Learning to Survive: Organism Evolution in a Simulated Ecosystem Using Reinforcement Learning

EE641: Deep Learning Systems, Final Project Report

Anni Li (annili@usc.edu), Zeli Liu (zeliliu@usc.edu)
University of Southern California

December 15th, 2024

1. Introduction

The evolution of organisms within ecosystems has long been a fascinating research area, offering valuable insights into survival strategies, adaptive behaviors, and interactions with dynamic environments. In this project, we simulate the survival and evolution of AI-controlled organisms in a grid-based ecosystem using reinforcement learning. By modeling the complexities of biological systems, the study explores how artificial agent can learn and adapt to maximize their survival under changing environmental conditions.

In this project, an AI agent is designed to survive in a simulated environment containing food, predators, and shelters, as illustrated in Figure. 1. The agent relies on reinforcement learning to optimize decision-making processes, managing internal states such as hunger, health, and attack level while interacting with external elements like environmental features.

Although time-constraint and experience-limit for us, our project successfully demonstrates how reinforcement learning can replicate certain survival behaviors in human sense. We also discuss the importance of RNN in the model architecture, challenges and potential improvements.

2. Literature Review

Reinforcement Learning (RL) has achieved notable success in solving sequential decision-making problems, especially with the integration of deep learning methods [8]. However, RL still faces several challenges, including sample inefficiency, sparse reward signals, and difficulties in adapting to dynamic environments [2]. Evolutionary strategies (ES) have been proposed to address these limitations by using population-based search methods to optimize policies in RL tasks [10].

Recent research has explored the combination of RL with evolutionary computation, known as Evolutionary Reinforcement Learning (EvoRL). EvoRL techniques maintain a population of agents to improve exploration and handle hyperparameter sensitivities more robustly [6]. Methods like novelty search and quality diversity algorithms have been effective in sparse reward environments, encouraging agents to discover diverse and high-performing strategies [4]. These approaches help agents overcome challenges such as deceptive rewards and adapt to dynamic scenarios.

Co-Reyes et al. [3] introduced ecological reinforcement learning to address continuous, non-episodic interactions between agents and environments. Their work highlighted the importance of dynamic and shaped environments for learning in sparse reward settings. This aligns with our project, where the agent learns to survive in a grid-based ecosystem with evolving food, threats, and shelters.

Liu [7] explored cooperative behaviors in multi-agent RL games, showing that agents can learn collaborative strategies in competitive and shared-resource environments. Their work emphasizes the importance of

Figure 1: Environment

interaction-based learning, which aligns with our goal of modeling adaptive behaviors in resource-constrained ecosystems.

Morita and Hosobe [9] demonstrated how human-like behaviors could be achieved in game agents by combining Deep Q-Networks (DQN) with biologically-inspired constraints. Sun et al. [12] introduced DQN-based intelligent decision-making in wargame environments, leveraging prior knowledge to enhance learning efficiency. Sebastianelli et al. [11] applied Deep Q-Learning to the classic Snake game, highlighting the effectiveness of RL in learning spatial navigation and food-seeking strategies. Similarly, our project involves learning spatial survival behaviors, including movement and resource optimization, within a grid-based ecosystem.

Peng et al. [1] introduced an environment comprehension mechanism to achieve safer deep RL training, emphasizing the role of environment awareness in agent performance. In our work, partial observability and dynamic elements drive the agent to make informed decisions based on its limited surroundings.

Informed by these studies, our project models a dynamic survival environment where agents learn to balance internal states and external interactions. By designing a resource-constrained ecosystem with evolving challenges, this work showcases how RL can simulate adaptive behaviors in dynamic settings.

3. Environment

3.1 Grid

The environment for this project is a grid-based survival game implemented as a custom OpenAI Gymasium, named `SurvivalGameEnv`. It is designed to simulate complex survival scenarios where an agent intereacts with food, threat, and shelters to maintain its internal states such as health, hungry, and attack power. The environment is represented as a square grid, the size of which can be configured (default is 16×16). Each cell in the grid may can belong to one of the following categories:

- **Walls (1):** Impassable boundaries around the grid.
- **Food (2):** Can increase agent's hunger level when consumed.
- **Threats (3):** Movable threats that can engage in combat with the agent(Figure. 2).
- **Agent (4):** The position of the agent itself.
- **Caves (5):** Shelters where the agent can recover health.
- **Empty (0):** If nothing.

