Dungeon Escaping Agents Using Machine Learning & Collaboration

Rebecca Lin, Jay Shrestha

Abstract

We are researching the application of Machine Learning in collaborative agent simulations using Unity Engine's ML Agent package. We will be using the MA-POCA learning system to train agents to collaborate with each other to complete a task that requires them to defeat a dragon in order to retrieve a key that'll allow them to exit a dungeon. The simulation parameters will be designed such that groups will be rewarded based on how well agents perform collaboratively.

1 Introduction

The motivation behind this project comes from the previous implementation of Pacman. We worked on reinforcement learning agents for pacman and the ghosts where they would co-evolve using Unity's ML Agents package, specifically using a technique called Proximal Policy Optimization (PPO). We found that many of the ghosts were not as collaborative as we hoped and wanted to take on the challenge of making collaborative machine learning agents.

This project will be using several packages across different platforms. The simulations will be run in the Unity Engine, specifically version 2022.3.8f1 and has a few packages being utilized beyond what is installed by default, with several of them handling the running of the simulation environment such as the AI Navigation for the dragon who does not utilize machine learning but will instead move towards one location using a Navigation Mesh. The packages that will be directly responsible for the implementation of Machine Learning is the ML Agents package which will allow us to interface with Python packages that control agents with a neural network that can be trained in simulations. An additional Unity package being used will be Unity Sentis which is an experimental package that embeds an AI model, neural network, in Unity Runtime. As aforementioned, the ML Agents package will be interfacing with a Python package which will be responsible for the development of the neural network, taking the inputs of observations we give from the simulation and outputting actions for the agents to perform. We will be initiating the Python functionality through Anaconda.

We will be presenting the MA-POCA learning system, which stands for Multi-Agent POsthumous Credit Assignment, which is a learning system that naturally handles agents who are created and destroyed within an episode though all share a reward function. Luckily this architecture scales easily for additional agents, which works well for this situation where we will have several dungeon simulations running simultaneously, each of which will have three agents working together.

2 Original Simulation

Original Simulation Design

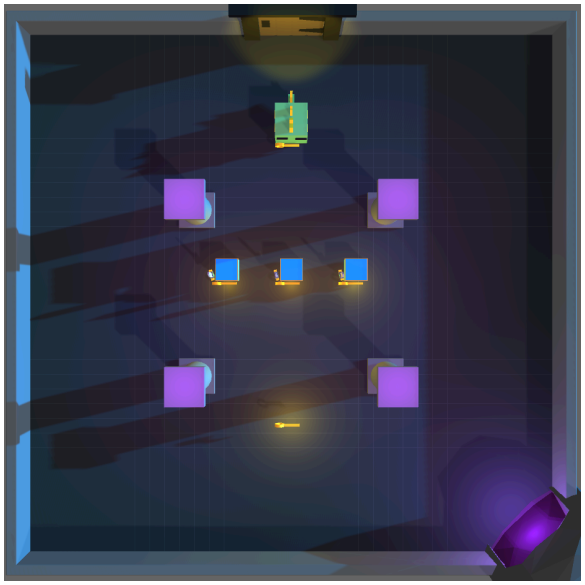
The original simulation for this Dungeon Escape had one 'dungeon room' designed as a square with four pillars in the center forming a square, with an exit door on one wall and a cave entrance across from it at a corner.

In the original simulation there was one dragon whose behavior would be to move in the direction of the cave entrance with the intent of reaching it. Upon reaching it, the simulation would be reset with no rewards given to the agents, as this scenario would be counted as a failure on the agents' part as it is their assessment to defeat the dragon which requires defeating it before it reaches the cave, providing a sense of urgency to the AI agents.

The design of the agents was that they could recover the door's key by combating the dragon which occurred when one agent would collide with the dragon. Upon collision, both the agent and the dragon would 'die,' which means they would be set to be deactivated and would therefore stop partaking in the current episode. At the same time, a key object would be dropped on the dragon's location of death. This object could be picked up by other agents, and if an agent has the key with them, they can unlock the exit door if they reach it, allowing them and the other living agents to escape. Upon exit of the cave, the agents would receive a base reward of 1 as a positive

reinforcement for managing to properly exit the cave. Agents were spawned at random locations on that map so that it focuses on agent collaboration and agents don't just learn what to do from their specific location.

