CSCI 599: Applied ML for Games

Final Project Report

FiRescue

Professor: Michael Zyda

Date: April 28th, 2020

Project By:

Anthony Prajwal Prakash (prak127@usc.edu)

Ayush Bihani (bihani@usc.edu)

Deepthi Bhat (dbhat@usc.edu)

Karthik Bhat (kabhat@usc.edu)

Nisha Mariam Thomas (nfnu@usc.edu)

Table of Contents

Abstract

Artificial Intelligence has become a necessary ingredient for the gaming industry to make the games more challenging and entertaining. Machine Learning is the superset of these AI technologies which gives the system the ability to learn and improve from experiences. Games whether developed for the purpose of entertainment, education or simulation, provides great opportunity to apply machine learning. With Machine learning techniques being substantially researched and developed, it provides a way to improve games by going beyond scripted interactions.

The recent improvements in GPU processing speed and the large amount of data that is available helps in building Machine learning models which becomes the core of interactive systems that are responsive, adaptive, and intelligent. The ML models are trained by learning the behavior of the players during the game play and adapt their own behaviors beyond the pre-programmed rules provided by the game author. This provides a way to improve the behavioral dynamics of the computer-controlled games by facilitating the automated generation and selection of behaviors and hence providing the opportunity to create more engaging and entertaining game-play experiences.

The use of artificial intelligence specifically Reinforcement learning in the gaming industry in recent years paves a new path of developing better non-playable characters. OpenAI, a pioneer in Reinforcement learning have developed several classes of algorithms and AI bots that have managed to beat top players in Dota2, Go board game, StarCraft, etc. These models are now being used in a wide variety of games, simulations, testing behavior of robotic agents in real world.

More importantly, this serves as an avenue of developing smarter non playable characters. Instead of defining a fixed set of actions it can perform, it can be trained on the game environment to learn wider ranges of actions that may not have possibly been defined by a human. For example, an AI bot learned a move in Go board game, not foreseen by the human player, that led it to win the game. Such progress in this field encouraged us to develop a unity-based simulation of an agent in an environment, very much applicable to the real world.

Background and Motivation

The Idea for the project was motivated from the recent devastated bushfires which occurred in the forests of Australia. The fires burnt an estimated 18.6 million hectares, destroyed around 5900 buildings and killed at least 34 people! Koalas have been hit particularly hard by Australia's raging bushfires. It is estimated that a third of New South Wales' koala population has died, while numbers are higher in other states. With an approximate population of 100,000 to 200,000 in the wild before the fires, koalas were already on the verge of extinction.

Koalas are also known to breed so slowly that it could take 100 years for the population to rebuild. This means that saving even one is very crucial. The National Disaster Search Dog Foundation, a California-based organization, said in a statement that search dogs can sniff out scents most humans can't detect, making their noses "a critical, life-saving tool to help the koala population survive." The forest rescue team in Australia used trained dogs to save Koala bears which were stuck in the bushfire. The trained dogs were able to identify the vulnerable Koala bears in the fire by the bears' cry noises and saved them by carrying them with their mouths out of the dangerous areas.

This served as the main motivation for our project. A trained robot (like a search dog) could help the government and fire-fighters perform critical tasks and help save lives! We worked on developing a game to simulate the bush fires and train a dog agent to perform critical search and rescue tasks without hurting itself. This training could be later imparted to a robot which could do the same.

Prior Work

Prior work on this subject has been documented in a project called G.E.A.R - Garbage Evaporating Autonomous Robot, conducted by - The Chair for Computer Aided Medical Procedures & Augmented Reality at the Technical University of Munich. Motivation for this project was to use autonomous robots to clear out massive amounts of garbage generated each day of the Oktoberfest. A swarm of synchronized, autonomous robots were capable of outperforming human workers.

The first step was to simulate the robot using a Unity 3D game engine. Also, a Unity Machine Learning Agents Toolkit (ML-Agents) plug-in that enables game scenes to serve as environments for training intelligent agents was used. This allowed the user to train algorithms using reinforcement learning, imitation learning, neuro-evolution, or other machine learning methods through a simple-to-use Python API.

The setup for the agent was a Bavarian-themed room. The goal of a robot was to explore the environment and learn the proper reasoning (policy), which was indirectly enforced on G.E.A.R through a set of rewards and punishments.

The goal of the robot was:

- to approach and gather the collectibles (stale loaves of bread, red plastic cups and white sausages).
- not to collide with static objects (chairs and tables), slamming against the wall or collecting wooden trays (they belong to the owner of a Bavarian tent and should not be collected by the robot for future disposal).

The agent was trained using several algorithms such as PPO, PPO with segmentation network, Behavioral cloning and Heuristic based algorithms. A simulation of the autonomous robot was created and tested in a custom-made environment. However, there were still plenty of improvements to be made to convert the entire project into an actual product that brought about business value. This project provided us with some motivation and idea to implement a similar strategy to save animals from bushfires.