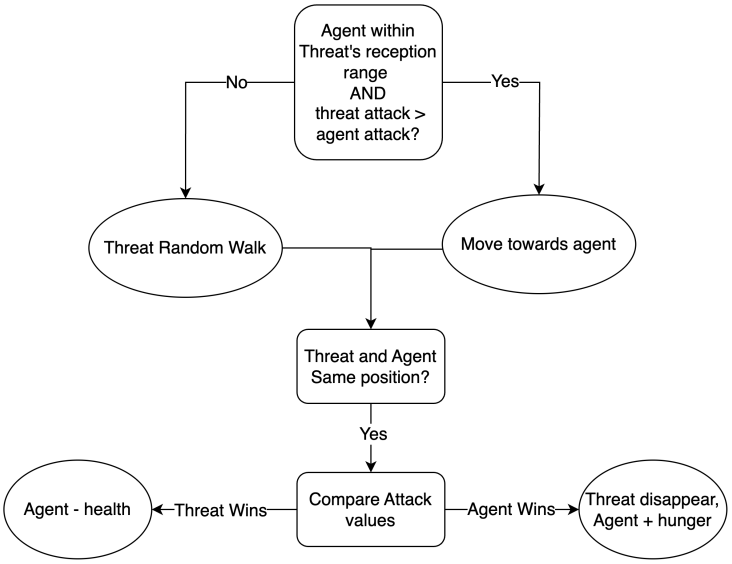

Figure 2: Interaction between Agent and Threats

At each episode, food, threats, and caves are generated randomly and assigned uniform random values within configurable ranges. Threats either move randomly or toward the agent when within their perception range. They pose a risk to the agent by reducing its health upon contact. Caves are stationary shelters where the agent can recover health, offering a strategic point for survival. As time progresses, the agent's hunger level decays, penalizing it with health loss if hunger falls below a critical threshold. When a food or threat is consumed, a new one is generated at a random location with random value assigned, ensuring that their total number remains constant throughout the game to provide continuous interaction source for agent. The episode terminates when the agent's health reaches zero or the maximum number of steps is exceeded.

3.2 Agent

The agent, positioned randomly at the start with uniform random attack value assigned at each episode, can take one of five discrete actions in each step:

- **0:** Stay in the current position.
- **1:** Move up.
- **2:** Move down.
- **3:** Move left.
- **4:** Move right.

The agent has a limited observation range, typically a square region centered around its current position (default is 9×9) . Within this range, the agent perceives the environment through multi-channel observations, including one-hot encoded grid item categories (walls, agent, caves), and two other channels of normalized food values and threat attack values. This restricted perception mimics real-world scenarios where decisions must be made based on partial information.

3.3 Rewards

Rewards in the environment are designed to encourage survival and adaptive behavior. We initially made a harsh reward design, mainly included negative rewards under situations where agent is not maintaining a high health, and this resulted in an unsatisfied performance of the model. According to [5], we realized that negative rewards can lead to premature termination of learning episodes in reinforcement learning, therefore we re-designed the reward to ensure the positive reward is achievable through the learning process and also rewards are more directed to desirable actions. For significant situations, such as being alive when reached maximum step and health goes zero, rewards are comparatively higher absolute values to indicate what's most important in the game. In addition, relatively small amount of positive rewards are granted for staying alive, consuming food, enter cave, and defeating threats. Conversely, reward penalties are applied for health loss, critical hunger level, and low health level. These reward signals aim to guide the agent toward optimal strategies for survival.

4. Model Design

Our agent is implemented using a Deep Q-Network (DQN) architecture designed to handle both spatial input and internal agent states. The model processes the grid-based environment through two convolutional layers (CNN), concatenating the output with the agent's internal state (health, hunger, attack), and optionally incorporates temporal information using a recurrent neural network (RNN) implemented by Long Short-Term Memory, and finally feed into fully connected layers, as illustrated in Figure. 3

4.1 Input Representation

The input to the model consists of two main components. First, the spatial input is represented as a multi-channel grid where each channel encodes specific environmental information. These include one-hot encodings for walls, the agent's position, and caves. Additional channels are values for food energy and threat attack levels are normalized. Together, these channels form a tensor with dimensions $(batch\_size, 5, k, k)$, where k is the size of the observation scope of the agent.

After convolution, the agent's internal states, including health, hunger, and attack level, are provided as a separate feature vector concatenated with the output of the conv layers. Each value is normalized to ensure consistent scaling.

4.2 Convolutional Layers and Feature Fusion

The grid input is passed through two convolutional layers to extract spatial features. The first convolutional layer applies 16 filters with a kernel size of 3×3 and a padding of 1, followed by a ReLU activation function. The second convolutional layer further refines these features using 32 filters with the same kernel size and activation function.

To incorporate the agent's internal state, the flattened spatial features are concatenated with the normalized internal state vector. This step ensures that the model considers both the external environmental observations and the agent's internal dynamics for decision-making.

