# Predator and Prey Simulations Using Proximal Policy Optimization (PPO)

Mahi Pasarkar
*Rensselaer Polytechnic Institute*
Troy, NY
pasarm@rpi.edu

***Abstract*—Creating complex behaviors and interactions between agents is a challenge within the field of computing and the field of video games. A typical interaction scenario can be modeled by creating two competing agents, one ”Predator” and one ”Prey.” By iteratively adding features, I was able to control the way the behaviors developed and track how they grew in complexity. I found that an effective way to create interesting behaviors is to have an agent balance between several simple goals.**

***Index Terms*—Proximal Policy Optimization, Simulation, Unity ML-Agents, Predator/Prey**

## I. Introduction

Agents using Proximal Policy Optimization (PPO) are highly effective at maximizing reward in a given environment. I wanted to see how agents would behave when placed in a competitive environment where both agents have conflicting goals. Using Unity and the ML-Agents package, two Agents were created: one ”Predator” that would get a reward for colliding with the ”Prey,” and a ”Prey” that would get a punishment for colliding with the ”Predator.” The agents were put in a 2D environment where they could move in all directions, including diagonally. The environment also had a few obstacles in order to allow for more complex behavior and to necessitate the development of obstacle avoidance. In later runs, a static ”Goal” was added for the prey to reach, simulating plants that prey may graze on.

The designs of the agents and environment were improved iteratively, adding complexity with each iteration. By using this method, the simulation can grow in complexity in a controlled way, similar to curriculum training.

## II. Related Work

Creating agents with conflicting goals and watching them strategize against each other is not a new idea. The original inspiration behind this paper was *Emergent Tool Use From Multi-Agent Autocurricula* [1], a paper that created agents to play ”Hide and Seek.” There were two teams, one with hiders, and the other with seekers. Rewards were earned for the team rather than the individual, meaning that if a hider was caught, the rest of the hiders would get a negative reward. This incentivized team strategies, such that hiders would ensure all hiders were safe, and seekers would work together for only one seeker to capture the hiders.

The interesting component of this paper lies in the emergent behavior that the two teams exhibited. The agents were in a 3D space and were able to grab, move, and lock objects such as boxes, walls, and ramps. There are also static walls that cannot be moved. In order to give the hiders a chance, seekers are unable to move for a brief period at the start of the round. At first, the hiders were consistently caught by the seekers. Then, they discovered they could move boxes to cover up their shelter and avoid being caught. The seekers now couldn’t access the hiders, so they learned to move a ramp to the wall and jump over. To combat this, the hiders would take the ramp into their shelter before the seekers could get to it.

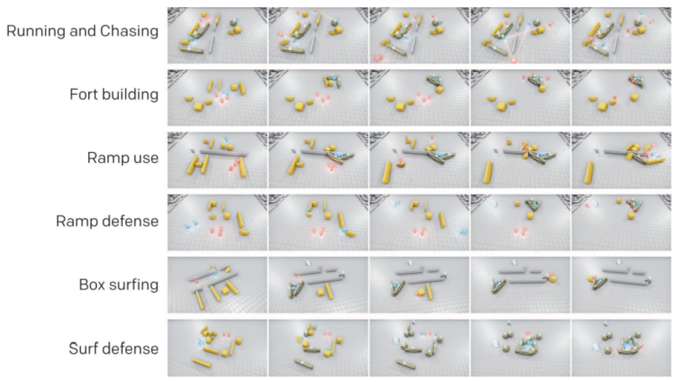

Fig. 1. A Compilation of Emergent Behaviors from Hide and Seek

After this point, the agents were moved to a random environment with far more movable objects. The hiders learned to lock every ramp they could before hiding in a shelter they constructed out of movable walls. The researchers assumed that this would be the end of the strategies, but seekers surprised them by coming up with a strategy called ”box surfing.” In this strategy, a seeker would grab a box and take it near a locked ramp. It would run up the ramp and continue to grab the box until it was on top of it. It could now move the box while also being on top of it, allowing it to scale the hider’s fort. The final strategy for the hiders was to lock as many boxes as they could before making their fort in order to prevent box surfing.