Overview of the Game

In our project we simulate a bushfire in a forest terrain which encompasses trees, plants, bushes, stones and lakes, which is the home of various animals. Our goal is to build trained dogs which could save endangered and vulnerable animals from the fire without itself hurting from the fire. We are focusing on building the forest terrain from ground up using Unity3D. We have designed trained bots using Unity Machine Learning agents that would learn to save the animals from the fire.

We have trained dog agents on different sets of RL based algorithms in the same environment and with a fixed training period to avoid bias. This will allow us to determine which sets of algorithms work better for particular scenarios. This model could be further ported into bots that could help emergency services in search and rescue operations!

Methodology

1. Unity Environment setup

Our initial step was to setup the Unity Environment with ml-agents installed. The steps for installation are as follows:

1. Install Unity 3D (v 2018.4)
2. Install Python (v 3.6.1)
3. Install *com.unity.ml-agents* package
4. Install the *ml-agents* python package

2. Forest Environment

To build the forest environment we used the unity standard assets. This package provides Cameras, Characters, Cross Platform Input, Effects, Environment, Particle Systems, Prototyping, Utility and Vehicles. We used this package to build trees, lakes, mountains, and grass in our forest terrain environment.

3. Agent, Animals and Fire animations

We used unity assets to design -

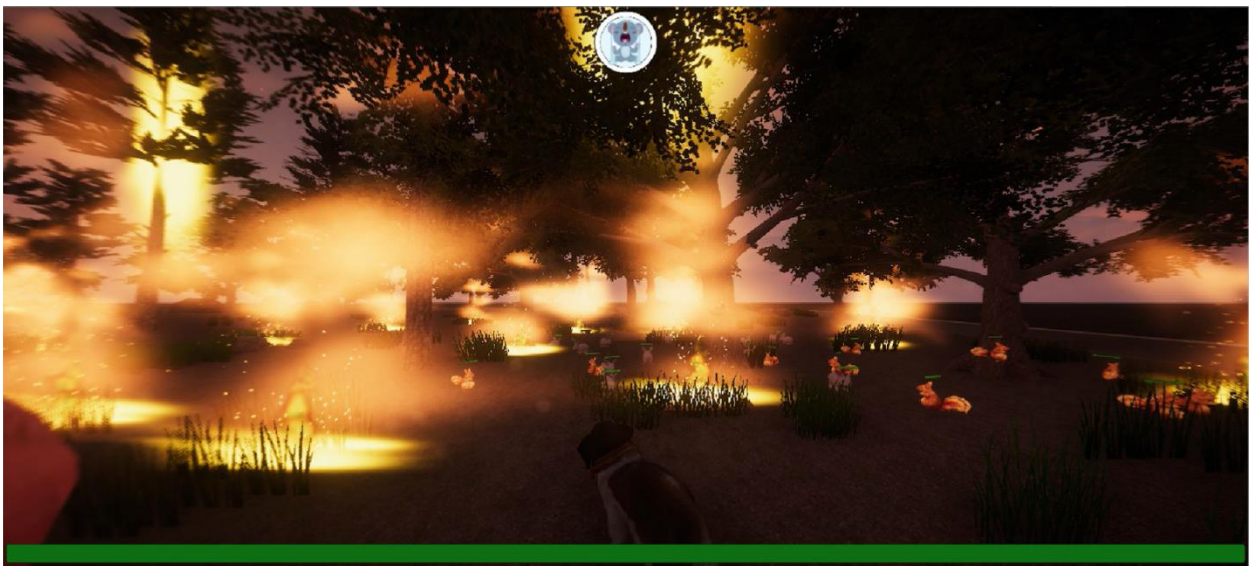
- A dog, which is the main agent of the game,
- Squirrels and rabbits which are the animals stuck in the forest fire and need to be saved by the dog,
- Fire, and
- A girl, who defines the safe zone in the game

Gameplay

Our dog agent is trained using various algorithms mentioned below. The agent is then placed in a forest fire simulated environment which consists of -

1. Burning bushes and trees,
2. A safe zone (consists of a girl)
3. Animals scattered all around the forest fire with a health bar

The aim of the dog is to save as many animals as possible from the forest fire and drop them one by one to the safe zone. Each animal has a health bar attached to it. The health of an animal gets depleted based on the proximity to the fire. Also, attached is a health bar that monitors the health of our dog, and a progress bar that indicates the number of animals saved. The dog needs to save the animal before it loses its health and dies. We also have various sound effects that replicate different emotions of the dog, the girl (safe zone) and the animals that need to be saved. The training and the testing environments also contain randomly spawned animals. To enhance the immersive nature of this simulation we have a dynamic camera which directs the entire strategy of our dog agent.