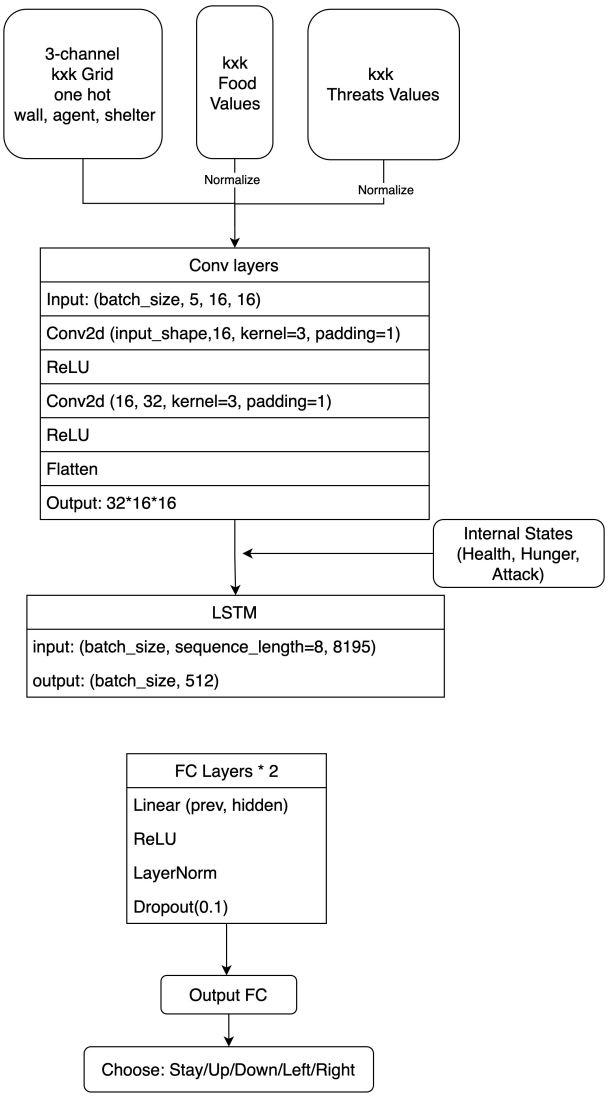

Figure 3: Model Structure

4.3 Temporal Feature Handling

Inspired by [13], the concatenated features can optionally pass through a Long Short-Term Memory (LSTM) layer for agent to capture temporal dependencies in its observations and actions. This recurrent layer allows the model to utilize historical context, which is particularly useful in dynamic environments where decisions depend on prior states. Note that this step is optional and we will analyze its importance in the result section.

4.4 Fully Connected Layers

The output from the LSTM, or directly from the concatenated features, is fed into a series of fully connected layers. These layers consist of linear transformations, followed by ReLU activation functions, normalization layer, and dropout layer to improve generalization and prevent overfitting.

The final output layer maps the refined features to Q-values corresponding to each action in the agent's action

space. During training, the agent selects actions using an epsilon-greedy strategy to balance exploration and exploitation. The value of ϵ decays exponentially over time, starting from 1.0 and approaching a lower bound of 0.05. During inference, the agent selects the action with the highest Q-value to maximize its expected reward.

5. Reinforcement Learning

The training process of our reinforcement learning agent follows the Deep Q-Network (DQN) paradigm, which combines Q-learning with neural networks and experience replay. The overall structure involves the **policy network**, the **target network**, a **replay buffer**, and the interaction between the agent and the environment.

The **policy network** is the core decision-making module. It approximates the Q-value function, which maps the current state of the agent to Q-values for each possible action. The Q-values represent the agent's expected cumulative reward for taking a particular action in the given state, followed by an optimal policy. During training, the policy network selects actions using an ϵ-greedy strategy, balancing exploration and exploitation. Initially, the agent explores more frequently to collect diverse experiences, and over time, ϵ decays, encouraging the agent to exploit learned policies.

The agent interacts with the **environment** to gather experience. At each step, the agent observes the current state, selects an action, and receives the next state, reward, and a termination signal. This interaction generates transitions of the form $(state, action, reward, next\_state, done)$, which are stored in the **replay buffer**. The replay buffer serves as a memory bank that allows the agent to sample experiences randomly during training. This approach reduces the temporal correlation between consecutive experiences, stabilizing the training process.

To ensure stable learning, the DQN framework employs a **target network** alongside the policy network. The target network is identical to the policy network but is updated less frequently, often by copying the weights of the policy network at fixed intervals. This separation between the networks helps prevent instability caused by rapidly changing target Q-values during training.

During training, the agent samples mini-batches of transitions from the replay buffer. For each transition, the **Q-value loss** is MSE loss calculated based on the Bellman equation:

$$\text{Loss} = \text{E}\left[\left(Q(s,a) - \left(r + \gamma \max_{a'} Q_{\text{target}}(s',a')\right)\right)^2\right]$$

where $Q(s,a)$ is the predicted Q-value from the policy network, r is the immediate reward, $Q_{\text{target}}(s',a')$ is the Q-value estimate from the target network for the next state, and γ is the discount factor. The gradients of the loss are backpropagated through the policy network to update its weights. Gradient clipping is applied to ensure numerical stability during optimization.

The training process is illustrated in Figure 4. The policy network selects actions based on the current state, while the environment generates rewards and transitions. These transitions are stored in the replay buffer, which is later sampled to compute the loss. The policy network uses this loss to adjust its parameters, and the target network periodically synchronizes its weights with the updated policy network.

