Interactive Quest-based Combat Game OriginX

Damini Cousik
Computer Science Department
University of Southern California
cousik@usc.edu

Srujana Subramanya
Electrical Engineering Department
University of Southern California
srujanas@usc.edu

Raghavendra Vedula
Computer Science Department
University of Southern California
rvedula@usc.edu

Akshata Sheth
Computer Science Department
University of Southern California
shetha@usc.edu

Kaushik Kiran Karalgikar
Computer Science Department
University of Southern California
karalgik@usc.edu

Darshan Hassan Shashikumar
Computer Science Department
University of Southern California
hassansh@usc.edu

***Abstract*—Reinforcement learning has proven to be very successful in static game environments and board games. It has surpassed human-level performance in fixed game environments and turn-based two player board games. In recent days, Reinforcement learning is extending its boundaries to a variety of game genres. In this paper, the goal is to create an Interactive Quest-based Combat Game that was enhanced by interaction with a machine learning agent. This is achieved by training two bots at each level to varying extents using Reinforcement Learning. The players' experience will be defined by their interactions with the agent.**

I. Introduction

The driving force behind this interactive quest-based combat game is Reinforcement Learning. Over the years, reinforcement learning has materialized substantially and is now dominant in many fields such as game theory, control theory, operations research , etc.

In game theory, basic reinforcement is modeled as a Markov decision process (MDP). Fighting games—as one of the most representative types of complex PVP (player versus player) games—have been the focus of multiple studies that have made progress in this area. Imitation Learning (IL) is used to efficiently write reward functions. The simplest form of imitation learning is behavioral cloning (BC), which focuses on learning the expert's policy using supervised learning. These strategies are used to train 2 AI bots to help play against a human player at each level. The extent of training for the bot will vary for each level which is dependent on the performance of the player. The goal of the player is to defeat the bot and conquer all the worlds to finally destroy and recreate a new world.

II. Background and Motivation

Reinforcement learning (RL) is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision making problems in a wide range of fields in both natural and social sciences, and engineering.[1] Reinforcement learning has steadily progressed its capability from last decade, as trained systems have surpassed world champion human players in the games of poker, Go, and Chess. Moreover, OpenAI and DeepMind have achieved significant milestones in real time strategy. These latter accomplishments are of particular interest to the military domain, as RTS games share many characteristics with constructive simulations and war games. However, unlike military constructive simulations, RTS games abstract many important aspects of combat, to include intelligence and logistical functions.[2]

This paper introduces the game OriginX which is a user interactive quest-based combat game. At every level, the player has to fight with an AI agent and successfully defeat the agent. The goal of OriginX is to develop a combat game where the player fights against an AI agent to defeat the agent and advance to the next level. The player wins the game by defeating the AI agent at every level.

III. Prior Research

Fighting games find their origin in boxing games but evolved towards battles between characters with fantastic abilities and complex special maneuvers. Sega's black and white boxing game Heavyweight Champ, which was released in 1976 is one of the oldest known combat games. Artificial intelligence has been an integral part of video games since their inception in the 1950s. AI in video games is a distinct sub field and differs from academic AI. It serves to improve the game-player experience rather than machine learning or decision making.[3]

Several methods of machine learning have been used in combat games.A lot of varied approaches have been proposed for reinforcement learning with neural network function approximates. Deep Q-learning, "vanilla" policy gradient methods, and trust region / natural policy gradient methods are some of the leading contenders in this space. However, there is room for improvement in developing a method that is scalable, data efficient, and robust. Hence, Proximal Policy Optimization is the best way to implement this as it encompasses all these conditions.

Proximal Policy Optimization performs comparably or better than state-of-the-art approaches while being much simpler to implement and tune. PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance.[4] PPO has been used widely in many games with great success like puzzle games (Lily's Garden from Tactile Games), Atari Breakout game etc. AWS DeepRacer uses the Proximal Policy Optimization (PPO) algorithm to train the reinforcement learning model. PPO uses two neural networks during training: a policy network and a value network. The RTS game StarCraft I has been used as a platform for AI research for many years. Many multi-agent reinforcement learning algorithms have been proposed to learn agents either independently or jointly with communications to perform collaborative tasks. PPO lets us train AI policies in challenging environments, like the Roboschool where an agent tries to reach a target (the pink sphere), learning to walk, run, turn, use its momentum to recover from minor hits, and how to stand up from the ground when it is knocked over.

The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents. Agents can be trained using reinforcement learning, imitation learning, neuroevolution, or other machine learning methods through a simple-to-use Python API.[5] The ML-Agents Toolkit allows to use pre-trained neural network models inside our Unity games. This support is possible thanks to the Unity Inference Engine (code named Barracuda). The Barracuda package is a lightweight cross-platform neural network inference library for Unity.

Unity ML-Agents toolkit for Proximal Policy Optimization is used for giving rewards or punishments to the agent based on its number of attacks it makes and defends.