Training the Agent

Training of the dog agent involved different permutations and combinations of rewards and punishments for the dog to learn the environment from. At every step, the dog stored the following observations -

1. Distance between itself and the safe zone,
2. Distance between itself and the animals that need to be saved,
3. Current reward, and
4. Trigger update for fire

We tried using various algorithms and compared the efficiency of the algorithms. Below is an overview of all the algorithms used and the results obtained:

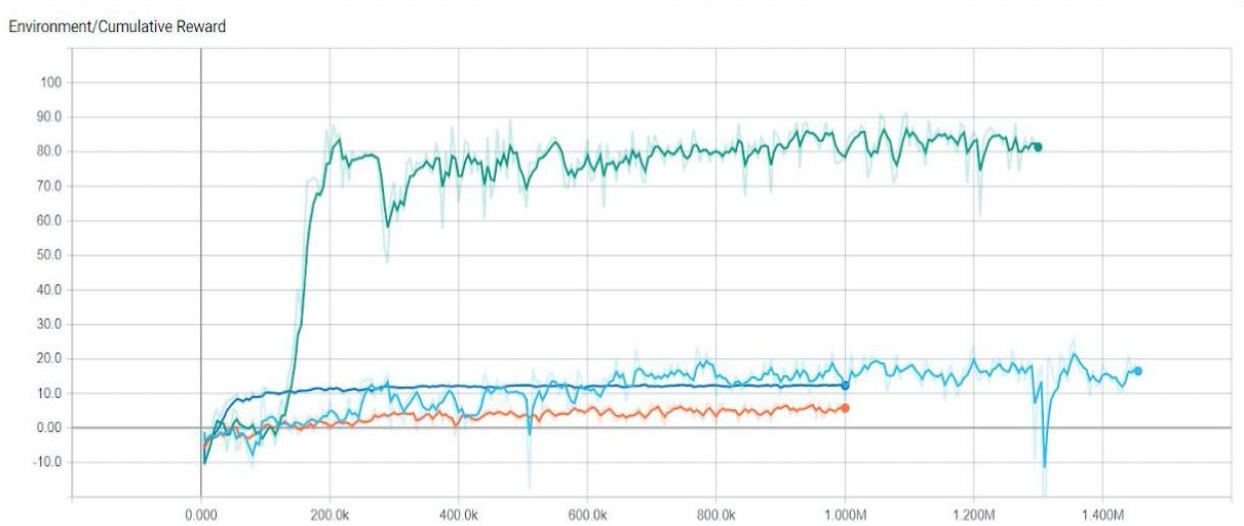
1. Proximal Policy Optimization

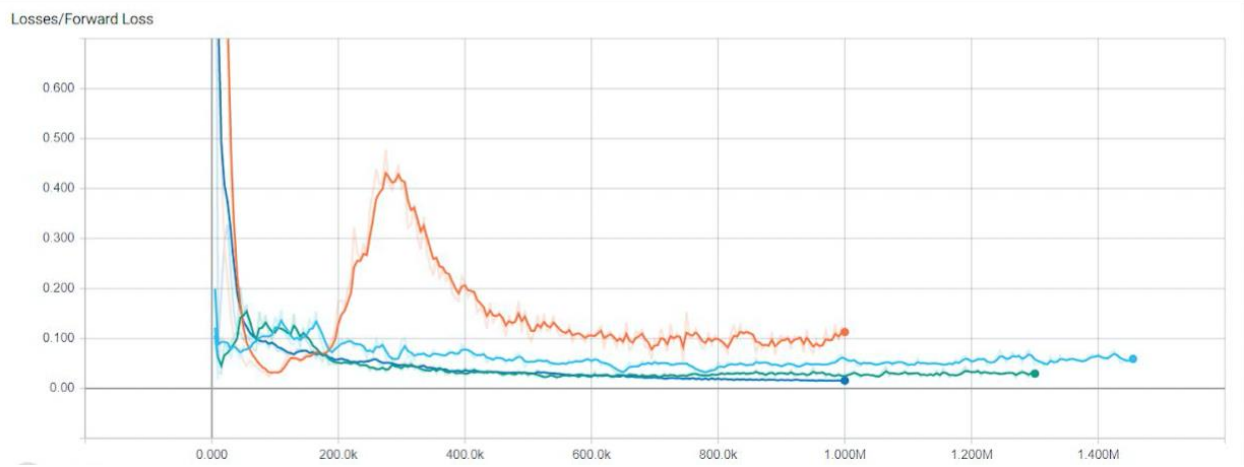
Our initial training configuration was defined to use the proximal policy optimization algorithm with no recurrent memory. These default configurations are provided by unity ml agents.

The proximal policy optimization (PPO) is a model-free, online, on-policy, policy gradient reinforcement learning method. This algorithm is a type of policy gradient training that alternates between sampling data through environmental interaction and optimizing a clipped surrogate objective function using stochastic gradient descent.

