CSCI 566 - Final Report: Animal Crackers

Jeffrey Guo jxguo@usc.edu
Bowen Leng bleng@usc.edu
Spencer Lin linspenc@usc.edu
Smit Shah smithite@usc.edu

Abstract

The project studies the evolution of group behaviors in colonies of agents using neural networks. By leveraging multi-agent reinforcement learning algorithms, we aim to uncover how complex group behaviors emerge through cooperative strategies in a dynamic environment and how feasible it is for downstream gaming applications.

1 Introduction

Aesthetic upgrades in graphical fidelity [1] and generative animation [2] [3] continue to add realism to agents or non-player characters (NPCs) in video games and other applications. Despite this, current agents rely on deterministic systems such as behavior trees or finite state machines to dictate behaviors [4]. This limitation often results in predictable, rule-based interactions that can diminish user engagement and fail to capture the complexity of more realistic social dynamics in multi-agent setups.

1.1 Proposed Solution: Evolving Prosocial Behaviors Through Deep Reinforcement Learning

A great factor contributing to the perceived intelligence of an agent is through demonstrated social behaviors such as prosociality (e.g., altruism, reciprocity, cooperation, and selfishness) [5]. Our goal is to simulate evolved prosocial behaviors through Deep Reinforcement Learning (DRL). We set up a series of simple test environments leveraging the Unity game engine [6] with tasks that require agents to work together. As agents sharpen their mastery of completing these objectives (e.g. collecting food), they progress to environments with more complex tasks such as cooperatively breaking down walls to gain access to more food or introducing new constraints such as hunger and dying of old age. Ultimately, we will assess if a DRL approach can train agents that intelligently adapt their social behavior in dynamic environments and how feasible this approach is for downstream gaming applications.

1.2 Challenges in Implementation

Several challenges are posed to this DRL approach. First, good metrics and policies need to be designed so as to best define and measure agent "success" (e.g., individual vs. group utility). Then, fair testing environments need to be designed so they are effective for agents to learn cooperative behaviors, yet simple enough to not confuse the agents. Finally, the test environment needs to be watertight so agents do not overfit to certain tactics or discover ways to cheat (e.g. phasing through walls or abusing certain mechanics).

Preprint. Under review.

1.3 Potential Benefits and Applications

The potential applications of developing DRL-driven agents are wide and include enhancing gaming experiences with more dynamic and intelligent agent interactions. Furthermore, DRL-driven agents can provide potential insights to evolutionary biology and ecology, perhaps informing conservation efforts and ecosystem management strategies. Finally, this research can provide general insights into DRL.

2 Related Works

2.1 Evolutionary Game Theory

To begin, early works on evolutionary prosocial behaviour was based on evolutionary game theory [5]. In short, pure mathematical probabilities and statistics without any machine learning. Nowak shows that following this framework, altruism eventually wins out as the most stable strategy for agents to adopt. Though mathematically sound, this approach takes a more macroscopic approach in which all agents are assumed static in disposition until death. In reality, humans and animals are dynamic creatures which adopt different social strategies at different moments. Originally we hoped to take inspiration from this neuroevolution technique to bring that dynamism to evolutionary simulations. However, our current approach is more focused on emerging group behaviors based on collective performance.

2.2 Dynamic Alliance

A method used in a 2011 study of hunting robot cooperation was the "Dynamic Alliance" approach [7]. By selecting one robot to be a temporary commander with nearby robots formed into an alliance, the commander could efficiently request information from other robots and dynamically change the alliance based on closer evader distance and robot failure. The cooperation relates to collective intelligence and the emergence of coordination between our agents in order to complete a common goal. The dynamic assignment of teams and receiving information by a single leader can be incorporated similarly to gather food, complete tasks, and combat.

This approach is adaptable and flexible to changing environments and number of agents. It's scalable because it limits computational overhead by minimizing communication between agents. The results were convincing, showing through multiple simulations of varying environments, hunters, and evaders, that this approach generally performed more efficiently than previous approaches. This concept is something we initially looked into because of its effectiveness at dynamic coordination. Currently, hunting of this style is not included in our environment, but it is a potential approach we can take in the future for observing cooperative behaviors.