Original Map



Original Simulation Results

The resultant neural network developed over training from this original simulation led to the agents having a general procedure:

- An agent would seek out the dragon and collide with it, therefore killing the dragon, dropping the exit key, but sacrificing the agent in the process
- One of the remaining agents would find and collect the key off the dragon's death location.
- The agent with the key would navigate to the exit door and exit with the key.

This was generally how the agents performed with the neural network gained when running the original simulation, though there were issues with this design that we hoped to improve upon in this research.

Original Simulation Flaws

One flaw of the original simulation was that the neural network produced was not adept at handling scenarios where agents were in a room with more complex obstacles, as the single room they were trained in had little obstacles with the four pillars in the center doing little to obstruct their path, resulting in their movements being able to be nearly fully free. This leads to the agents only performing well under rather basic conditions and not being able to handle more complex environments.

Another flaw that only worked with this simple design was the simple navigation of the dragon. Given that the dragon only moved in the direction of the cave entrance, it only ever made a straight line path to its destination, making it very easy to defeat for the agents as there was no complexity in its movements. This resulted in a rather weak training ground for the agents.

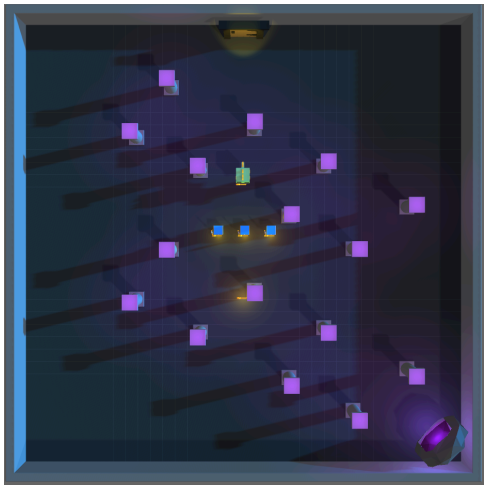
Lastly and perhaps the largest problem, the agents were not incentivized much to properly collaborate with each other. Their behaviors led them to function soloistically, with one killing the dragon and another picking up the key. The agents never really needed to act together, one would just wait for another to complete its action. The fact the dragon could also be defeated in a single collision by an agent also downplayed the necessity for collaboration, as a single agent could act independently and still destroy the dragon, not requiring multiple agents to fight the dragon together. The requisite for an agent's sacrifice also felt flawed in that it required a loss of collaboration, for agents to need to remove themselves, though the reward design saw no issue as the agents would receive the same base reward for simply exiting the dungeon. This led to an almost sociopathic behavior of the agents to disregard each other's lives, letting one sacrifice themselves so the others could escape without any means to protect each other.

3 Simulation Improvements

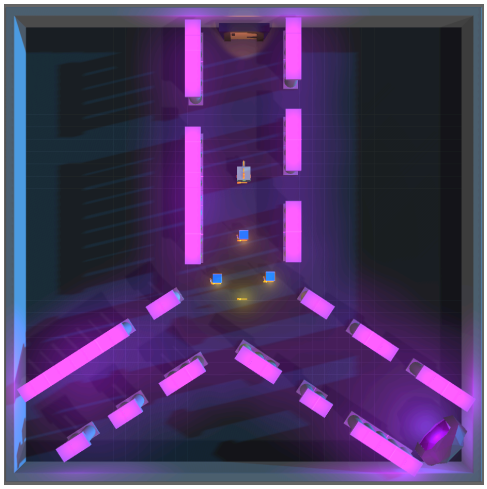
Map Improvements

In terms of improving the map, we replaced the single simple map with five different maps that had more various environments that better challenged navigation. Each of these maps had diverse designs with some that had scattered pillars to challenge close navigation around obstacles with others having designs more resembling mazes to test the agent's ability to navigate a complex space. Each of these maps add more nuance in the dungeon simulation and better train the agents to adapt given the harshe environment.