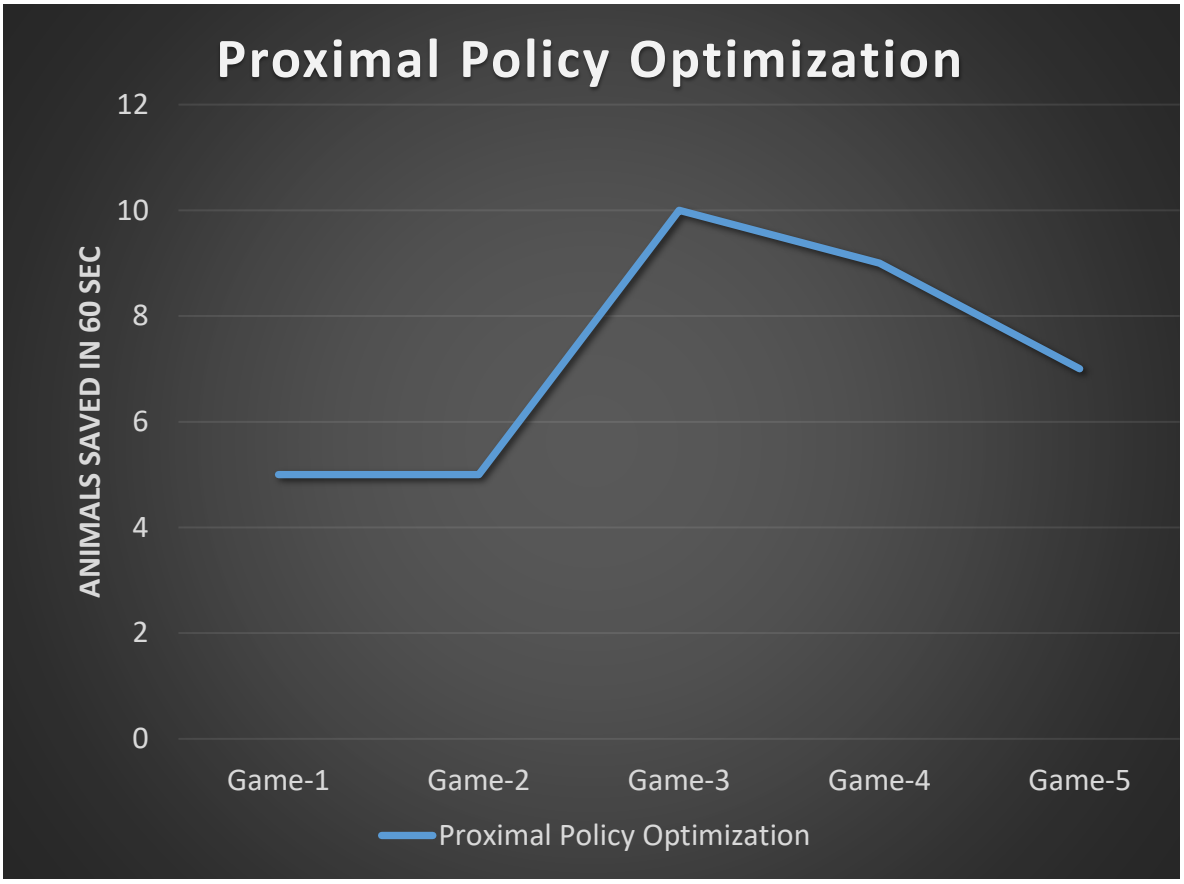
The training period ranged from 3 hours to 12 hours for a scaled down version of the environment. The training time reduced as we used more accurate values of rewards, punishments and the training configuration. Larger environments require extensive training periods and a significant amount of memory. As shown by the graphs below, as the training proceeds, the expected cumulative reward constantly rises, and the policy loss decreases. This signifies that the agent is learning a better policy, one that seeks positive reward. The training is stopped when the agent has gone through 1.4 million steps in the environment.

The training behavior is controlled by unity that interface with a python API. The API consists of a TensorFlow backend that enables the training in the environment.





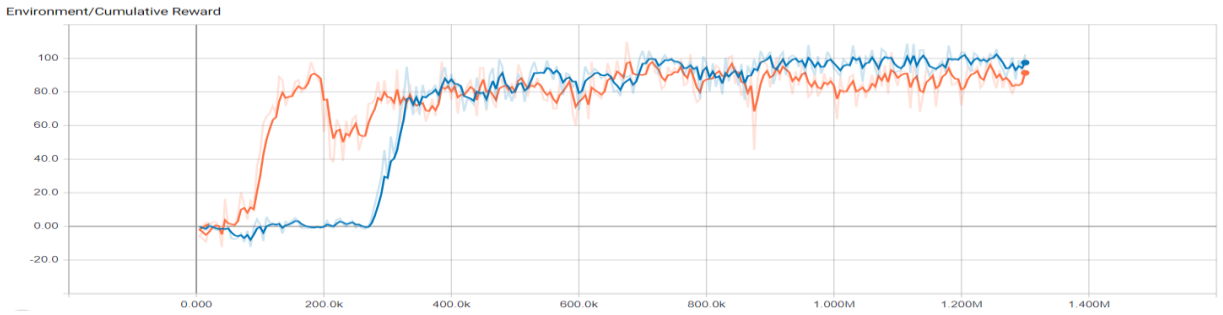
After a number of runs with the PPO algorithm, we found that it takes around 8.33 seconds to save one animal. The graph below shows the time taken to save an animal for 5 games.