2.3 NeuroEvolution of Augmenting Topologies

We previously identified the NEAT (NeuroEvolution of Augmenting Topologies) algorithm as the approach to neuroevolution [8] we would focus on. NEAT allows for dynamic changes in neural network topology through and inheritance and mutation, promoting diversity in agent behaviors and problem-solving approaches. The algorithm employs historical markings, speciation for innovation protection, and incremental growth from minimal structures, which aligned with our aim to evolve efficient and diverse collective behaviors. NEAT efficiently leverages structural modifications to minimize search space and creates solutions that become more complex as they optimize, mirroring natural evolutionary processes. NEAT's simultaneous optimization and complexification of solutions strengthens the analogy between genetic algorithms and natural evolution. This was a key aspect of our research before, but takes a more individual approach compared to our current group learning idea.

2.4 Multi-Agent Posthumous Credit Assignment

We will focus on the MA-POCA (Multi-Agent Posthumous Credit Assignment) algorithm as a key existing approach to cooperative learning [9]. MA-POCA handles multi-agent reinforcement learning by sharing the reward function among agents within an episode. It improves upon previous MARL

(Multi-Agent Reinforcement Learning) algorithms by using a centralized critic that assesses the value of each agent's actions, allowing each agent to optimize its actions while considering the broader environment, but with the agents still acting independently based on its local observations. The agents will have improved coordination abilities through indirectly considering other agents' actions, which is especially useful in working towards shared goals. This particular learning capability is what we hope to capitalize on in our research.

3 Methods

3.1 MA-POCA

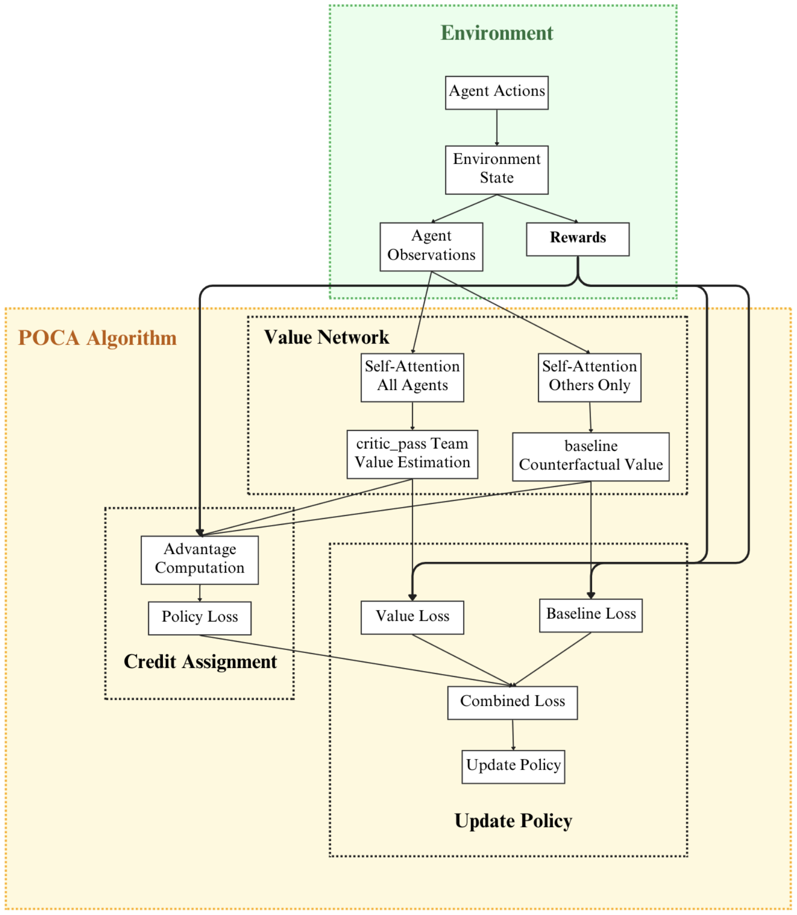


Figure 1: POCA Algorithm Architecture: showing the flow of observations, actions, and rewards through the network components. Pseudocode in Appendix: Algorithm 1.

Our project focuses on employing the MA-POCA (Multi-Agent Proximal Policy Optimization with Centralized Critics and Decentralized Actors) algorithm [9], a state-of-the-art multi-agent reinforcement learning (MARL) framework specifically designed for scenarios where agents can be dynamically created or terminated within an episode. This approach overcomes the traditional challenges faced by MARL systems using absorbing states. We leverage the Unity ML-Agents framework [10] to provide an interface to a physics-based training environment.