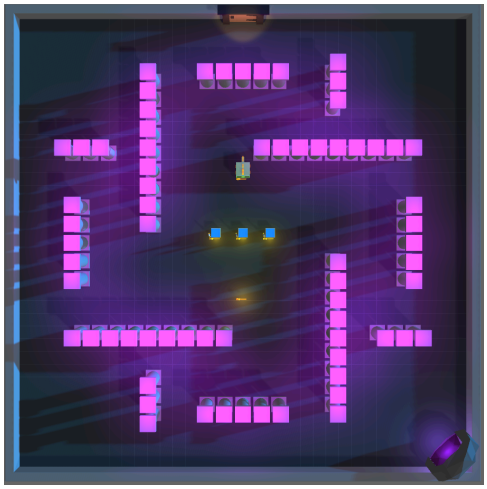
Map 1: Scattered Pillars



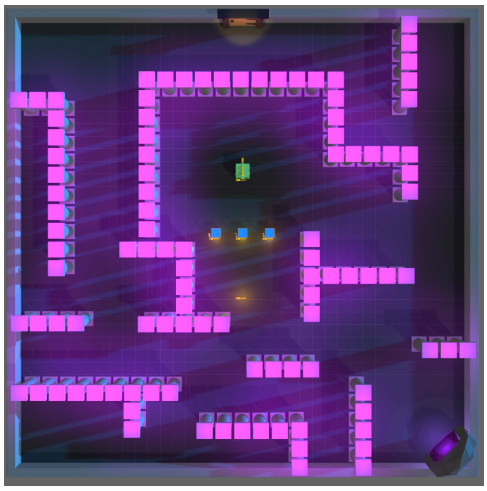
Map 2: Intersection



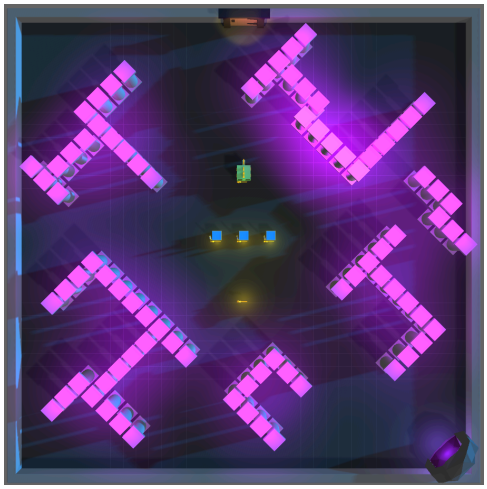
Map 3: Walls and Obstacles



Map 4: Maze



Map 5: Angled Obstacles



Dragon Improvements

Another improvement was to the dragon, which in a way was necessary given the aforementioned change. Instead of the dragon simply moving in the direction of the cave, the dragon operated as an agent of a Navigation mesh given to each dungeon room, using the mesh to determine a path for the cave entrance to more intelligently move in its direction. This resulted in the dragon not having just a single straight line path, but rather now the dragon can move with twists and curves to navigate around the complex obstacles, further testing the agent's ability to locate and destroy the dragon.

The dragon was still defeated from collisions with the agents, and upon defeat the dragon would be vanquished and would drop the key to the

exit door. One change made however was that the dragon would not be destroyed upon a single collision with an agent, but instead it required to be collided with twice in order to be defeated and for its key to be dropped. This resulted in the dragon not being easily defeated by a single lone agent, and added further challenge for the agents to overcome. It removed the simplicity of soloistic action from the agents, causing two of them to perish if they each fought the dragon alone.

Agent Improvements

At the same time, the agent may not necessarily be sacrificed given the condition that when they collide with the dragon, they are at that moment fairly proximate to an ally agent. If they are near an ally when they collide with a dragon, the dragon will still suffer the collision and take 'damage' from it, but the agent who collided with the dragon will remain alive and not be destroyed together.

This change now allows for agents to corroborate together when dealing with the dragon, able to change the outcome while still being able to defeat the dragon alone if they so wish to as to not radically break the original simulation's intent.