Other researchers have tweaked reward algorithms to alter agent behavior in the video game Pong [2]. In a competitive reward scheme, each agent gets a reward when the opposing agent misses the ball. In a collaborative reward scheme, both agents get a punishment when one misses a ball.

Both reward schemes favored slower balls, as faster balls were unpredictable and more frequently led to a loss. Wall bounces also became infrequent among both reward schemes due to similar reasons. What is interesting about this finding is that despite having different reward schemes, similar behavior emerged. Within any given environment, competing agents may cooperate in certain cases.

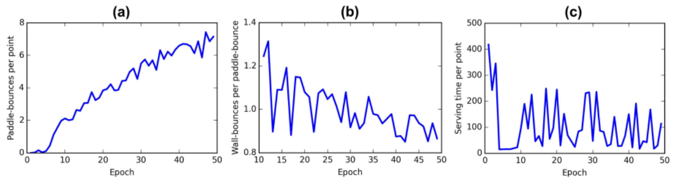

Fig. 2. Evolution of the Behavior of the Competitive Pong Agents During Training.

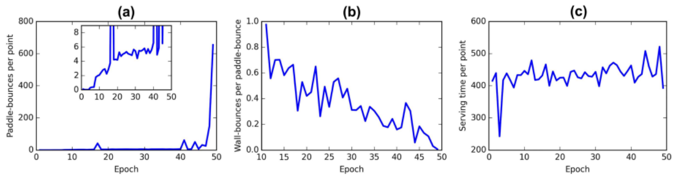

Fig. 3. Evolution of the Behavior of the Collaborative Pong Agents During Training.

Competitive agents have also been used to improve real-time strategy (RTS) game AI [3] and to train AI to play Super Smash Bros. Melee [4]

## III. Agents Setup

Fig. 4. Image of "Prey" Agent

Fig. 5. Image of "Predator" Agent

### A. Actions

Both agents have two continuous actions from $[-1, 1]$. One continuous action indicates the x movement that step, and the other indicates the y movement. For example, if action one is $(-0.75)$ and action two is $(0.9)$, the agent will move to the left by $0.75*$ moveSpeed and up by $0.9*$ moveSpeed. The resultant movement vector is normalized by Time.deltaTime, which ensures that it will move consistently despite variable frame rates. moveSpeed is a custom variable that scales the speed of the agent and can be different for each agent.

The pseudocode for actions is as follows:

```
position += (action[0], action[1], 0) *
    moveSpeed * Time.deltaTime
```

### B. Observations

Agents have two types of observations:

1) Position observation
2) Raycast observation

Position observations consist of 3 floats (defined in C# as Vector3). These floats contain the $x$, $y$, and $z$ positions of the observation. While $z$ is unnecessary in a 2D environment, the position value that Unity transforms store are in Vector3 format, so $z$ was kept.

Raycast observations consist of an array of floats (defined in C# as List<float>). These floats contain the distances each raycast traveled before colliding with an object. The order of the raycast distances in the array is always the same, meaning if a given raycast maps its value to index 3, it will always map its value to index 3.

### C. Rewards

The predator receives a reward for colliding with the prey. The reward is calculated as

$$R(t) = 100 + (100/t) \tag{1}$$

where $t$ is the seconds elapsed since the episode began. The below figure demonstrates the reward over time. The longer it takes to reach the prey, the less reward the predator will receive. This incentivizes catching the prey as soon as possible.

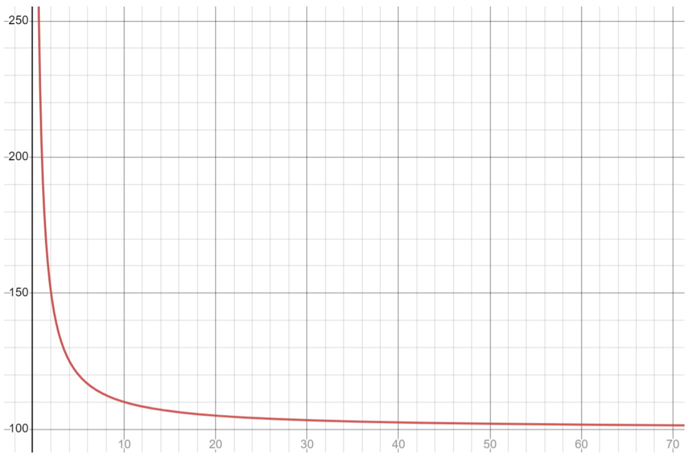

Fig. 6. Graph of Predator's Reward Function (x is time, y is reward)