The centralized critic network in MA-POCA employs a self-attention mechanism to dynamically process the observations of active agents. This method addresses the inefficiencies associated with absorbing states, which were used in earlier MARL methods to handle agent termination. By using

Residual Self-Attention (RSA) blocks, the critic efficiently models inter-agent dependencies and scales to different numbers of agents without needing to predefine a maximum agent count.

MA-POCA implements a counterfactual baseline technique to solve the Posthumous Credit Assignment problem. This baseline marginalizes out an agent's action to estimate its specific contribution to the team's reward, even after termination. The use of attention-based mechanisms (depicted in Figure 2) ensures that each agent's advantage is then accurately calculated, enhancing learning efficiency in environments with variable agent participation. Finally, each agent operates autonomously using local observations, in line with the centralized training and decentralized execution framework. This enables the agents to coordinate effectively during runtime without requiring centralized control. An architecture diagram of the MA-POCA algorithm can be referred to in Figure 1.

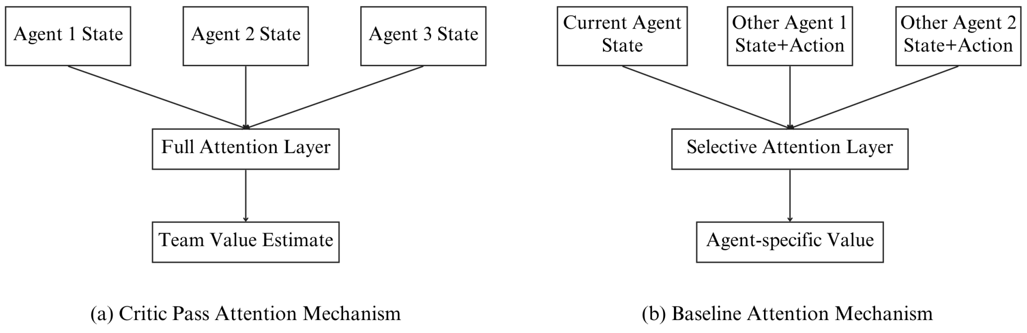

Agent 1 State
Agent 2 State
Agent 3 State
Full Attention Layer
Team Value Estimate
(a) Critic Pass Attention Mechanism
Current Agent State
Other Agent 1 State+Action
Other Agent 2 State+Action
Selective Attention Layer
Agent-specific Value
(b) Baseline Attention Mechanism

Figure 2: POCA Attention Mechanism. Pseudocode in Appendix: Algorithm 2.

4 Experiments

Videos of our experiments can be viewed here.

4.1 Simple Experiment: Wall Break

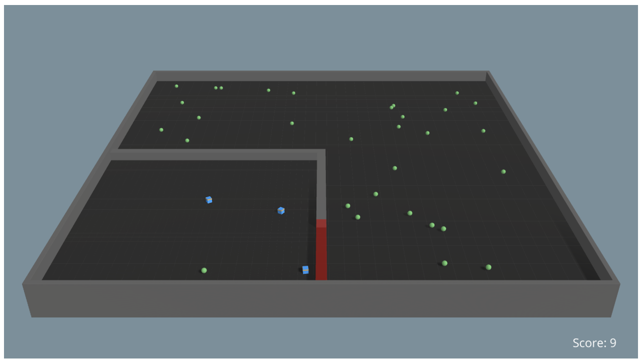


Figure 3: Unity training environment with 3 agents and a currently unbroken wall (colored red)

We began experimentation with 3 agents in a simple cooperative food collection scenario. Each agent is given a camera sensor to observe the surroundings, visualized in Figure 7. Agents are tasked with collecting as many food pellets in the play space as possible in an allotted amount of steps. A dilemma is presented in this environment design: agents spawn in the bottom-left quadrant of the play space which is completely blocked off, visualized in Figure 3. There are 10 non-respawning food pellets which can be collected in this first quadrant as well as a breakable wall that requires 2 or more agents to come in contact with to break. Outside, there are are a total of 30 food pellets which respawn in a random location outside the first quadrant once collected. The policies used can be referred to in section 6.