The training continues for a configurable number of episodes. During evaluation, the agent's performance is measured using average rewards, episode lengths, and comparisons to baseline agents, such as random agents or earlier versions of the model. By incorporating experience replay, a target network, and a structured loss function, the reinforcement learning framework ensures a stable and efficient learning process, enabling the agent to adapt to dynamic and partially observable environments.

6. Results and Analysis

We analyze the results of the training and evaluation of our reinforcement learning agent under two configurations: with and without temporal features (RNN). The performance metrics include training rewards,

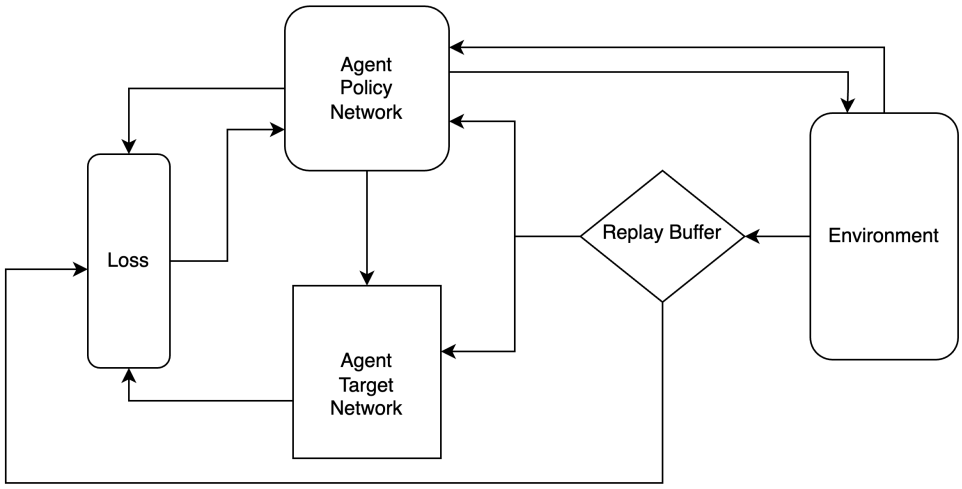

Figure 4: Training Process

evaluation rewards, agent health, and training loss. We also compare the model with some novel approaches, like random decisions. These results allow us to analyze the strengths and limitations of each model configuration.

6.1 Training and Evaluation Rewards

Figure 5 and Figure 6 compares the reward of DQN agent compared to baseline agent (DQN at episode *max episode // 10*) and a random action selector, using two different configs. Comparing two plots, trained DQN with RNN shows more reward peaks than trained DQN without RNN, and the trained agent using RNN comparatively has in general higher imrpoved values of reward in comparison to the baseline and random agents, indicating the agent's improved ability to generalize and adapt to the environment by leveraging temporal dependencies. This suggests that incorporating an RNN allows the agent to utilize historical observations, leading to better decision-making in dynamic scenarios.

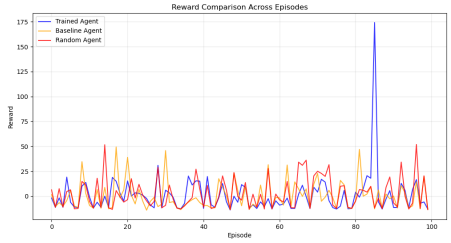

Figure 5: Reward Comparison, without RNN

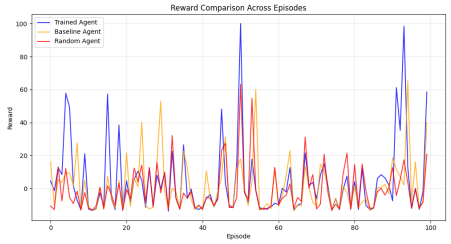

Figure 6: Reward Comparison, with RNN

6.2 Agent Health and Training Loss

With an extremely large maximum time step per episode, the agent's health at the end of each episode is expected to remain relatively static during training. This setup allows the agent to explore both simple and challenging survival scenarios effectively.

The training loss curves further highlight the differences in convergence between the two configurations.

Without the RNN (Figure 7), the loss decreases initially but plateaus early, reflecting suboptimal policy learning. However, in the RNN setup (Figure 8), the loss decreases more gradually and stabilizes at a much lower value, suggesting that the model continues to improve over time by learning meaningful temporal patterns in the data.

Note that the plots for the agent health when episode ends continuously shows zero, which we suspect as some bugs in the source code, because in the training rewards there are some high values indicating strong reward granted, which can be possible that the agent gets those reward values through living till the end.