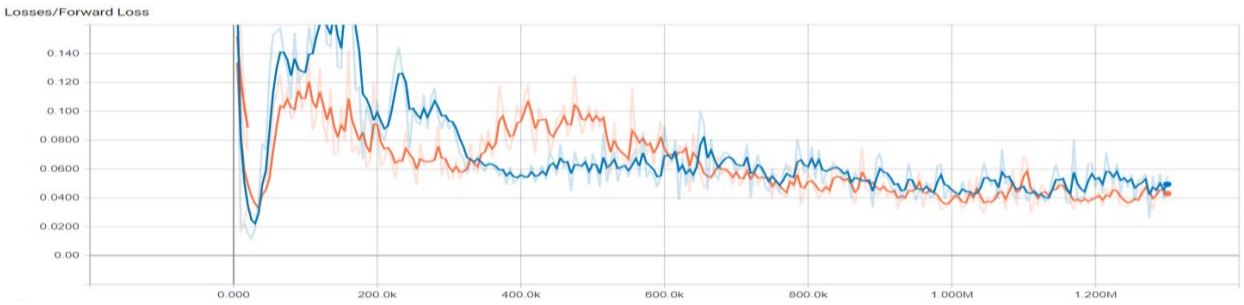


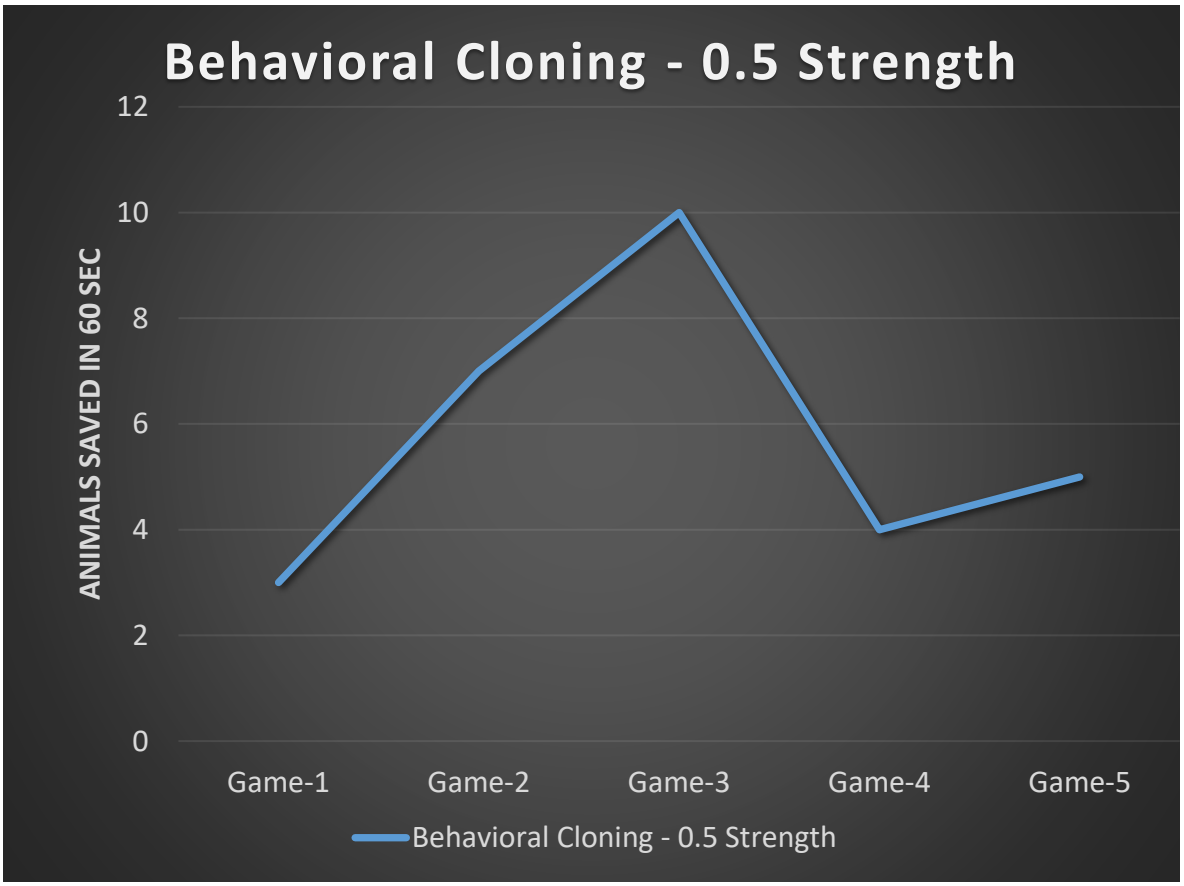
2. Behavioral Cloning

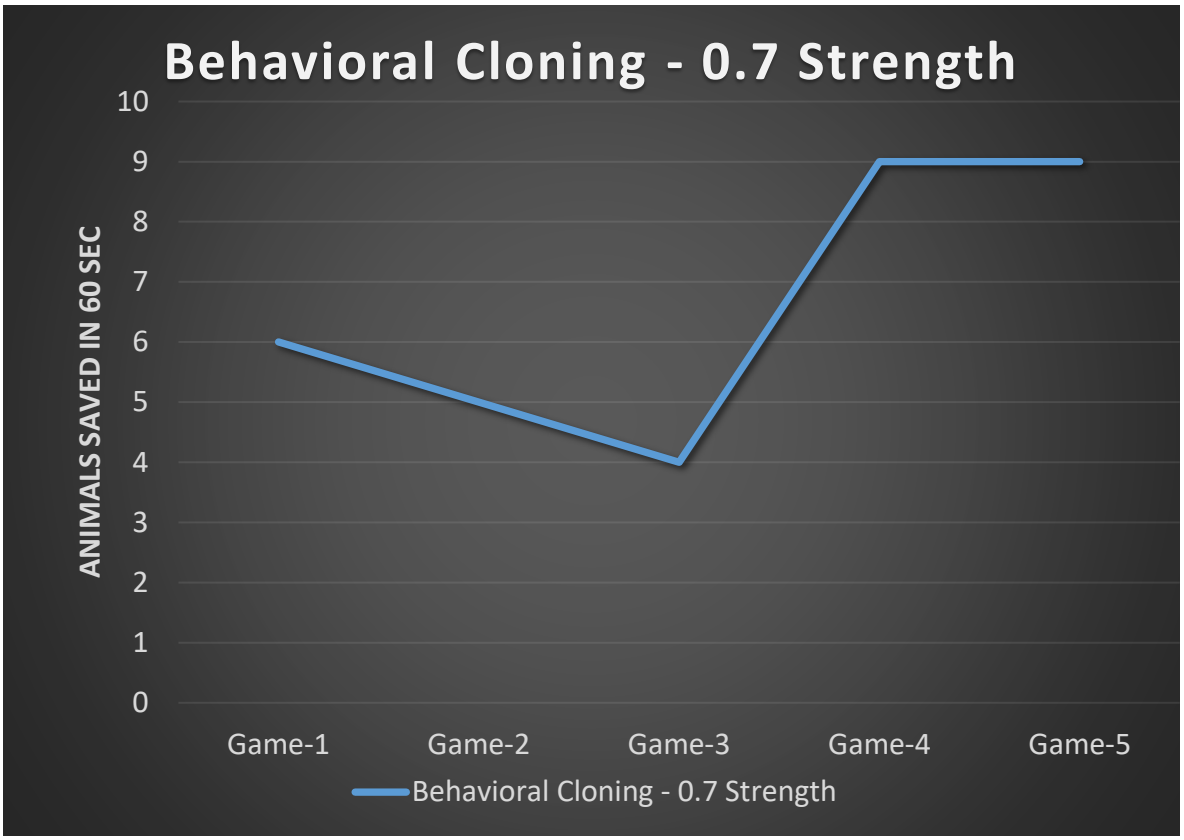
Behavioral cloning is a method by which human sub cognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training.

A demo of the game was created, and it's meta data was extracted. A combination of demo frames and reinforcement learning was used to train our dog agent. We tried a 0.7 strength and a 0.5 strength BC algorithm. Surprisingly, the 0.7 strength BC