Initially, it was feared that agents may have trouble learning how to break the wall without prior experience. As such, we adopted two approaches. Scenario 1 consisted of training the agent for 1M steps in which only one agent is required to touch the wall to break it. For the next 1M steps the requirement is bumped back up to 2 agents. We compared this approach to Scenario 2 which held the 2 agent requirement to break the wall throughout all 2M steps. We discuss our findings in the Discussion section.

4.2 Complex Experiment: Reproducing Agents

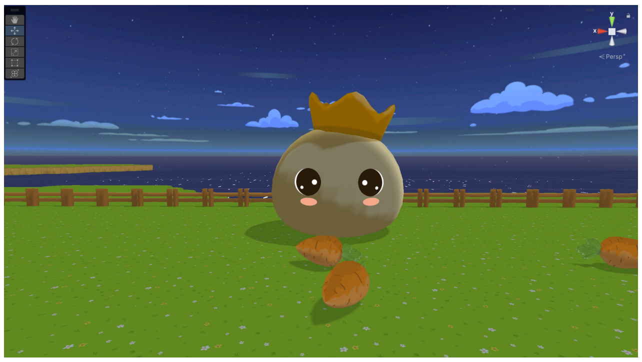


Figure 4: More complex environment for Reproducing Agents experiment with "queen" entity (center) which reproduces new agents when fed food by the agents.

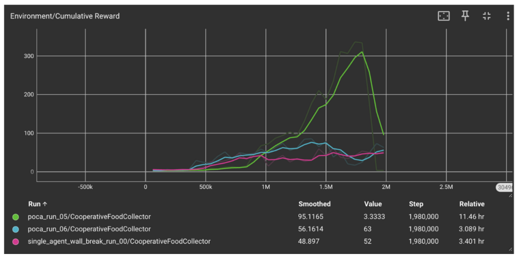
To test if MA-POCA is suitable for modeling evolved intelligence through agent reproduction, we upgraded the testing environment. We introduced new mechanics and rebalanced the reward structure 6, built a significantly more visually complex environment illustrated in Figure 4. Agents are still tasked with maximizing food collection efficiency but can now perish from hunger/old age as well as pass down their "genes" by asexually reproducing via feeding a "queen" entity. Detailed environment parameters can be found in Section 6.1.

Testing in a visually complex environment provides a more representative environment to a video game, a possible downstream application of our DRL agents. However, the added complexity necessitated the use of the more advanced nature CNN [11] vision encoder. This is especially true due to the food object, a carrot, having a similar color to the barrier fence object. We train the agent for 2M steps using two visual encoder sizes and compare the results.

5 Discussion

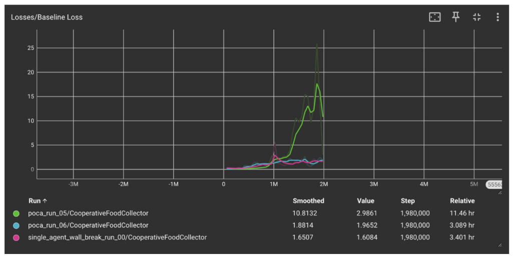
5.1 Break Wall Results

Our results demonstrate that agents are able to learn and adapt collaborative strategies increasingly efficiently over time. The training logs reveal a significant reduction overtime in the time taken for agents to collectively break the wall — initially requiring approximately one minute to synchronize their actions, which has progressively decreased to under three seconds by the end of training. Initial increasing loss values indicate healthy exploration phases, with agents actively discovering new strategies and experiencing higher uncertainty. The reward graphs demonstrate initial rapid learning as agents discovered basic cooperative strategies, such as breaking out as soon as possible, leading to a sharp performance increase around 1M steps. This improvement peaked at approximately 1.8M steps, with POCA_run_05 achieving the highest rewards ($\sim$300). However, the subsequent decline in performance, coupled with decreasing loss metrics, suggests our agents began overfitting. Finally, contrary to our original belief, we found that the MA-POCA algorithm performs even better when used in scenario 2 which may be due to the agents having to "unlearn" certain behaviors in the middle of training for scenario 1.



