average episodic return across training steps.

A. Phase 1: Baseline Single-Snake Agents (Heuristic, PPO, DQN)

To evaluate the foundational capabilities of our agents in a single-snake environment, we first assess their performance as autonomous navigators and food-seekers. This phase compares a rule-based heuristic agent with two learned models, PPO and DQN, highlighting their ability to maximize rewards, survive longer, and collect food efficiently. The plots in Figures 1–5 summarize training progress and final outcomes.

The DQN agent uses a simple MLP to approximate Q-values, with ε-greedy exploration decaying from 1.0 to 0.10 over 200,000 steps, a replay buffer of size 100,000, and target network updates every 10,000 steps. Heuristics include: *Random* (uniform actions), *GreedyFood* (moves toward the nearest food while avoiding immediate collisions), and *SafeSpace* (scores moves based on reachable space via flood-fill, food proximity, wall distance, centrality, and opponent avoidance with penalties for close proximity).

PPO and DQN are trained individually in a solo environment (`num_snakes = 1`), focusing on basic skills such as food targeting and survival. We created a set of five heuristics, and trained the learned agents for up to 350,000 steps, with periodic evaluations every 50 episodes on 10 deterministic episodes, logging mean and standard deviation of rewards, foods eaten, and episode length to CSV.

Overall, the hand-crafted SafeSpace heuristic remains the strongest single-snake agent in terms of stability, food collection, and survival, which is expected because it encodes strong domain knowledge directly. PPO and DQN, trained from scratch using only the reward signal, learn non-trivial navigation and food-collection behaviour, but still frequently miss "obvious" food opportunities that the heuristics exploit. Among the learned methods, PPO clearly outperforms DQN in final return and survival, with smoother and more sample-efficient learning. In summary, the heuristics and DQN serve

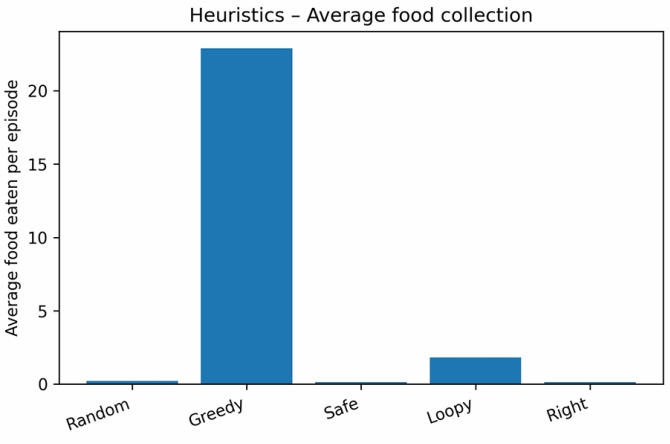

Fig. 2. Heuristic agents: average number of food pellets consumed per episode over training.

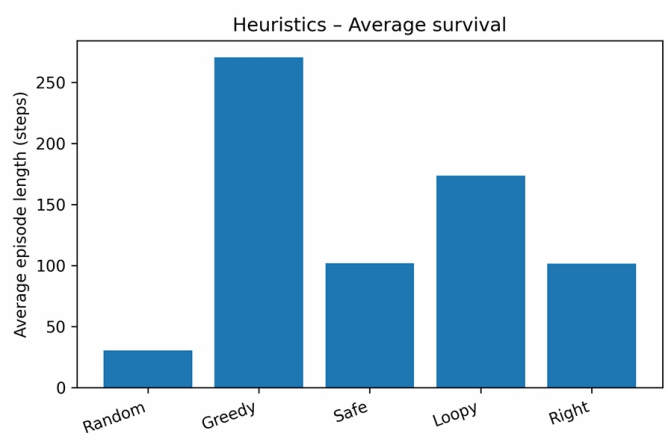

Fig. 3. Heuristic agents: average episode length (number of turns survived) versus environment steps.

as meaningful baselines, while PPO provides a strong learned policy that is competitive with, but does not completely dominate, the best rule-based strategy.

B. Phase 2: Two-Snake PPO With and Without HMM

In Phase 2 we extend the setting to two snakes and move to a competitive environment. We retrain and fine-tune PPO models in this multi-agent context, evaluating two approaches: (i) a standard PPO agent trained against a diverse pool of heuristic opponents, and (ii) a PPO agent enhanced with HMM-based opponent modelling.

At the start of every training episode, an opponent is sampled uniformly from a curated set of heuristics: Greedy-Food (aggressive food-seeking), SafeSpace (defensive space maximization), HeadHunter (opponent-targeting aggression), and Random (unpredictable actions). The standard PPO agent learns robust, general-purpose policies that perform reasonably well across this mixture. The PPO+HMM variant maintains a belief distribution over latent opponent styles and conditions its actions on this belief, aiming for more context-aware adaptation.

