This information can also be found in the wiki of the repository

Description

Artificial intelligence, or AI, is technology that enables computers and machines to simulate intelligence.

Unity Machine Learning is an open-source project that enables games and simulations to serve as environments for training intelligent agents. For this project I created AI agents in a Unity project using unity machine learning (ML-Agents). I wanted to test how quick or efficient the learning process of agents under different condition would be. That's why I chose the following evaluation proposal.

Evaluation Proposal: Create an AI using Unity machine learning that wants to collect a reward spawned randomly. Compare the episode length when the AI has bombs in its way that reward him negatively with the normal AI and the episode length for the AI when it just has obstacles in its way with the normal AI.

Justification:

Nowadays, AI is a very discussed topic. According to the article "The Case for Usable AI: What Industry Professionals Make of Academic AI in Video Games" most video games nowadays use at least one form of AI. Either for pathfinding, AI opponents or other specific behaviors, AI has become a go to technology for many developers. I believe it is a very useful tool that I should be familiar with as a game developer.

Research and tutorials:

After deciding to use unity machine learning, I started researching it and learning how to use it. I used this tutorial (https://www.youtube.com/watch?v=zPFU30tbyKs) to install and try to get a small demo running. This combined with the official documentation and another tutorial (https://www.youtube.com/watch?v=RANRz9oyzko) helped me understand how ML-Agents works. These helped me understand how to setup a script for my agents, what inference and heuristic are used for and how to create a neural network.

References:

Johannes Pfau, Jan David Smeddinck, and Rainer Malaka. 2020. The Case for Usable AI: What Industry Professionals Make of Academic AI in Video Games. In Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '20). Association for Computing Machinery, New York, NY, USA, 330–334. https://doi.org/10.1145/3383668.3419905

(https://unity.com/products/machine-learning-agents & https://github.com/Unity-Technologies/ml-agents) – Unity Machine Learning Documentation

Development Tracking:

I started out by trying to install ML-Agents which proved to be harder than it seemed. In total I believe I spent around 6-10 hours trying to install this. To use ML-agents one need to get the package in unity, have python of a certain version installed and then upgrade or install all the python packages that are needed and mentioned in the documentation to their proper version. Having run into problems I had to repeat this process a few times. After that I could finally spend time experimenting and deciding exactly what I wanted to test using ML-Agents. My first idea was to try to create a walking AI. Unfortunately, that didn't go far since I was unable to get a 3D agent to walk. The agent had 4 legs each with a joint, but he most it could do was tremble and move a couple of millimeters. (The agent was being rewarded for travelling in a certain direction.) I went back to a much simple idea which was an agent trying to get to a location where it would collect a fruit getting rewarded. To make this idea more complicated I thought about using bombs which would be the same as the fruits but giving a negative reward. After this the rest of the progress can be tracked using the git repository, as I had finally had a set idea.

Testing setting:

Setup:

For the setup I created 3 Unity scenes, each with prefabs of the environment I am testing. Each environment contains, the platform, the walls preventing the avatar from falling off, the avatar which also holds the brain and the scripts and the fruit (the reward they need to collect). Additionally they could have bombs that would give the avatar negative rewards or obstacles on the platform that would jus t collide with the avatar.

Actions:

Each avatar can move on the x and z axis with a preset speed. (These actions can also be testing using the heuristic setting in the behavior parameters.)

Observations:

The AI model is given the following information: its position, the position of the fruit (reward) and, if it is the case, the location of the bomb. In total getting a maximum of 9 observations.

Rewards:

When an AI touches a fruit (reward) it gets 10 points. When it touches a bomb or any of the side walls it gets –10 points. For my setup any of these actions result in the episode ending. To better visually track my AI when it touches a fruit the floor turns green, for a bomb it turns red and for a wall it turns white.

On episode begin:

When an episode begins the avatar gets moved back to the Vector.Zero local position of the environment and the fruit and bomb (if it is the case) get moved around.

Optimizations:

To prevent the AI from just memorizing that one specific position is always the way to go or to avoid, the fruit and the bomb get moved around. In order to have a bit more control on where anything spawns, I create grid at the start of the program. I am also preventing having a fruit and a bomb be spawned in the same spot.