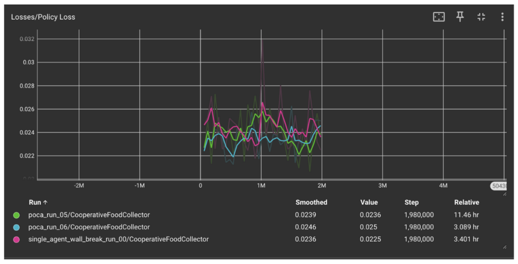
Run ↑	Smoothed	Value	Step	Relative
● poca_run_05/CooperativeFoodCollector	95.1165	3.3333	1,980,000	11.46 hr
● poca_run_06/CooperativeFoodCollector	56.1614	63	1,980,000	3.089 hr
● single_agent_wall_break_run_00/CooperativeFoodCollector	48.897	52	1,980,000	3.401 hr

(a) Cumulative Reward Graph measuring critic performance.
Pink: Scenario 1. Green/Blue: Scenario 2.



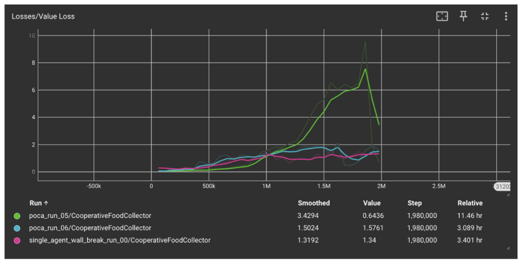
Run ↑	Smoothed	Value	Step	Relative
● poca_run_05/CooperativeFoodCollector	10.8132	2.9861	1,980,000	11.46 hr
● poca_run_06/CooperativeFoodCollector	1.8814	1.9652	1,980,000	3.089 hr
● single_agent_wall_break_run_00/CooperativeFoodCollector	1.6507	1.6084	1,980,000	3.401 hr

(b) Baseline Loss Graph measuring critic performance.
Pink: Scenario 1. Green/Blue: Scenario 2.



Run ↑	Smoothed	Value	Step	Relative
● poca_run_05/CooperativeFoodCollector	0.0239	0.0236	1,980,000	11.46 hr
● poca_run_06/CooperativeFoodCollector	0.0246	0.025	1,980,000	3.089 hr
● single_agent_wall_break_run_00/CooperativeFoodCollector	0.0236	0.0225	1,980,000	3.401 hr

(c) Policy Loss Graph measuring actor performance.
Pink: Scenario 1. Green/Blue: Scenario 2.

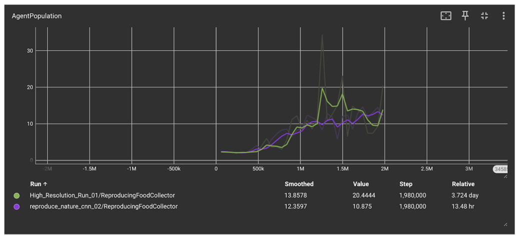


Run ↑	Smoothed	Value	Step	Relative
● poca_run_05/CooperativeFoodCollector	3.4294	0.6436	1,980,000	11.46 hr
● poca_run_06/CooperativeFoodCollector	1.5024	1.5761	1,980,000	3.089 hr
● single_agent_wall_break_run_00/CooperativeFoodCollector	1.3192	1.34	1,980,000	3.401 hr

(d) Value Loss Graph measuring critic performance.
Pink: Scenario 1. Green/Blue: Scenario 2.

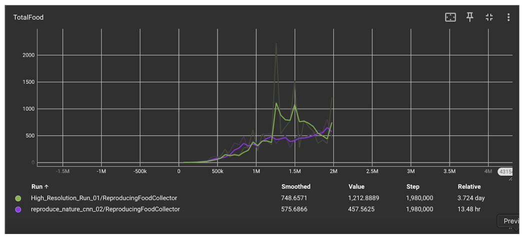
Figure 5: Break Wall training results demonstrating agents successfully learning overtime to work together

5.2 Reproducing Agents Results



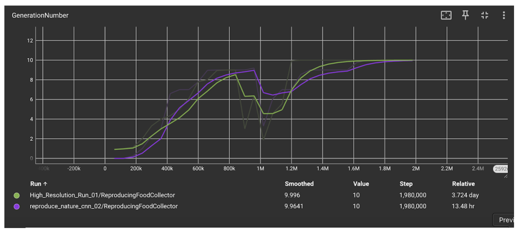
Run ↑	Smoothed	Value	Step	Relative
● High_Resolution_Run_01/ReproducingFoodCollector	13.8578	20.4444	1,980,000	3.724 day
● reproduce_nature_cnn_02/ReproducingFoodCollector	12.3597	10.875	1,980,000	13.48 hr

(a) Agent Population Graph
Green: 128x128 Visual Encoder. 96x96 Visual Encoder.