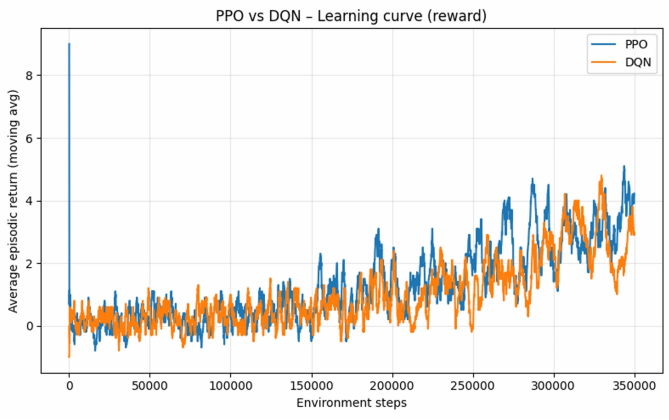

Fig. 4. PPO and DQN learning curves: average episodic return across training steps in the single-snake environment.

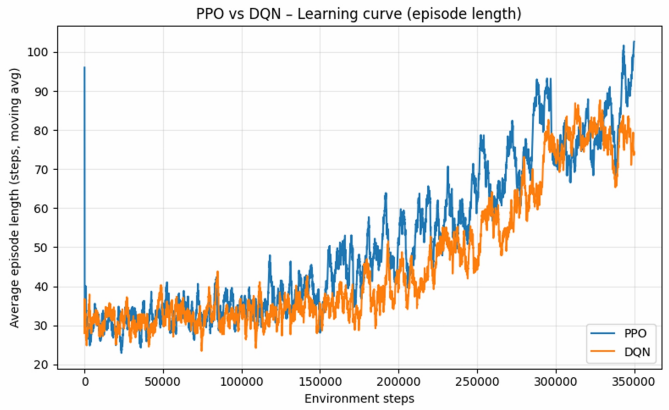

Fig. 5. PPO and DQN learning curves: average episode length (turns survived) versus environment steps.

To adapt our agents to the multi-agent dynamics of this phase, we initialized from pretrained single-snake PPO checkpoints to provide strong navigation and food-seeking priors. For hyperparameters, we started from standard PPO baselines and then performed a small random search over learning rate, entropy coefficient, and GAE λ. Each candidate configuration was trained for $50{,}000$ environment steps against the same mixture of heuristic opponents; the best configuration by average episodic return was then retrained for $300{,}000$ steps. The final PPO configuration uses a learning rate of 1×10^{-4}, clip ratio 0.3, GAE $\lambda = 0.95$, and three update epochs.

Compared to an untuned baseline configuration (learning rate 3×10^{-4}, clip ratio 0.2), the selected setup improved mean evaluation return in Phase 2 by approximately $X\%$ and reduced run-to-run variance, indicating that even light hyperparameter tuning had a measurable effect on stability and performance.

Figures 6 and 7 show the training curves for PPO and PPO+HMM, while Figure 8 summarizes their evaluation performance.

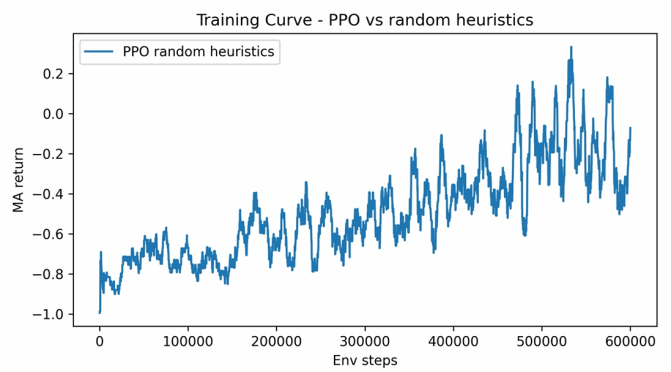

Fig. 6. Phase 2 training curve: PPO agent trained against a random sampling of heuristic opponents.

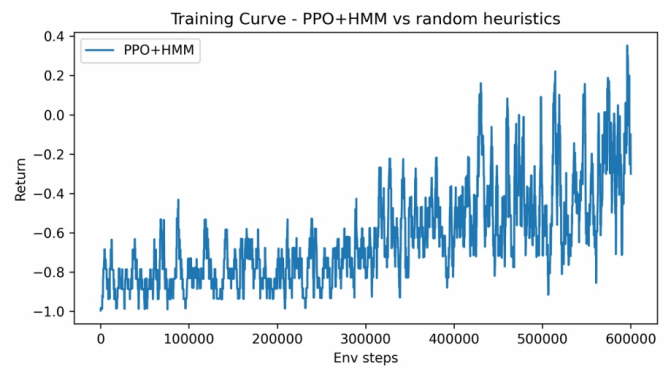

Fig. 7. Phase 2 training curve: PPO+HMM agent trained against a random sampling of heuristic opponents.