Similarly, the prey receives a punishment (negative reward) for colliding with the predator, calculated as

$$R(t) = -100 - (100/t) \tag{2}$$

where $t$ is the seconds elapsed since the episode began. The below figure demonstrates the reward over time. The longer the

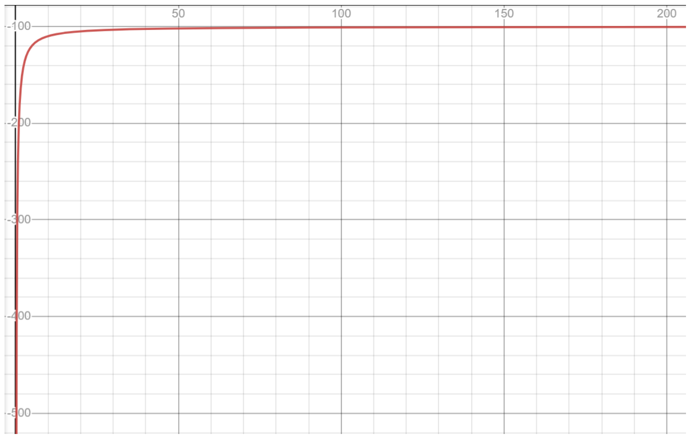

Fig. 7. Graph of Prey's Reward Function (x is time, y is reward)

prey lasts, the less its punishment would be. This incentivizes surviving for as long as possible.

The prey also receives a reward for reaching a goal, calculated as

$$R(t) = 50 + (100/t) \tag{3}$$

where $t$ is the seconds elapsed since the last time it reached a goal (or since the start of the episode if it has yet to reach a goal). The below figure demonstrates the reward over time. The faster the prey reaches the goal, the higher the reward is. This incentivizes finding the goal as fast as possible.

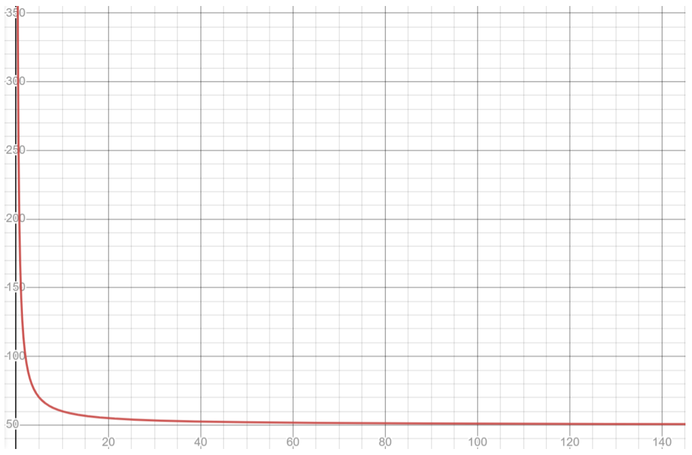

Fig. 8. Graph of Prey's Goal Reward Function (x is time, y is reward)

### D. Hyperparameters

During training the following hyperparameters were set:

```
batch_size: 128
buffer_size: 2048
learning_rate: 0.0003
beta: 0.01
epsilon: 0.2
lambd: 0.95
num_epoch: 3
learning_rate_schedule: linear
```

This is what each hyperparameter means:

- batch size: The number of experiences in each iteration of gradient descent.
- buffer size: The number of experiences to collect before updating the policy model.
- learning rate: The initial learning rate for gradient descent.
- beta: The strength of entropy regularization.
- epsilon: Influences how rapidly the policy can evolve during training.
- lambda (lambd): The regularization parameter.
- num epoch: The number of passes to make through the experience buffer when performing gradient descent optimization.
- learning rate schedule: How the learning rate changes over time.