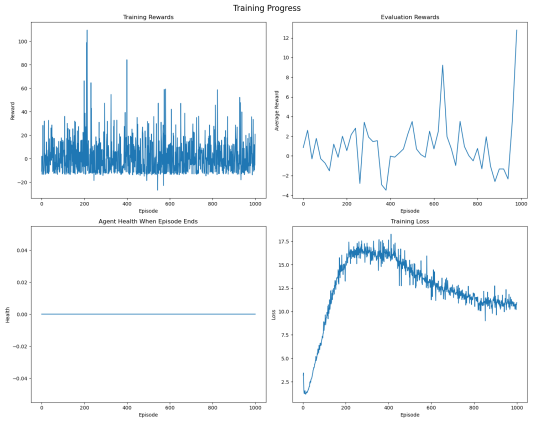

Figure 7: Training Progress without RNN

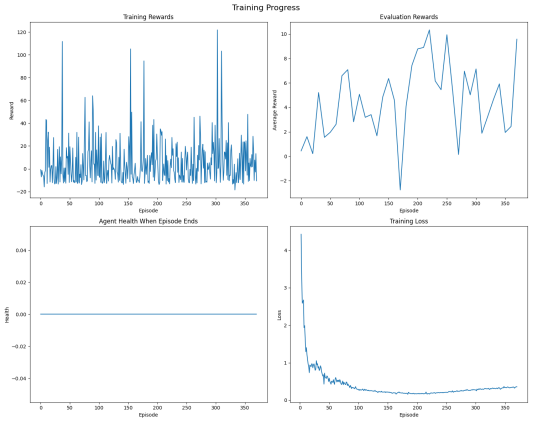

Figure 8: Training Progress with RNN

6.3 Action Distribution Comparison

Figure 9 10 compares the conditional action distributions of three agents: a trained agent, a baseline agent, and a random agent in the DQN without RNN configuration. The comparison reveals some insightful differences in decision-making behaviors among the agents.

In comparison with baseline and random agents, the trained agent exhibits a stronger preference for movement actions, accounting for over 94% of its total actions in without rnn setting, while the number of "Stay" actions is minimal. This behavior indicates that the trained agent actively explores the environment rather than remaining idle. Furthermore, the trained agent demonstrates strategic interactions with environmental elements. It equally treats food no matter if hunger is relatively high or low. In combat scenarios, the agent prefers fighting weaker threats while avoiding stronger ones, showing a risk-averse strategy to preserve health. Similarly, the agent enters caves more frequently when its health drops below a critical threshold (Health < 100), indicating adaptive behavior aimed at health recovery, compared with baseline and random one.

Comparing without RNN and with RNN configs, the latter shows a stronger tendency of fighting with weaker threats and entering caves when health low. It is interesting to note that agent using RNN enters caves significatly more times than the agent without RNN. This aligns with our hypothesis that agent using RNN can strengthen the "memory" of helpful information: because locations of caves do not change through one episode, the agent might memorize the locations of caves and would love to enter more times to live longer. The actions of eating food and threats, in comparison, showed less in RNN model, probably because locations of food and threat changes every timestep, and thus the weak temporal information makes RNN model has less tendency to perform actions of eating food and fighting with threats.