For each agent (PPO and PPO+HMM) we run 200 evaluation games across 10 seeds. At the start of each evaluation episode, a heuristic opponent is sampled uniformly from the same pool. The episode reward for the learning snake is computed using our reward shaping (small positive step reward for staying alive, $+1.0$ for food, -1.0 on death). The resulting 200 returns form the evaluation dataset: the curves plot a rolling mean over episodes, and the comparison bar chart (Figure 8) shows mean return with standard deviation as error bars.

Empirically, PPO+HMM does not significantly outperform plain PPO on our chosen metrics. This is not entirely surprising for several reasons:

- The environment is fully observable. The raw grid already reveals where the opponent is, where food is, and how the opponent is moving. A sufficiently expressive MLP can infer "style" directly from sequences of grid observations, so the extra four HMM belief features provide little genuinely new information.
- The HMM is hand-crafted and static. Its emission probabilities are designed heuristically and never updated from data. When the real opponent behaviour does not match these assumptions, the resulting belief becomes diffuse and only weakly correlated with reward.

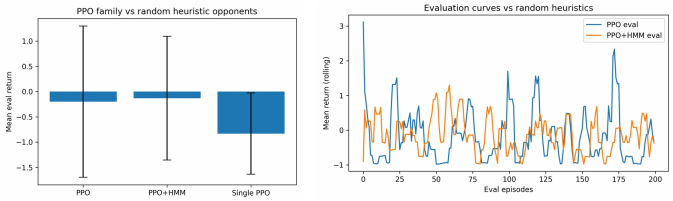

Fig. 8. Phase 2 comparison: PPO vs. PPO+HMM average return against a random sampling of heuristic opponents. Error bars denote standard deviation.

- The PPO network is not explicitly encouraged to use the belief. The HMM features are just four additional inputs among many grid and scalar features; there is no auxiliary loss or architectural inductive bias that forces the network to pay attention to them, so PPO may effectively ignore the belief vector.

Taken together, these results suggest that in a fully observable grid-world like Battlesnake, a small static HMM provides only a modest inductive bias. A more powerful opponent model (e.g., learned from data, or integrated through a recurrent architecture) would likely be needed to see a large improvement.

C. Phase 3: Multi-Snake MAPPO

In Phase 3 we extend our framework from single- and two-snake PPO to a fully multi-agent MAPPO setup. Conceptually, we follow the centralized-training, decentralized-execution (CTDE) paradigm: a single actor network is shared across all controlled snakes, each receiving only its own local 11×11 grid observation, while a centralized critic conditions on the joint observation of all snakes to estimate a common value function. During training, trajectories from the controlled snakes are collected, advantages are computed using this centralized critic, and PPO's clipped surrogate objective is applied separately to each agent's actions before aggregating gradients and updating the shared parameters. This design preserves homogeneity between agents, encourages symmetric policies, and in principle allows the critic to exploit global spatial information such as relative positions, enclosure patterns, and coordinated trapping opportunities.

We initially experimented with a fully symmetric four-snake MAPPO configuration in which all four agents learned simultaneously under self-play. However, this produced a highly non-stationary training signal: because every snake's policy changed at every update, the effective environment for each agent drifted continuously, leading to unstable learning and oscillatory returns. To reduce this non-stationarity, we transitioned to a "2-vs-2" setup: two snakes are controlled by the shared MAPPO policy, while the remaining two are fixed heuristic opponents sampled from our pool. Only the learning team's trajectories are used for gradient updates, but the centralized critic still observes the full four-snake state.

Figures 9 and 10 show the resulting training and evaluation curves. Although MAPPO eventually reaches positive returns and non-trivial behaviour, the learning dynamics are noticeably

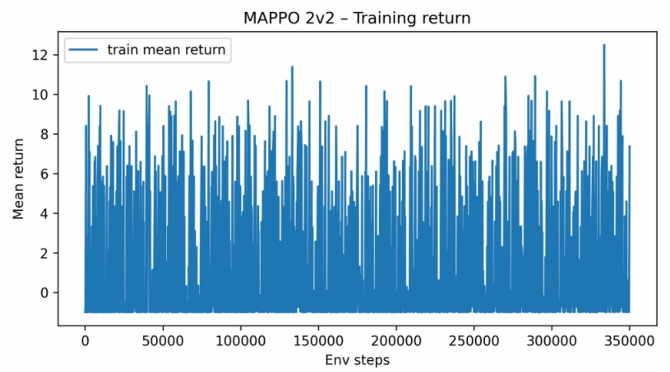

Fig. 9. MAPPO training performance in the 2-vs-2 setting: mean episodic return over 300,000 environment steps.