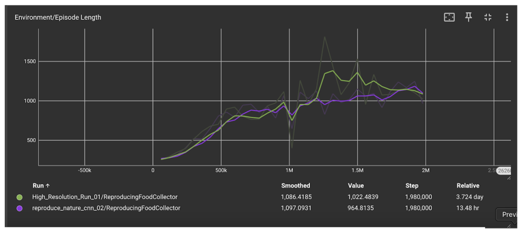
Run ↑	Smoothed	Value	Step	Relative
● High_Resolution_Run_01/ReproducingFoodCollector	748.6571	1,212.8889	1,980,000	3.724 day
● reproduce_nature_cnn_02/ReproducingFoodCollector	575.6866	457.5625	1,980,000	13.48 hr

(b) Total Food Graph
Green: 128x128 Visual Encoder. 96x96 Visual Encoder.



Run ↑	Smoothed	Value	Step	Relative
● High_Resolution_Run_01/ReproducingFoodCollector	9.996	10	1,980,000	3.724 day
● reproduce_nature_cnn_02/ReproducingFoodCollector	9.9641	10	1,980,000	13.48 hr

(c) Generation Number Graph
Green: 128x128 Visual Encoder. 96x96 Visual Encoder.



Run ↑	Smoothed	Value	Step	Relative
● High_Resolution_Run_01/ReproducingFoodCollector	1,086.4185	1,022.4839	1,980,000	3.724 day
● reproduce_nature_cnn_02/ReproducingFoodCollector	1,097.0931	964.8135	1,980,000	13.48 hr

(d) Environment/Episode Length Graph
Green: 128x128 Visual Encoder. 96x96 Visual Encoder.

Figure 6: Reproducing Agents training results comparing vision encoder performance

The Reproducing Agents experiment confirmed the applicability of the MA-POCA algorithm in evolutionary scenarios. However, we found the biggest constraint of implementing such algorithms for gaming applications is the unscalable nature of each agent having a vision encoder. Even in our relatively simple game environment, the vision encoder performs suboptimally unless a larger resolution is fed to the CNN. As such, we compared the performance between a low resolution training run and a higher resolution run. As shown in Figure 6, the agents in both runs struggled at collecting food until 500K steps, but the continual increase in episode length (survival lifespan of the agent colony) signified that the agents were slowly optimizing their behaviors. From 500K onwards, the max agent population achieved would increase, leveling out at 10 generations born. As we had predicted, the agents with the larger 128x128 pixel visual encoder size was able to perform substantially better than the 96x96 pixel agents. They moved in more cohesive ways and crashed into walls less, presumably because they were better at sensing their environment and distinguishing between the fences and carrots. Though, we were surprised that the agent colony never reached a more stable stagnation of performance as one might expect a population of animals to do in a finite space, effectively unlimited food, and a limited time quantum. This may be due to stochastic nature of the MA-POCA algorithm to prioritize exploration/exploitation or due to the stochastic nature of the locations where the food respawn.

6 Conclusion

Given the promise of our initial experiments, more experiments can be conducted to compare the performance of our current asexually reproducing agent colony to colonies capable of reproducing agents with mutations or sexual reproduction. The reward structure can be further enhanced to incentivize more sophisticated strategies. More efficient vision encoders such as a SAM can be used instead of a CNN. Additionally, to streamline training, we can continue to explore parallel training by duplicating multiple environments, where all agents share their learning across these areas during training with MA-POCA. An example of such a training setup is shown in Figure 8.

Ultimately, our results are promising and align with our expectations that the MA-POCA algorithm is effective in fostering cooperation among agents in environments that require collective action. The agents display emergent collaborative behavior without requiring explicit rule-based coordination mechanisms. Though this approach is held back by its scalability, it is promising and warrants future exploration.

References

[1] Marc Habermann, Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Gerard Pons-Moll, and Christian Theobalt. Real-time deep dynamic characters. *ACM Trans. Graph.*, 40(4), July 2021.