## IV. Environment Setup

The environment consists of a 24x14 grid with 2 6x1 horizontal walls and a 1x10 vertical wall in the center. Predator

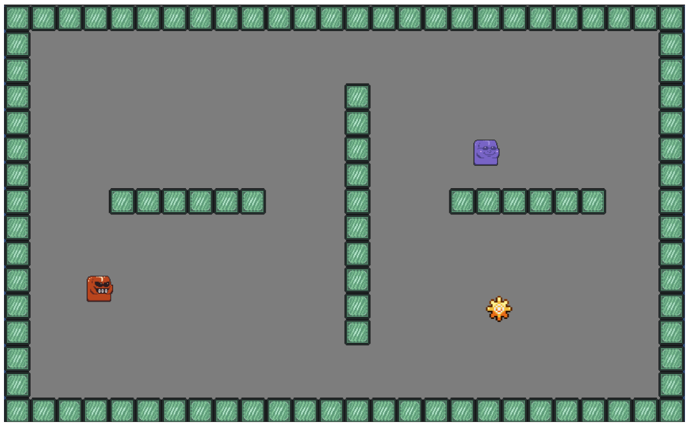

Fig. 9. Possible Starting Conditions in the Environment

and prey will always be spawned at least 5 units away from each other. If a goal is spawned, it will always be spawned at least 5 units away from the prey. This is in order to make sure an agent always has to move some distance in order to reach its goal.

## V. Simulations

Each simulation iteration will be explained in terms of what was added. Unless otherwise specified, all elements of previous simulations will be present.

## VI. Simulation I

### A. Movement

Predator and prey are both able to move horizontally and vertically, as defined in Agents Setup. The predator has a moveSpeed value of 5, while prey has a value of 6. prey was given a speed advantage in order to encourage the predator to develop strategies to catch the Prey.

### B. Observations

The predator has the following observations:

1) Predator position
2) Prey position

Similarly, the prey has the following observations:

1) Predator position
2) Prey position

This means that the prey and predator both know each other's location and their own.

### C. Rewards

Predator receives a reward for colliding with prey, as defined in Agents Setup (1). Similarly, prey receives a negative reward for colliding with a predator, as defined in Agents Setup (2).

### D. Other Notes

Please note goals for the prey have not been introduced yet. They will be added in later iterations. Furthermore, raycast observations are also not present.

### E. Results

The predator followed the prey around and was successful sometimes. However, as expected, it was unable to account for any walls in the environment and sometimes became stuck on them. As of now, the predator is simply moving in the direction of the prey. Hopefully in later iterations a more complex behavior can emerge.

The prey was relatively successful at running from the predator. It would "win", as in not get caught, by staying in the middle left or middle right of the environment while the predator is stuck on the other side of the middle wall. However, the predator was able to catch it when it moved into a corner. The prey's behavior seems to be to go in the same direction the predator goes in, as that is the optimal direction to increase distance between agents. However, this is extremely basic and lacks the ability to detect walls, leading in it sometimes running into a corner and trapping itself.

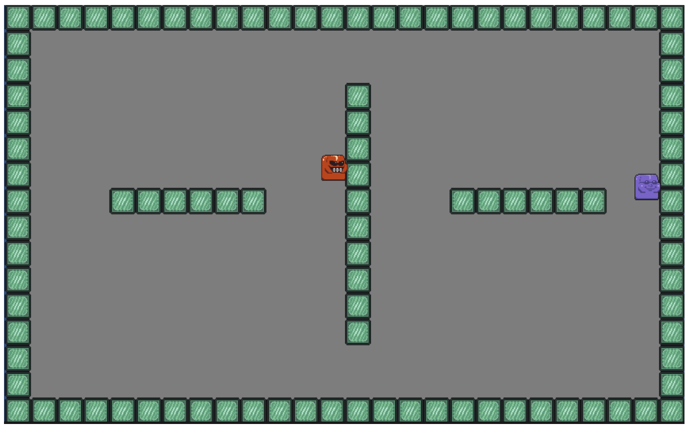

Fig. 10. Predator Stuck on a Wall

## VII. Simulation II

### A. Observations

Predator and prey now both have 8 direction raycasts. The raycasts point North, North East, East, South East, South, South West, West, and North West and have a maximum distance of 20 units. This is large enough to detect immediate surroundings in the environment. The specific representation of the raycasts is discussed in Agents Setup.

These raycasts allow agents to "see" the environment, which will ideally aid in developing more complex behavior.