tried to overfit the data. The 0.5 strength BC algorithm did a better job training.


Environment/Cumulative Reward
100
80.0
60.0
40.0
20.0
0.00
-20.0
0.000
200.0k
400.0k
600.0k
800.0k
1.000M
1.200M
1.400M


Losses/Forward Loss
0.140
0.120
0.100
0.0800
0.0600
0.0400
0.0200
0.00
0.000
200.0k
400.0k
600.0k
800.0k
1.000M
1.200M

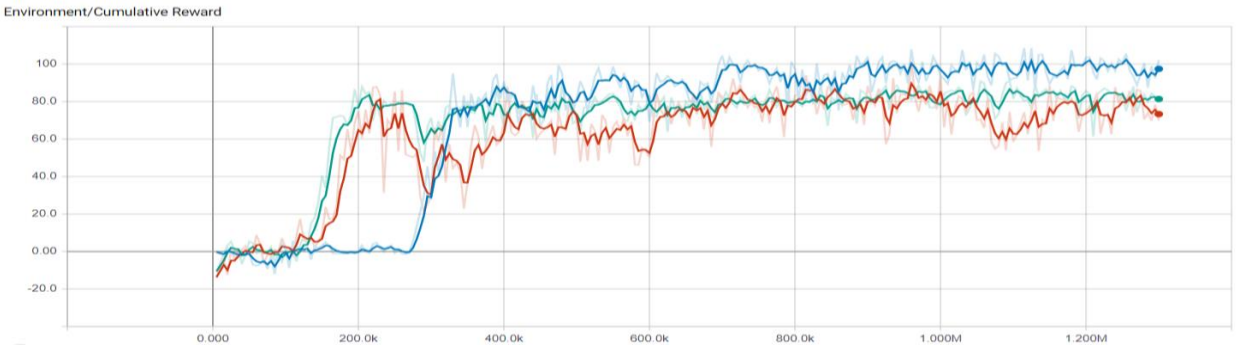

Behavioral Cloning - 0.5 Strength
ANIMALS SAVED IN 60 SEC
12
10
8
6
4
2
0
Game-1
Game-2
Game-3
Game-4
Game-5
Behavioral Cloning - 0.5 Strength