[2] Sebastian Starke, Ian Mason, and Taku Komura. Deepphase: periodic autoencoders for learning motion phase manifolds. *ACM Trans. Graph.*, 41(4), July 2022.

[3] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local motion phases for learning multi-contact character movements. *ACM Trans. Graph.*, 39(4), August 2020.

[4] Cagkan Uludagli and Kaya Oguz. Non-player character decision-making in computer games. *Artificial Intelligence Review*, 56:1–33, 04 2023.

[5] Birgit Lugrin, Catherine Pelachaud, and David Traum, editors. *The Handbook on Socially Interactive Agents: 20 years of Research on Embodied Conversational Agents, Intelligent Virtual Agents, and Social Robotics Volume 1: Methods, Behavior, Cognition*, volume 37. Association for Computing Machinery, New York, NY, USA, 1 edition, 2021.

[6] Unity Technologies. Unity, 2023. Game development platform.

[7] Jianjun Ni and Simon X. Yang. Bioinspired neural network for real-time cooperative hunting by multirobots in unknown environments. *IEEE Transactions on Neural Networks*, 22(12):2062–2077, 2011.

[8] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[9] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. *RL in Games Workshop AAAI 2022*, 2022.

[10] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Appendix

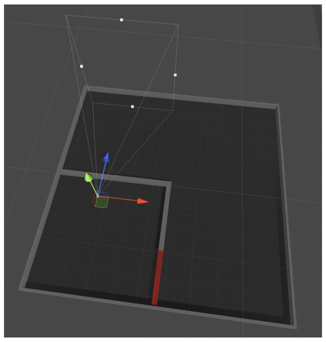


Figure 7: Agent camera sensor field of view (Break Wall experiment)

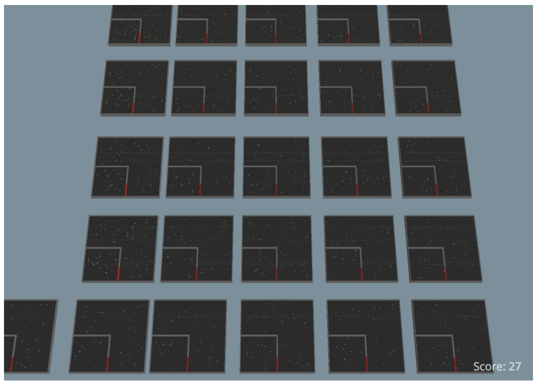


Figure 8: Parallel training setup for Break Wall experiment to boost efficiency of training further

Wall Break Agent Rewards

Individual Agent Rewards

1. **Eating a food pellet:** $+1$ pt
2. **Total Individual Reward:**

$$I_r(t) = n_{\text{food}}(t) \cdot 1$$

Group Rewards

- **Eating a food pellet:** $+1$ pt
- **Breaking the wall:** $+10$ pt

- **Hurry Up Penalty:**
$$G_p(t) = -\frac{0.5}{\text{MaxEnvironmentSteps}} \cdot t$$
- **Total Group Reward:**
$$G_r(t) = n_{\text{food}}(t) \cdot 1 + n_{\text{wall}}(t) \cdot 10 - \frac{0.5}{\text{MaxEnvironmentSteps}} \cdot t$$

Reproducing Agents Rewards

Individual Agent Rewards

1. **Collecting a Carrot:**
$$R_{\text{food}} = 10$$
2. **Successful Procreation:**
$$R_{\text{procreation}} = 5$$
3. **Starvation Penalty:**
$$R_{\text{starvation}} = -50$$
4. **Hunger Penalty Over Time:**
$$R_{\text{hunger}} = -0.001 \cdot T_{\text{last food}}$$
where $T_{\text{last food}}$ is the time elapsed since the agent last ate.
5. **Total Individual Reward:**
$$R_{\text{total}} = R_{\text{carrot}} + R_{\text{procreation}} + R_{\text{starvation}} + R_{\text{hunger}}$$

Group Rewards

1. **Collecting a Carrot (Group Reward):**
$$G_{\text{carrot}} = 10$$
2. **Successful Procreation (Group Reward):**
$$G_{\text{procreation}} = 20$$
3. **All Agents Dying (Group Penalty):**
$$G_{\text{death}} = -20$$
4. **Total Group Reward:**
$$G_{\text{total}} = G_{\text{carrot}} + G_{\text{procreation}} + G_{\text{death}}$$