### B. Rewards

The predator is given a reward of $-0.1$ for each frame it is colliding with a wall. This negative reinforcement was added in the hopes that the predator would develop an obstacle avoidance behavior using its raycasts.

A punishment wasn't added to the prey in order to allow for the prey to have more freedom of movement compared to the predator.

### C. Brain Settings

The neural network that represents the agent's brains was upgraded from 2 layers to 3 layers. This was done in order to allow for more complex behavior using the new raycast observations

### D. Results

The predator became very effective at catching the prey. Its behavior seemed to be to move towards the prey until it is close to a wall. Then, it would move in a direction until it no longer had a wall in the direction towards the prey. It would then start moving towards the prey and repeating as many times as necessary.

The prey did not increase much in effectiveness. They still go in the same direction the predator goes in to avoid them, and this leads to them being cornered and caught. However, an interesting behavior was observed where if a predator was near the middle wall, the prey would stay close to the other side of the middle wall to use the wall as a shield. However, the predator would simply go around the wall and eventually capture the prey.

## VIII. Simulation III

### A. Rewards

The prey receives a reward of $-0.1$ for each frame it collides with a wall. This is the same penalty applied to the predator. The goal of this reward is to encourage the prey to develop obstacle avoidance.

### B. Results

There were no major changes in the predator's behavior.

The prey vastly improved at avoiding the predator. It tries to ensure the direction it headed in would be clear of a wall when on the run from the predator. However, it wasn't perfect as it would sometimes bump into the border walls and get caught. Most prey were caught somewhere near the corner, as that was the most advantageous position for the predators.

## IX. Simulation IV

### A. Environment

Goals were added into the environment. Goals were spawned as described in Environment Setup. When the prey collides with it, it will be respawned in the same way it did at the beginning of the episode. There is no limit to the number of goals a prey can collect. Predators cannot collide with them, they would go through them.

### B. Observations

The prey now has the goal position observation. Interestingly, so does the predator. The hope was that the predator would use the knowledge of the location in order to predict the prey's movement and to create an interaction where the prey has to lure the predator away from the goal.

### C. Rewards

The prey receives a reward for reaching the goal. The reward function (3) is described in Agents Setup. The reward has a base of $+50$ and the base reward for getting caught is $-100$, making reaching the goal around half as rewarding as getting caught is punishing.

### D. Results

Despite the new knowledge of the goal's location, the predator's behavior didn't change much. It still decided to chase the prey in a similar fashion to how it did before.

The prey became very effective at evading the predator. Some unique behaviors were exhibited. One of them was when the prey and predator were on opposite sides of the 2 horizontal walls, the prey would go in the opposite horizontal direction of the predator such that the predator wouldn't be able to catch it by going around. Sometimes, the predator would notice and switch directions, and the prey would switch as well, continuing the cycle. There were times when the prey moved right past the goal without collecting it, likely estimating that the risk of getting caught wasn't worth the potential reward of collecting the goal. Because the reward of the goal was around half the punishment of getting caught, it incentivized the prey not to push their luck.

Overall, the prey exhibited far more complex behavior as it had to balance between 3 goals:

1) Avoiding getting caught by the predator
2) Reaching as many goals as possible
3) Avoiding colliding with walls

Conversely, the predator had more simple behavior as it only had 2 goals:

1) Reaching the prey
2) Avoiding colliding with walls

## X. Conclusion

Through each iteration, the total reward has changed a lot. from simulations one to two, the predator becomes far more effective at catching the prey, which results in a spike in reward and a corresponding dip in reward for the prey. After the prey learns to avoid obstacles (simulation 3), there are some episodes where it doesn't get caught, resulting in a reward greater than $-100$. With the addition of the goal (simulation 4), the prey's reward jumps to -2, which is due to the goal giving a reward. The predator's reward stays mostly the same, indicating that the prey didn't change in survival time much.