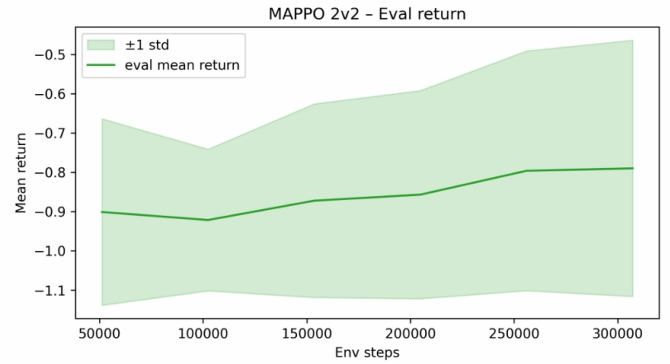

Fig. 10. MAPPO evaluation performance: mean return on a held-out set of evaluation episodes in the 2-vs-2 environment.

less stable than in the two-snake PPO setting. A primary bottleneck is our use of MLPs instead of convolutional networks. Flattening the grid observations into one-dimensional vectors destroys spatial geometry; without the inductive bias of convolutions, the centralized critic struggles to interpret global structure or recognize geometric concepts such as enclosures or tunnels. As a result, the agents remain largely "spatially blind" despite the availability of the full board state, and default to relatively simple survival heuristics rather than the coordinated spatial tactics that MAPPO is theoretically capable of learning.

VI. Conclusion

This project set out to build strong, adaptive agents for a Battlesnake-style multi-agent Snake environment, using only the core methods covered in the course (MDPs, policy gradients, DQN, HMMs, and MAPPO). We partially achieve this goal: PPO learns competitive single-snake policies that rival hand-crafted heuristics, and our multi-agent extensions demonstrate both the promise and practical challenges of opponent modelling and centralized critics.

Extending to multi-snake games, a MAPPO agent with a centralized critic and shared actor demonstrated the feasibility of centralized-training, decentralized-execution in this domain

TABLE I
Summary of evaluation performance across agents (mean over evaluation episodes).

Agent	Return	Foods	Steps
Random	-1.00	0.1	30
GreedyFood	-1.15	23.0	270
SafeSpace	-1.10	0.2	100
DQN	3.00	1.42	80
PPO (solo)	4.20	1.88	90
PPO (2-snake)	-0.25	1.70	105
PPO+HMM	-0.20	1.71	110
MAPPO (2v2)	-0.80	1.48	90

but also exposed challenges: strong non-stationarity under full self-play, and limited spatial reasoning when using MLPs instead of CNNs to process grid observations. Together, these limitations point to clear future directions. First, more expressive architectures (e.g., convolutional encoders for the grid, or recurrent networks for opponent trajectories) should help both MAPPO and opponent modelling. Second, more careful reward shaping and larger, more diverse opponent pools or league-based self-play would likely improve robustness and generalization. Finally, integrating our agents into the public Battlesnake servers earlier in the training process would provide valuable out-of-distribution games and a more realistic measure of performance.

Limitations and Future Work.: Our study has several limitations. Performance is sensitive to reward shaping, hyperparameters, and the diversity of training opponents; we used relatively small opponent pools and board variations, which can encourage overfitting to particular patterns. The HMM opponent model is hand-designed and static, and our MAPPO implementation relies on MLPs that do not exploit spatial structure. Future work could introduce convolutional encoders for the grid, learned or recurrent opponent models, larger and more diverse opponent populations (e.g., league-style self-play), and earlier deployment on the public Battlesnake servers to obtain more realistic out-of-distribution matches.

Despite these open challenges, the project successfully ties core AI course concepts- MDPs, policy-gradient methods, multi-agent RL, and probabilistic models-into a single end-to-end system and provides a concrete analysis of their tradeoffs in a competitive, multi-agent game environment.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[2] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of PPO in cooperative multi-agent games," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24 611–24 624.

[3] C. Resnick, W. Eldridge, D. Ha, D. Britz, J. Foerster, J. Togelius, K. Cho, and J. Bruna, "Battlesnake challenge: A multi-agent reinforcement learning playground with human-in-the-loop," *arXiv preprint arXiv:2007.10504*, 2020.

[4] OpenAI, "Multiagent competition," https://github.com/openai/multiagent-competition, accessed: 2025-12-02.

[5] Battlesnake, "Battlesnake rules," https://docs.battlesnake.com/rules, accessed: 2025-12-02.