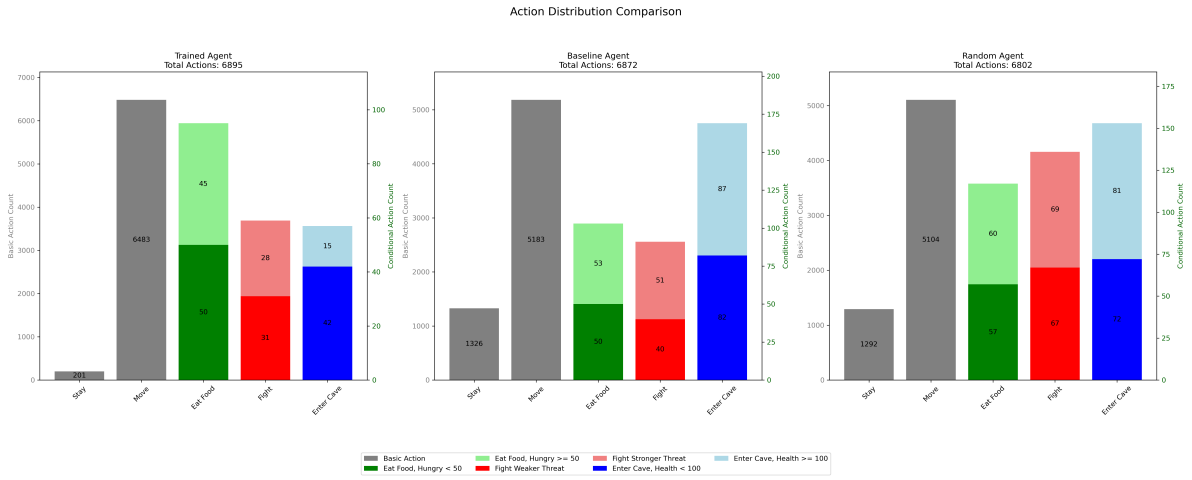

Figure 9: Action Distribution, without RNN

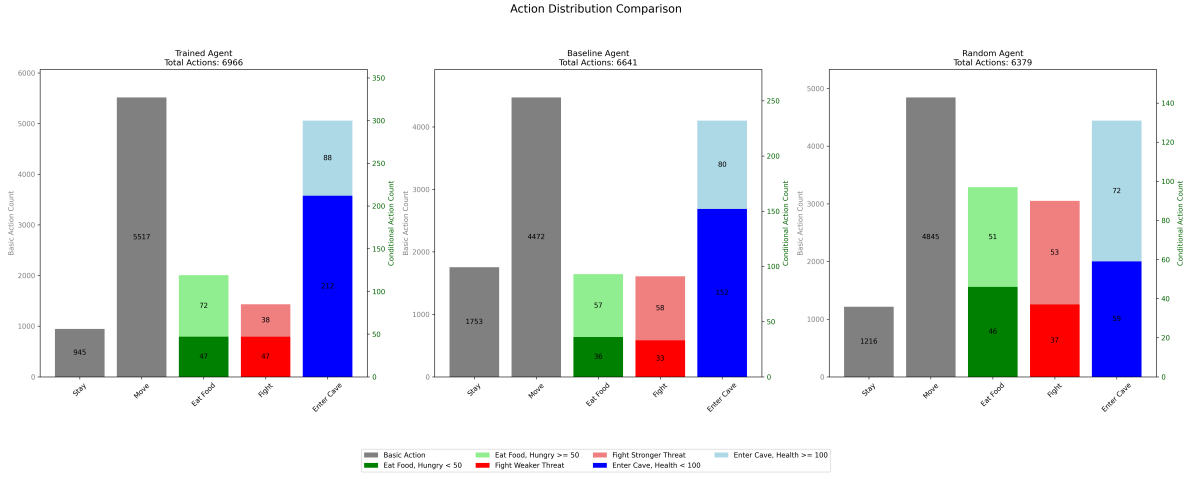

Figure 10: Action Distribution, with RNN

7. Challenges

During the implementation and training of our reinforcement learning agent, we encountered several challenges that impacted the overall design and performance of the system.

One significant challenge was the **high flexibility**, represented by large number of hyperparameters tunable involved in the training process. The model architecture, including convolutional layers, optional LSTM configurations, fully connected layers, and training configs, such as learning rate, discount factor, target network update frequency, gamma, epsilon, replay buffer size, are all configurable and requires careful fine-tuning for a desirable outcome. Each of these hyperparameters plays a critical role in the stability and efficiency of training, but the vast hyperparameter space made it difficult to identify optimal configurations within the limited training time available.

Another challenge was the **design of the reward function and game environment settings**. Reward shaping is crucial for guiding the agent toward learning effective strategies. However, balancing the rewards for food consumption, avoiding threats, and maintaining health proved to be non-trivial. Poorly designed rewards often led to unintended behaviors, such as the agent prioritizing short-term gains while ignoring long-term survival objectives. Additionally, the complexity of the environment, including dynamic food

and threat placements and their values, made it difficult to establish a reward structure that appropriately reflects the agent's performance.

Finally, due to time constraints, we were unable to implement a **multi-agent system**, which could have further enhanced the complexity and realism of the environment. Multi-agent interactions, such as cooperation or competition, are critical in many real-world scenarios and could provide additional challenges for the learning process. However, implementing such systems requires significant modifications to the environment and the training framework, which were beyond the scope of this project.

8. Conclusion

In this project, we implemented a reinforcement learning agent capable of survival and decision-making in a grid-based simulated ecosystem. By leveraging a Deep Q-Network (DQN) architecture with CNN and RNN, we demonstrated the agent's ability to interact with environmental elements, such as food, threats, and shelters, while managing internal states like health, hunger, and attack levels. The results highlight the importance of incorporating temporal features using RNNs, which improved performance and stability. Despite challenges related to hyperparameter optimization, reward function design, and the absence of a multi-agent system, the project illustrates the potential of reinforcement learning for modeling adaptive and survival-based behaviors in dynamic environments. Future work will explore multi-agent interactions, advanced reward shaping, and further optimization of the agent's learning process to enhance overall performance and realism.

9. Appendix

A. Supplementary Figures

Figure. 6 shows the reward distribution of DQN-RNN model compared to baseline and random agents. Figure. 12 shows the Q-value difference between DQN-RNN model and the baseline agent. Figure. 13 shows the location heatmap of the DQN-RNN model.