IV. Methodology

A. Unity

Unity is one of the best free cross-platform game engines. At the start of the game, the enemy agent advances towards the player and each of the two characters who are in combat, the player and the AI agent have two attack animations and one defence animation. They can also move around in the arena. When the enemy AI agent is within a certain range of the player, it starts attacking the player. In this game, Unity's Navmesh Agent is used to define the movements of the characters in the arena.

1) Navmesh Agent: NavMesh Agent for enemy bot is used to navigate its way around the arena quickly. NavMesh agent is an easy and quick way for moving characters around a scene and finding paths. Agents reason about the game world using the NavMesh and they know how to avoid each other as well as other moving obstacles. Pathfinding and spatial reasoning are handled using the scripting API of the NavMesh Agent.

Navmesh Agent is used to set the destination for the AI agent as the player's position. The enemy AI agent then chases towards the player and when it is within a defined range, it starts attacking the player.

B. Unity ML - Agents

1) Proximal Policy Optimization on Navmesh Agent: Proximal Policy Optimization (PPO) is used to determine what better actions a software agent should choose in a given environment in order to maximize reward.

PPO uses a novel objective function and this objective implements a way to do a Trust Region update which is compatible with Stochastic Gradient Descent, and simplifies the algorithm by removing the KL penalty and need to make adaptive updates.

$$L^{CLIP}(\theta) = \hat{E}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

θ is the policy parameter

ϵ is a hyper-parameter, usually 0.1 or 0.2

$\hat{A}_t$ is the estimated advantage at time t

$\hat{E}$ denotes the empirical expectation over time steps

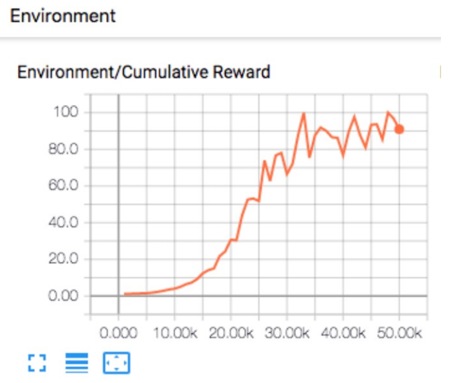
r_t is the ratio of the probability under the new and old policies

In-order to train the model against the Navmesh Agent, the below steps are followed.

- Assign the enemy character to be our Navmesh agent.
- Train the player character on this Navmesh agent using Proximal Policy Optimization.
- Use the trained model for the enemy agent.

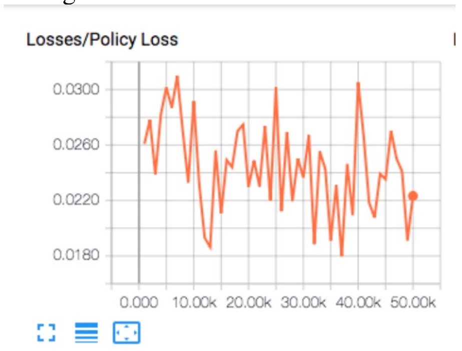
V. Experimental Results

Promising results are observed by using Proximal Policy Optimization against the Navmesh agent. Shown below is a graph of the Environment vs Cumulative Rewards for the agent:



A. Environment/Cumulative Reward

The mean cumulative episode reward over all agents. As this graph is consistently increasing, this model is well suited for this game.



The Losses vs Policy Loss (PPO; SAC) is calculated - This is the mean magnitude of policy loss function. It correlates to how much the policy (process for deciding actions) is changing. If the magnitude of this is decreasing then it would mean that the model is training well. As results agreed with the metrics for the graph, this solidified the reason for using Proximal Policy Optimization.

VI. Future Scope and Improvements

A. Interactive version

Now that this model is trained and applied to enemy agent, going forward the player controls over character and the player can then combat against the enemy and defeat the agent successfully.

B. Dynamic Difficulty Level Generation

Introduce different levels of the game. The player will have to defeat the enemy at each level and collect rewards from each level. At each level the agent will be trained to varying extents. The training of the agent will increase with each level. Thus it will get more difficult for the player to defeat the enemy AI agent with every increasing level with the last level being the toughest. Implement Generative Adversarial Networks to dynamically generate new levels and environments for the game.

References

[1] Yuxi Li. ”Deep reinforcement learning: An overview”. In: arXiv preprintarXiv:1701.07274 (2017).

[2] J. Boron, “Developing combat behavior through reinforcement learning.” M.S. Thesis, Dept. of Comp. Sci., NPS, Monterey, CA, USA, 2020.

[3] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction MIT press, 2018.

[4] John Schulman et al. ”Proximal policy optimization algorithms”. In: arXiv preprint arXiv:1707.06347 (2017).

[5] Carlos Florensa et al. ”Automatic goal generation for reinforcement learning agents”. In: arXiv preprint arXiv:1705.06366 (2017).