6.1 Reproducing Agents Environment Parameters

- **Environment Parameters:**
  - 3 second cooldown before carrots respawn
  - 5 second cooldown for procreating
  - One agent costs 10 carrots to create
  - Agents have a total lifespan of 2 minutes
  - Agents die of starvation if they don't eat in 20 seconds

Algorithm 1: POCA Main Training Algorithm Pseudocode

```
while not done do
    /* 1.  Collect experiences                                        */
    trajectories = collect_trajectories()
    /* 2.  Process each trajectory                                    */
    for trajectory in trajectories do
        /* 2.1 Value Estimation using Self-Attention                  */
        value_estimates = critic_pass(
            observations = all_agent_obs,
            self_attention_fn = lambda obs: // Process all agents together
                attention_weights = softmax(Q × K^T / sqrt(d_k))
                attended_values = attention_weights × V
                return attended_values
        )
        /* 2.2 Baseline Estimation using Self-Attention               */
        baseline_estimates = baseline(
            current_agent_obs=agent_obs,
            other_agents_obs_actions=(other_obs, other_actions),
            self_attention_fn=lambda obs_acts: // Process others, excluding current agent action
                attention_weights = softmax(Q × K^T/ sqrt(d_k))
                attended_values = attention_weights × V
                return attended_values
        )
        /* 2.3 Credit Assignment                                      */
        advantages = compute_advantages(
            value_estimates = value_estimates,
            baseline_estimates = baseline_estimates,
            rewards = rewards,
            lambda_returns = compute_lambda_returns( // Using TD() returns
                rewards,
                value_estimates,
                gamma,
                lambda
        )
        /* 2.4 Store processed trajectory                             */
        buffer.add(
            observations = observations,
            actions = actions,
            advantages = advantages,
            value_estimates = value_estimates,
            baseline_estimates = baseline_estimates
        )
    end
    /* 3.  Policy Update                                              */
    if buffer.is_ready() then
        /* 3.1 Get batch                                              */
        batch = buffer.get_batch()
        /* 3.2 Compute Losses                                         */
        policy_loss = compute_policy_loss(
            advantages = batch.advantages,
            current_policy = current_policy,
            old_policy = old_policy
        )
        value_loss = compute_value_loss(
            predicted_values = value_estimates,
            actual_returns = lambda_returns
        )
        baseline_loss = compute_baseline_loss(
            predicted_baseline = baseline_estimates,
            actual_returns = lambda_returns
        )
        /* 3.3 Combined Loss                                          */
        total_loss = policy_loss + 0.5 * (value_loss + 0.5 * baseline_loss) - entropy_coefficient * entropy
        /* 3.4 Update Networks                                        */
        optimizer.zero_grad()
        total_loss.backward()
        optimizer.step()
    end
end
```

Algorithm 2: Self-Attention Implementation Details Pseudocode

```
Result: Self-Attention Implementation
/* Self-Attention Implementation Details                                  */
Function SelfAttention(queries, keys, values):
    /* queries: Agent states we're computing attention for                */
    /* keys: States we're attending to                                    */
    /* values: Information to aggregate                                   */
    /* Compute attention scores                                           */
    attention_scores = matmul(queries, transpose(keys))
    attention_scores = attention_scores / sqrt(keys.shape[-1])
    /* Normalize with softmax                                             */
    attention_weights = softmax(attention_scores)
    /* Compute weighted sum of values                                     */
    attended_values = matmul(attention_weights, values)
    return attended_values
Function CriticPass(all_agent_obs):
    /* Team value estimation                                              */
    /* Process all agents equally                                         */
    queries = linear_transform(all_agent_obs)
    keys = linear_transform(all_agent_obs)
    values = linear_transform(all_agent_obs)
    attended = self_attention(queries, keys, values)
    return value_network(attended)
Function Baseline(current_agent_obs, other_obs_actions):
    /* Counterfactual value estimation                                    */
    /* Process current agent differently from others                      */
    current_query = linear_transform(current_agent_obs)
    other_keys = linear_transform(other_obs_actions)
    other_values = linear_transform(other_obs_actions)
    attended = self_attention(current_query, other_keys, other_values)
    return baseline_network(attended)
```