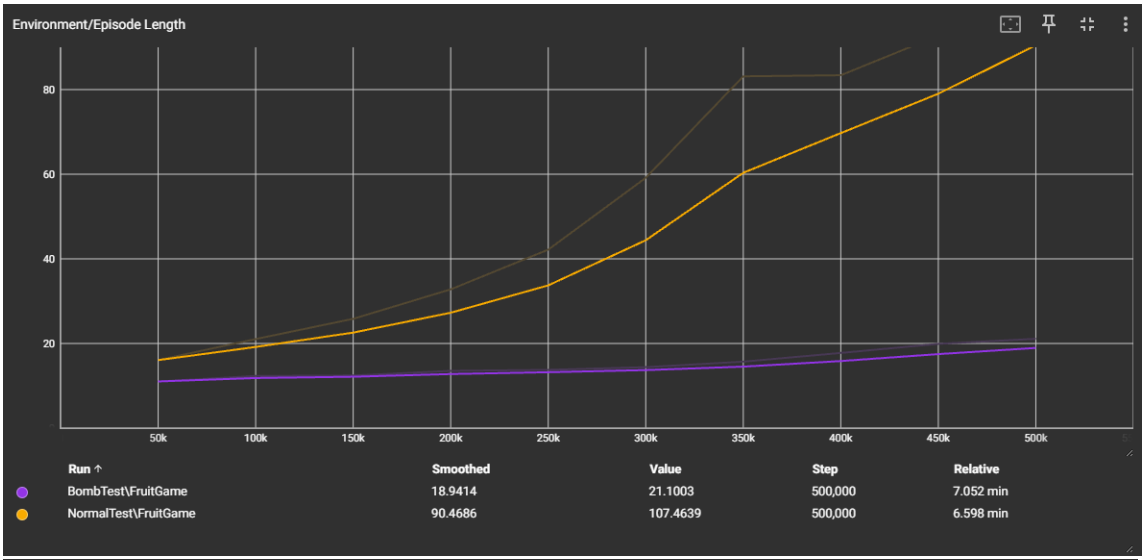
Additional attempts or ideas

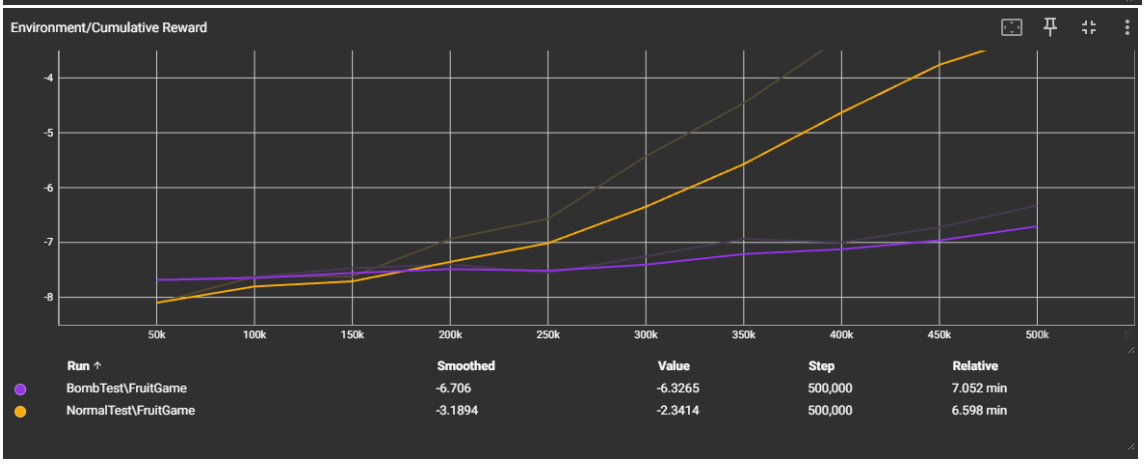
To improve on my AI tried different ideas. One was to play around with the size of the environment. I ended up keeping the environment quite small to make the agents life easier. Another idea was giving the AI an extra observation that would tell him the distance between the fruit and himself. I didn't notice much of an improvement in this case so I decided to scrap the idea to keep my testing less complex.

Charts

A quick explanation for the charts: The horizontal line represents the number of episodes. Additional to the episode length chart I also added a cumulative reward chart to show the improvement of the AI as well.