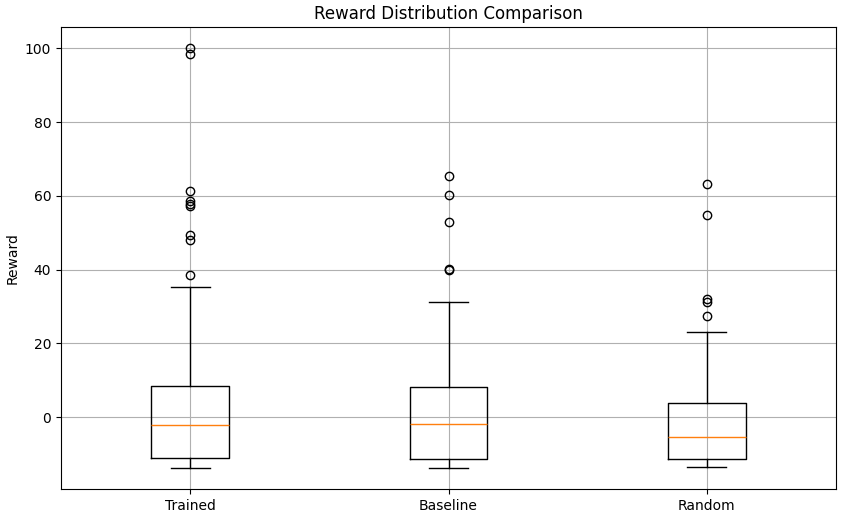

Figure 11: Reward Distribution of RNN Model Comparison

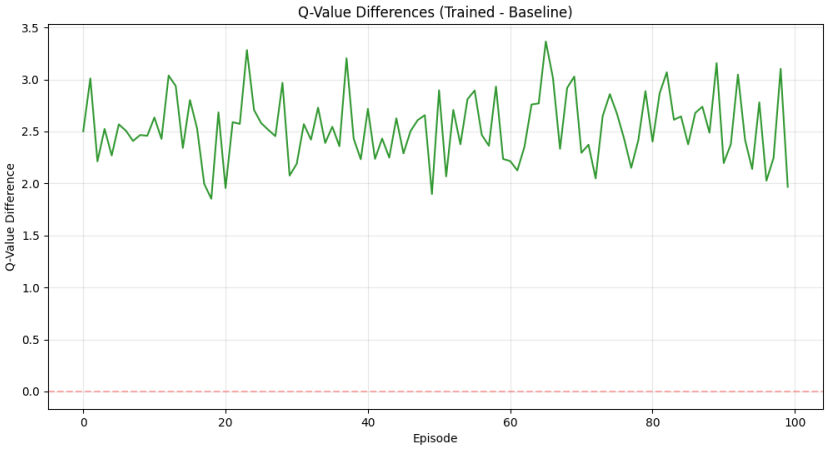

Figure 12: Q Value Difference between DQN Agent and Baseline Agent using RNN

B. Experiment Details

Table. 1 shows the configurations of the two model used in this report, and Table. 2 is the corresponding evaluation results. Note that the two configs has many differences, and we admit that due to time restriction we were unable to perform more rigorous ablation studies.

Table 1: Configurations of Models Used in This Report

Parameter	Without RNN	With RNN
Memory Size	500000	100000
Gamma	0.99	0.95
Learning Rate	0.0001	0.00001
Hidden Sizes	[256, 128]	[1024, 512, 256]
RNN Hidden Size	None	512
Sequence Length	None	4

Table 2: Evaluation Results

Evaluation	Without RNN	With RNN
Mean Reward (Agent)	1.0045	4.1165
Std Reward (Agent)	20.4479	22.1909
Mean Reward (Baseline)	0.4952	1.3875
Std Reward (Baseline)	14.3062	16.1039
Mean Q Diff (Baseline)	11.6392	2.5388
Std Q Diff (Baseline)	3.4646	0.3239
Mean Reward (Random)	2.7778	-0.6223
Std Reward (Random)	15.5099	14.0762

C. References and Codebases

During the development of our project, we referred to previous research, and open-source codebases. These references provided theoretical foundations, implementation guidance, and early-stage inspirations.