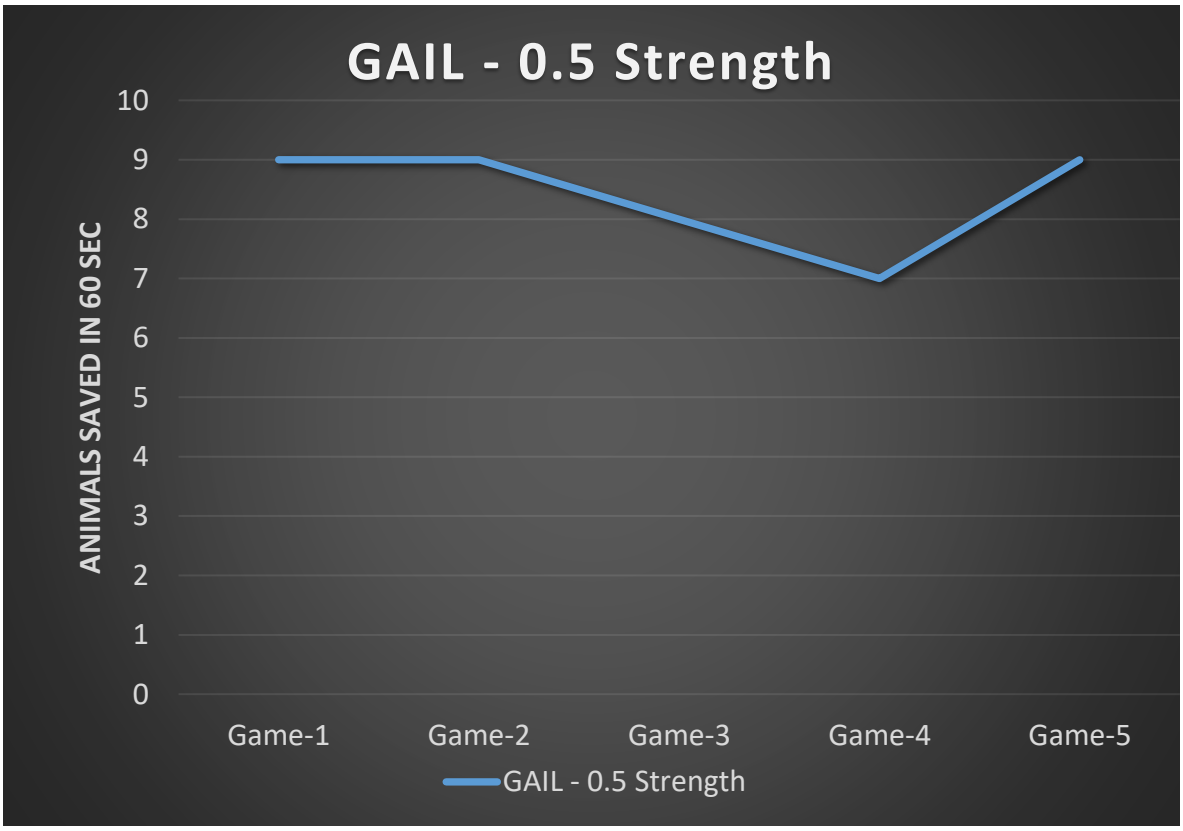


After a number of runs with the BC algorithm, we found that the 0.5 strength version takes around 10.34 seconds to save one animal, whereas the 0.7 strength algorithm takes 9.09 seconds to save an animal.

3. Generative Adversarial Imitation Learning (GAIL)

GAIL is a Model-free imitation Learning that imitates complex behaviors. It is an advanced version of a combination of Reinforcement Learning and Inverse Reinforcement Learning (IRL). The IRL learns a cost function that explains the expert's behavior but doesn't directly tell the learner how to act. Created a demo of the game and extracted its meta data. Used a combination of the demo frames and reinforcement learning to train our dog agent.