The additional change to the agents was their group reward though, as rather than receiving a basic reward of 1 for completing the room, they are instead given a reward that factors in the amount of agents who survived to reach the dungeon's exit, as a reward of 1 will be given if only one agent survives but a reward of 2 will be given if 2 survives, and 3 if all do. This results in the agents being rewarded for properly collaborating with each other and protecting each other from the dragon's collision, resulting in behaviors that show the agents remaining closer together and acting synchronously rather than simply waiting for one to sacrifice themselves before acting.

Collision Handling Pseudocode

Algorithm: Agent-Dragon Collision

Input: agent, enemy
Parameter: agent is the agent that has collided with the dragon; enemy is the dragon
Output: Properly destroys enemy and agent if appropriate

```
PROCEDURE KilledByEnemy(agent:
PushAgentEscape, enemy: Dragon)
1:    isProtected ← FALSE
2:   FOR EACH ally IN allies
3:       IF agent.gameObject ≠ ally THEN
4:          IF
Distance(agent.gameObject.transform.position,
ally.transform.position) ≤ protectionRange THEN
5:              isProtected ← TRUE
6:              BREAK
7:          END IF
8:       END IF
9:   END FOR

10:    IF NOT isProtected THEN
11:       numberOfRemainingPlayers ←
numberOfRemainingPlayers - 1
12:       agent.gameObject.SetActive(FALSE)
13:       PRINT enemy.gameObject.name, "ate",
agent.transform.name

14:       IF numberOfRemainingPlayers ≤ 0 THEN
          agentGroup.GroupEpisodeInterrupted()
15:          ResetScene()
16:       END IF
17:    END IF

18:    EnemyHealth ← EnemyHealth - 1

19:    IF EnemyHealth ≤ 0 THEN
20:       enemy.gameObject.SetActive(FALSE)

21:Key.transform.SetPositionAndRotation(enemy.coll
ider.transform.position,
enemy.collider.transform.rotation)
22:       Key.SetActive(TRUE)
23:    END IF
24:END PROCEDURE
```

---

4 Training

Procedure

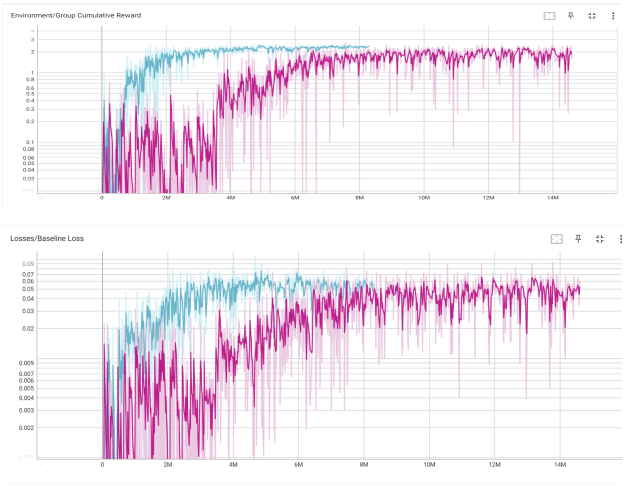
For training, we had each of the unique dungeon rooms in the same scene each with a set of agents and a dragon, functioning as individual dungeon simulations which were all run concurrently in the single Unity scene. The agents were trained without any initial models, and they were given at least eight hours of training each on average.

The main differences between the training sessions was the learning rate as we trained with three different learning rates: 0.0003, 0.0005, and 0.00003. Each of these learning rates were used for training to test different parameters of the training to gauge how it would affect the agent models. After training, we acquired graphs from tensorboard on the progress of the training.

We used the 0.0003 learning rate as the baseline model. We found that the agents performed well and the results plateaued around 2.3 for the group cumulative reward. Rewards are based on how many agents there are left when they defeat the dragon so the highest reward is actually 3.0 and the lowest reward is 0. This means that on average 2.3 agents survive each dungeon escape which is relatively successful.