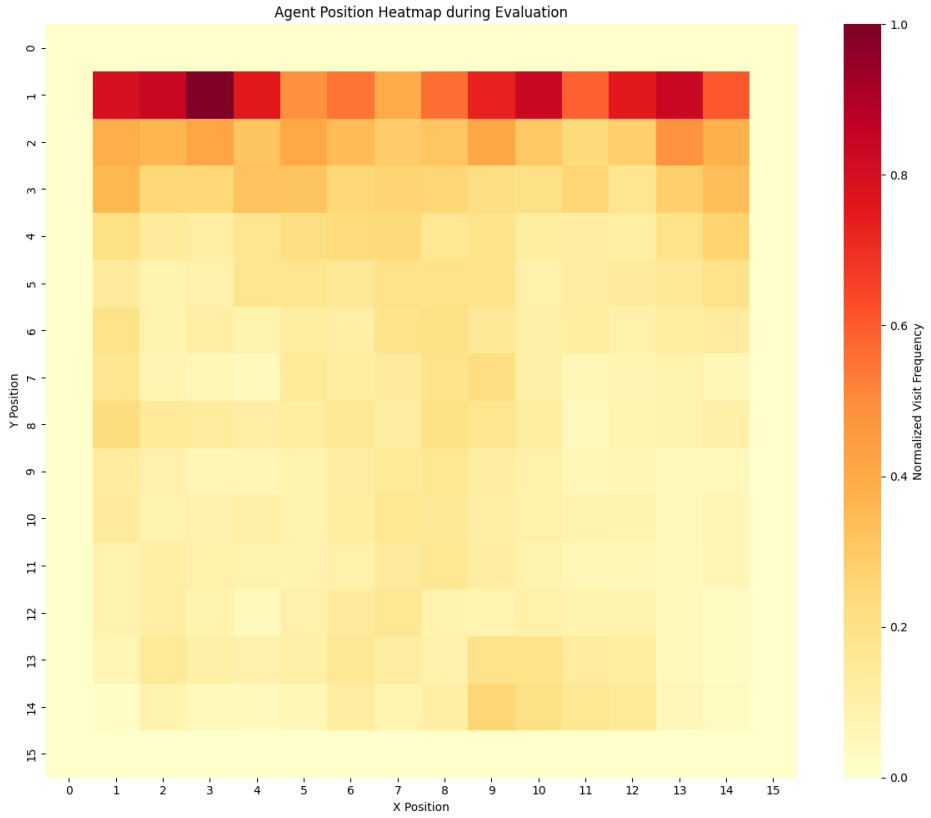

Figure 13: State Heatmap of RNN Agent

C.1 Primary References

References

[1] Peng Pai et al. "Achieving Safe Deep Reinforcement Learning via Environment Comprehension Mechanism". In: *Chinese Journal of Electronics* (2021).

[2] Hong Bai, Rong Cheng, and Yaochu Jin. "Evolutionary Reinforcement Learning: A Survey". In: *Intelligent Computing* 2 (2023). DOI: 10.34133/icomputing.0025.

[3] John Co-Reyes et al. "Ecological Reinforcement Learning". In: *arXiv preprint arXiv:2006.12478* (2020).

[4] Adrien Ecoffet et al. "Go-explore: a new approach for hard-exploration problems". In: *arXiv preprint arXiv:1901.10995* (2019).

[5] Nikki Lijing Kuang and Clement H. C. Leung. "Performance Dynamics and Termination Errors in Reinforcement Learning – A Unifying Perspective". In: *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. 2018, pp. 129–133. DOI: 10.1109/AIKE.2018.00028.

[6] Yifan Lin et al. "Evolutionary Reinforcement Learning: A Systematic Review and Future Directions". In: *arXiv preprint arXiv:2402.13296* (2024). URL: http://arxiv.org/abs/2402.13296.

[7] Hongbo Liu. "Cooperative multi-agent game based on reinforcement learning". In: *High-Confidence Computing* 4.1 (2024), p. 100205. ISSN: 2667-2952. DOI: 10.1016/j.hcc.2024.100205.

[8] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[9] T. Morita and H. Hosobe. "Video Game Agents with Human-like Behavior using the Deep Q-Network and Biological Constraints". In: *Proceedings of the 15th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART*. 2023, pp. 525–531. DOI: 10.5220/0011697500003393.

[10] Tim Salimans et al. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *arXiv preprint arXiv:1703.03864* (2017). URL: http://arxiv.org/abs/1703.03864.

[11] Alessandro Sebastianelli et al. "A Deep Q-Learning based approach applied to the Snake game". In: *2021 29th Mediterranean Conference on Control and Automation (MED)*. 2021. DOI: 10.1109/MED51440.2021.9480232.

[12] Y. Sun et al. "Research and Implementation of Intelligent Decision Based on a Priori Knowledge and DQN Algorithms in Wargame Environment". In: *Electronics* 9.10 (2020), p. 1668.

[13] Felipe Moreno Vera. "Performing Deep Recurrent Double Q-Learning for Atari Games". In: *CoRR* abs/1908.06040 (2019). arXiv: 1908.06040. URL: http://arxiv.org/abs/1908.06040.

C.2 Codebases

The following open-source codebases served as references for environment customization and early-stage reinforcement learning implementations:

1. OpenAI Gym: `https://gymnasium.farama.org/`
2. Use RL to Play Snake Game: `https://github.com/patrickloeber/snake-ai-pytorch`
3. RL in Maze Game: `https://github.com/erikdelange/Reinforcement-Learning-Maze`

D. Timeline

Nov. 11 Topic choose, revised proposal, system design
Nov. 20 Infrastructure design
Nov. 27 Infrastructure constructed
Dec. 7 Necessary features add
Dec. 10 Fine-tuning and bug fixed
Dec. 15 Code clean-up

E. Contributions

Anni Architecture design and employment, Experiment
Zeli Model design and employment, Experiment

F. Appendix: Project Links

The following is the GitHub repository. https://github.com/AnniLi1212/EE641_Learning_to_Survive.git

G. Project Dependencies

The following dependencies were used in this project, as specified in the `requirement.txt` file:

- numpy (`>=1.21.0`)

- torch (`>=2.0.0`)
- gymnasium (`>=0.29.0`)
- pygame (`>=2.5.0`)
- matplotlib (`>=3.7.0`)
- seaborn (`>=0.12.0`)
- opencv-python (`>=4.8.0`)
- pandas (`>=2.0.0`)
- pyyaml (`>=6.0.0`)
- tqdm (`>=4.65.0`)
- pillow (`>=9.5.0`)
- tensorboard (`>=2.12.0`)