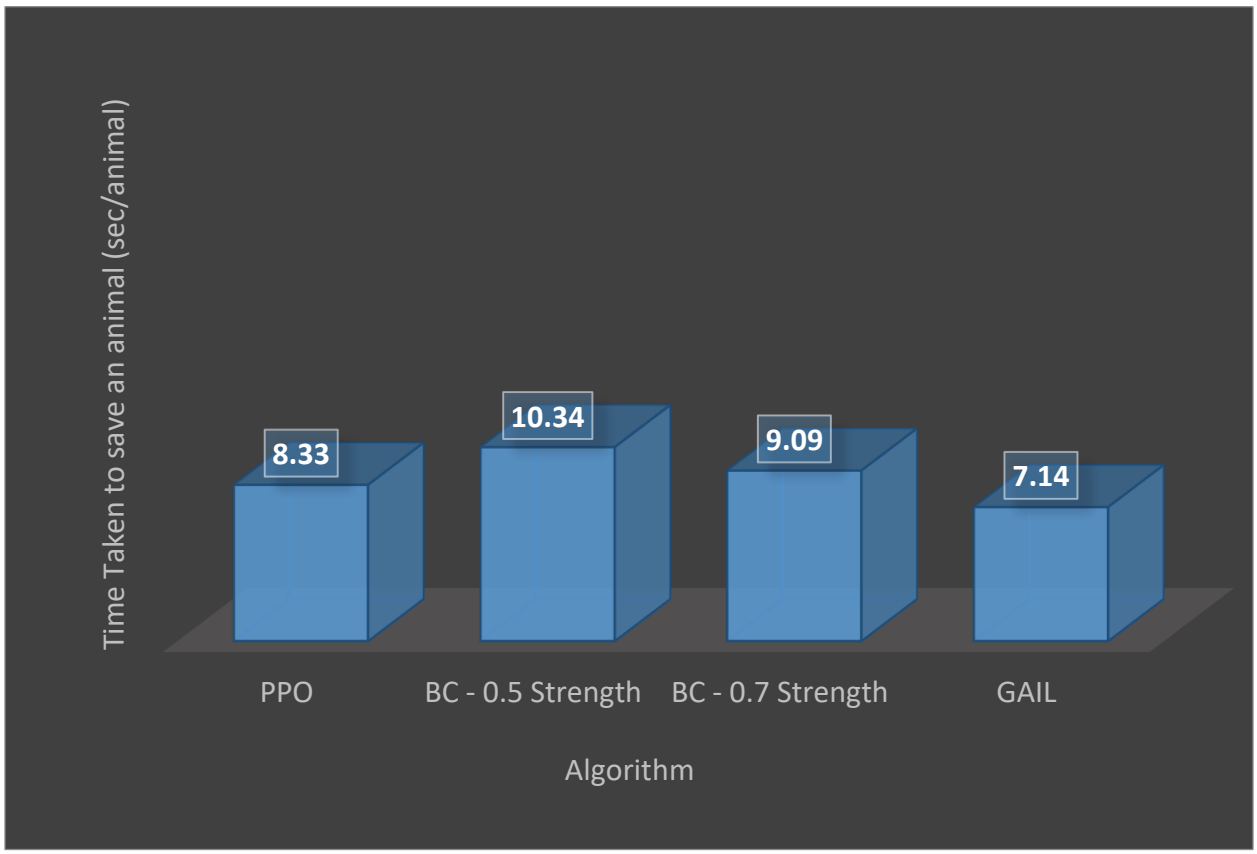


After a number of runs with the GAIL algorithm, we found that it takes around 7.14 seconds to save one animal.

Algorithm Comparison & Conclusion:

We tested PPO, BC (0.5 and 0.7 strength) and GAIL. The Behavioral Cloning algorithms (0.5 and 0.7 strength) work very well in the testing environment. However, they do not increase our agent's efficiency.

The addition of GAIL algorithm to PPO makes the dog agent extremely efficient. Below is a graph of the performance comparison between each of the algorithms and the time taken to save an animal in seconds.



Future Scope

This report outlines and summarizes different approaches and algorithms to train a dog agent to perform critical tasks such as search and rescue. We have implemented three different models (Proximal Policy Optimization, Behavioral Cloning and GAIL) for the main player agent. Some improvements that can be applied in the continuation of this project are:

- Implement the Soft Actor Critic algorithm to train the dog agent. SAC is defined for RL tasks involving continuous actions. Soft Actor Critic (SAC) is an algorithm that optimizes a stochastic policy in an off-policy way, forming a bridge between stochastic policy optimization and DDPG-style approaches
- Train a separate YOLO (You Only Look Once) object detection neural network
- Transfer the brains from the dog agent to real bots that can help fire and emergency services to rescue animals and humans in times of need
- Add a pretrained VGG neural network for precision.

References

[1]https://unity3d.com/get-unity/download

[2]https://github.com/Unity-Technologies/ml-agents

[3]https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Installation.md

[4]https://subscription.packtpub.com/book/game_development/9781789138139/1/ch01lvl1sec14/academy- agent-and-brain

[5]https://github.com/ayushbihani/Reinforcement-Learning-for-GridWorld

[6]https://github.com/ayushbihani/Dodgeball

[7]https://www.youtube.com/channel/UC1c0mDkk8R5sqhXO0mVJO3Q

[8]https://freesound.org/

[9]https://www.logikk.com/articles/machine-learning-in-game-development/

[10]https://pathmind.com/wiki/ai-vs-machine-learning-vs-deep-learning

[11] https://medium.com/@sanketgujar95/generative-adversarial-imitation-learning-266f45634e60

[12] https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Imitation-Learning.md

[13] https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md