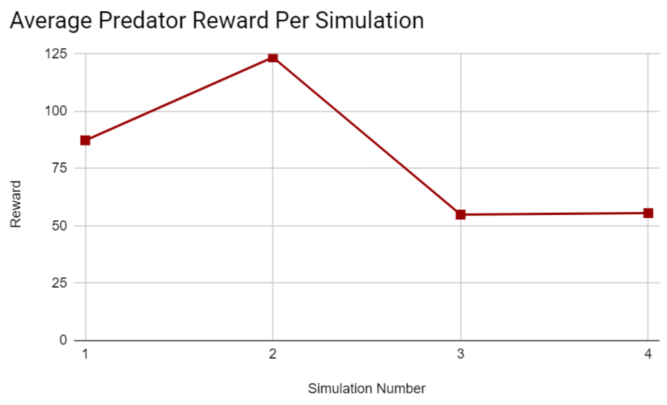

Fig. 11. Graph of Total Episode Reward for Predator

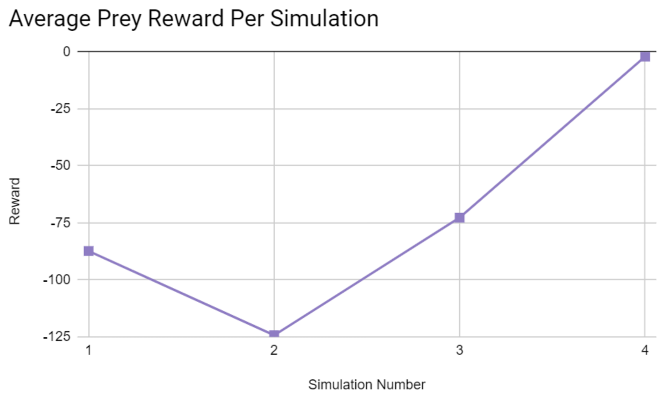

Fig. 12. Graph of Total Episode Reward for Prey

Starting from only a few simple rules and a basic environment, complex interactions between two agents emerged. In the first simulation, the predator simply moved toward the prey, and the prey simply moved away. By the final simulation, far more complex behaviors emerged, such as the prey "faking out" the predator when it was on the opposite side of a wall by going in one direction and then switching in order to leverage its higher speed and run past the predator. The results suggest that one way to create a sophisticated AI is to make it balance a few simple goals.

Furthermore, when designing Proximal Policy Optimization (PPO) systems, it is important to get the correct numbers. PPO doesn't have any sense of what the user wants it to do, it simply does math. For example, when the reward for colliding with a wall was $-0.01$ rather than $-0.1$, the punishment was negligible enough for the agent to mostly ignore it. In addition, had the goals been a higher reward than the punishment for getting caught by the predator, the prey would engage in far more risky behavior as the goal is far more rewarding than the predator is punishing.

It should also be noted that hyperparameters are very important for proper training. In an earlier version of the project, the beta (exploration rate) was set too low and the agents didn't explore enough options to decide on a good algorithm. This resulted in agents that would pick a direction (left or right) and move in it. This ended in them sticking to the wall for the rest of the episode. This likely occurred as sometimes that direction would result in a victory for either side, so it kept with it as a relative maximum. Increasing the beta fixed this issue.

## XI. Future Work

In the future, this project can be expanded in several ways. For example, randomizing wall locations would create more complex terrain and force the emergence of more complex path-finding. In addition, multiple predators and prey would lead to more interesting dynamics. Perhaps the predators would have individual scores, but the prey would share a team score. That would necessitate mechanics that complement team strategy, such as the ability to manipulate surroundings in some way.

## References

[1] Baker, Bowen, et al. "Emergent Tool Use From Multi-Agent Autocurricula." ArXiv, 2019.

[2] Tampuu A, Matiisen T, Kodelja D, Kuzovkin I, Korjus K, Aru J, et al. (2017) Multiagent cooperation and competition with deep reinforcement learning. PLoS ONE 12(4): e0172395. https://doi.org/10.1371/journal.pone.0172395

[3] N. A. Barriga, M. Stanescu, F. Besoain and M. Buro, "Improving RTS Game AI by Supervised Policy Learning, Tactical Search, and Deep Reinforcement Learning," in IEEE Computational Intelligence Magazine, vol. 14, no. 3, pp. 8-18, Aug. 2019, doi: 10.1109/MCI.2019.2919363.

[4] Firoiu, Vlad, et al. "Beating the World'S Best at Super Smash Bros. with Deep Reinforcement Learning." ArXiv, 2017, /abs/1702.06230.