Agent Chase: Training Opposing Agents in Unity

Michael Figueroa Nivedha Sreenivasan Ahmed Siddiqui Vaishnavi Josyula

Introduction

The integration of artificial intelligence into video games has led to the simulation and development of new environments in game engines, notably when training one or more agents using reinforcement learning techniques. These agents are now capable of learning the rules of a given game, adapting to complex environments, and ultimately playing against each other.

We trained two AI agents to play the game of Agent Chase using the ML-Agents framework on the Unity game engine.

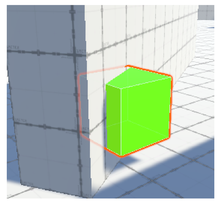
Rules

The premise of Agent Chase is for the goalie to defend the goal while the runner tries to avoid the goalie while running to the goal.

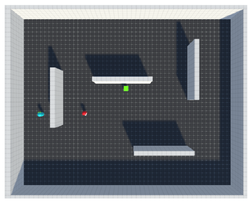
1. There are two players: the runner and the goalie. These two players, along with a goalpost, will spawn randomly in the map.
2. The runner's objective is to get to the goal and touch it before being caught by the goalie as fast as possible.
3. The goalie's objective is to catch the runner before the runner can reach the goal.

Environment Design

We first designed a map on Unity that the two agents would train in. The map included obstacles, the runner, the goalie, and a goal for the runner to reach. Obstacles did not move between episodes, but the location of the Goalie, Runner, and Goal were randomized with each episode. The main challenge was preventing these randomized objects from spawning within obstacles.



(a) Goal spawn-clipping into a wall.



(b) Full room with the goal, goalie, and runner.

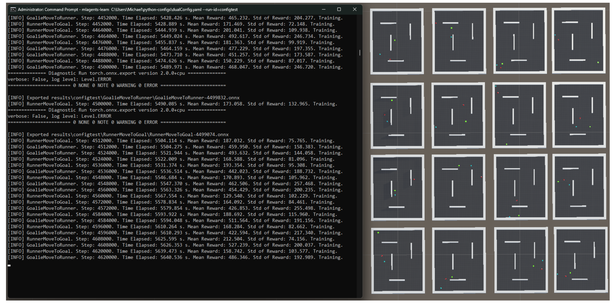
ML-Agents and Unity

Unity is a real-time development platform that offers tools for the production of 2D and 3D applications, simulations, and games. ML-Agents is a package used within the Unity engine to create a machine learning environment that trains "Agents" to interact with their surroundings.

Deep Reinforcement Learning (deep RL) is a subset of machine learning that involves training agents to learn and make decisions in complex environments through trial-and-error. Many deep RL algorithms, like Proximal Policy Optimization (PPO), use a reward system to "teach" agents about the game state – a positive reward encourages a certain behavior while a negative reward discourages said behavior. [1]. [2]. [3]. [4].

Training Environment

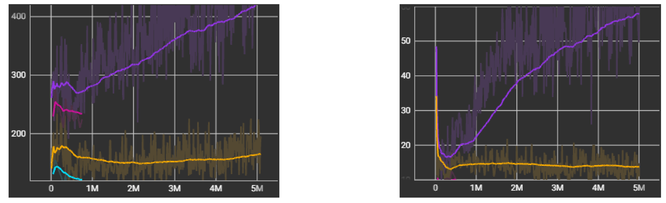
After creating a python environment to train the two agents, we multiplied our Unity environment to run several instances simultaneously. Before the training, the rewards were tweaked to encourage favorable behaviors and discourage unfavorable behaviors for each agent.



(a) Training simultaneous environments.

To train the runner and the goalie, we created two agent classes. These classes are ed after their main goals in the game and contain almost all the behaviors, rewards, and observations for the agents. The observations allow them to detect walls, the opponent, and the goal through raycasting. For reinforcement learning to take place, a rewards system is used, with different rewards for actions like walking away from the target, walking towards the target, winning, losing, hitting an obstacle, and the passage of time.

Analysis



(a) Rewards accumulated over time. (b) Loss-value over time

Figure 3. Results from our first training session.

The success of the runner (in orange) and the goalie (in purple) were vastly different. The goalie demonstrated a steady increase in cumulative reward with further training, indicating improvement over time. However, the runner struggled to increase its rewards function. This demonstrated a core issue at the heart of training asymmetric multi-agent models; one side can quickly outperform the other, which can heavily impede progress on one side. This is also shown by the value-loss graphs. The goalie's value loss increased over time, while the runner's value loss remained low and steady, implying that the goalie was improving faster than the runner could predict its movements. Potential solutions would be to adjust the reward values, pausing goalie training while the runner still learns, or simply letting it run for longer periods of time.

Conclusions

The Unity Engine proves helpful in training AI within a 3D environment. It not only supplies the tools for building an environment, but it also provides a machine learning package so that AI doesn't need to be coded from scratch. This package can be used for multi-agent reinforcement learning, even when they are asymmetric and opposed to each other. In this project, reinforcement learning proved to be useful in teaching the agents the rules of the game using a rewards system, although one side quickly overpowered the other, leading to difficulties in training. Tweaks to the reward system and altering training processes may resolve this discrepancy, which is something to experiment with in the future.

[1] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *CoRR*, abs/1809.02627, 2018.

[2] V. Kononen. Asymmetric multiagent reinforcement learning. pages 336–342, 2003.

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[4] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *CoRR*, abs/1911.10635, 2019.

Team Lead: Naveen Mukkatt, Faculty Advisor: Dr. Nicholas Ruozzi