Tensorboard Graphs

Learning Rate 0.0003 & 0.00003



Blue: 0.0003
Pink: 0.00003

Both of the training sessions proved effective, with the cumulative reward increasing greatly until reaching a plateau that was satisfactory. The main difference is that whereas the session with the baseline learning rate of 0.0003 showed a quick improvement that rose up immediately and had a rather steady rise and plateaus, the session with the lower learning rate of 0.00003 proved to have a far more chaotic experience as the rewards accumulated constantly rose and dropped over the course of the first quarter in training, and only after was there a consistent rise and plateau.

This behavior can also be examined with the Baseline loss, as while both sessions proved to have an effective improvement in baseline loss before plateauing, but whereas the baseline learning rate yielded a consistent and smooth progress, the lower learning rate yielded one that was far more chaotic with sporadic spikes and drops though it did eventually climb to a satisfying level where it plateaued.

Learning Rate 0.0003 & 0.0005

Blue: 0.0003
Pink: 0.0005

Both of these training sessions proved to be very similar since they both plateau around 2.3. The big difference between these two training sessions is the stability of the agents. With the learning rate of 0.0005, we saw dips in group reward around the 15M Step and the 21M Steps whereas in the 0.0003 you don't see any of that. Additionally, the 0.0003

learning rate climbed more consistently while the 0.0005 had more variance at the beginning.

The baseline losses for both were relatively similar as well. The 0.0005 again has more variance at the beginning stages but was actually lower than the 0.0003. They both plateau and hover around the same baseline loss.

5 Training Results

Model Assessment

After the completion of the training, we acquired the models generated and equipped the player agents with them in the Unity scene, allowing them to run the simulation without training but rather guided off of the produced models. The result we were provided was generally agents who were capable of escaping the dungeon with fairly low casualties. Simply completing the dungeon was quick for them as the agents were fast at chasing down and killing the dragon, picking up the key, and reaching the exit door, all procedures that were examined even in the original model.

The differences however lie in the fact that whereas the original model showcased single agents hunting the dragon themselves, resulting in their sacrifice, our produced models resulted in multiple agents moving together in hunt for the dragon, thereby protecting each other and resulting in no casualties.

Of course this was not always perfect, as there were instances in which agents strayed too far from allies and were sacrificed when combating the dragon, however these were infrequent and more often than not casualties were avoided. Instances where allies were sacrificed mainly occurred when agents were spawned far away from each other. Furthermore there were a few times when the dragon would manage to escape, however these were very infrequent.

Behavioral Differences from Map to Map

There are noticeable deviations in the effectiveness of agents from the different maps, at least most noticeable in the efficiency of navigation from the agents. In the simpler maps with more sparse pillars, the agents had little difficulty in navigation and were quick to move from the dragon and then to the exit. However, in other rooms such as the mazes that had walls of pillars forming strict corridors, the agents had more complex paths they formed, sometimes moving around the edge of the room to reach the exit.

It should be noted that even with the more complex navigation, the agents were still well able to navigate the room as necessary to both hunt the dragon and reach the exit door.

6 Conclusion

Future Work

To further explore this research and develop this Machine Learning experiment in a meaningful way, we could for one develop additional dungeon rooms to add more diversity in the environments the agents would train in while also offering more unique navigation challenges.

We could also incorporate more challenges in terms of enemies, and have more than merely the single dragon but instead either multiple dragons or other smaller enemies that need to be dealt with, dividing the agent's objectives into multiple concurrent ones which they need to manage to collaborate through. These enemies could have more complex behaviors instead of the dragon merely walking towards the exit, as in fact the dragon could be improved to provide a direct combative threat to the agents and further add difficulty to test their cooperation.

Lastly the concept of a more complex combat system was thought of during this research and for future work this could be revisited. The concept would be a combat system for the agents that added additional complexity such as time between strikes, stamina, and attack durations, and other parameters that further added dimensionality to the agent combat which could insight collaboration. For example, if agent attacks were slower, multiple agents could be advised to work together in each attacking sequentially as to deal rapid damage in a way single individual agents couldn't. Generally there are several directions this could be taken, but it would all be in service of providing a new challenge to the agents which could be overcome with teamwork.