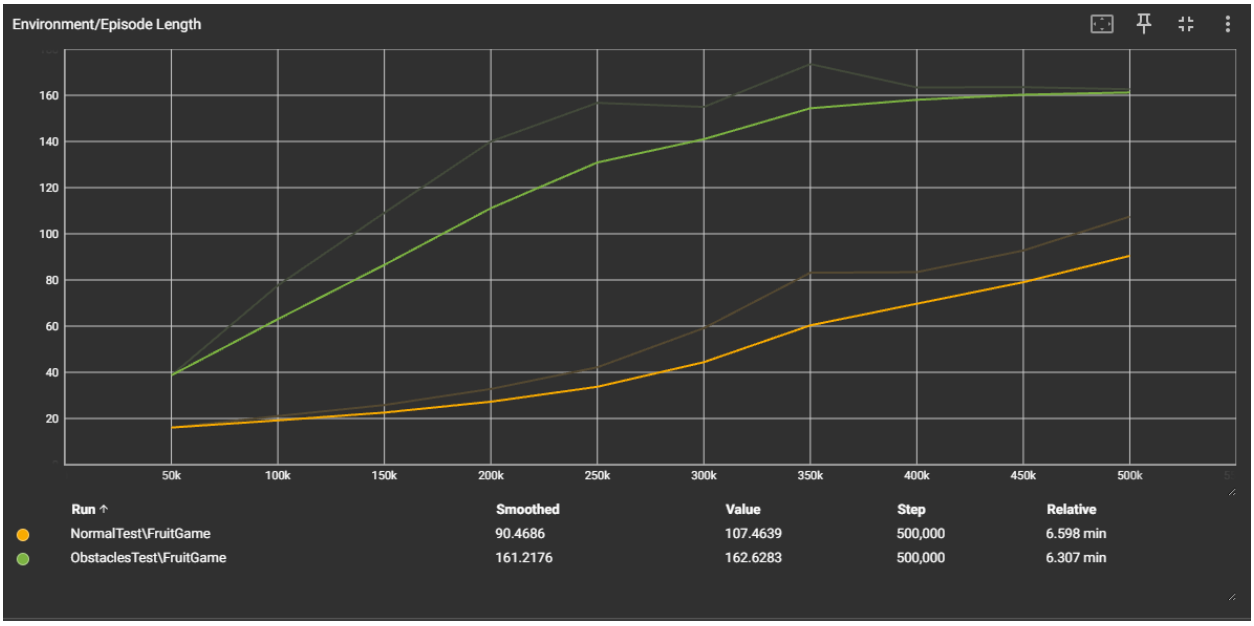
Comparing the normal AI with the bombs AI

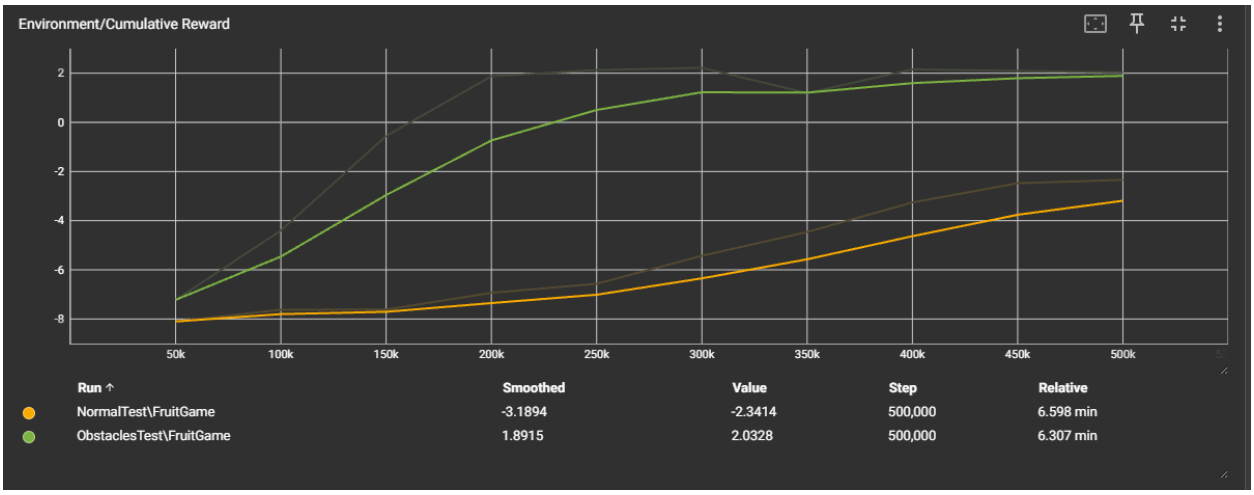




A steady increase of the episode length can be seen in the normal AI, while bomb AI has a much less drastic increase. Combined with the fact that the bomb AI clearly had a harder tiem learning it is still quite a big diffrence. The bomb AI should have learned to better avoid the bombs and outter walls.

Comparing the normal AI with the obstacles AI





Completely opposite to the bomb AI the obstacle algorithms seems to have improved the amount of rewards the agent would get. The episode length follows a more drastic increase.

Analysis

Adding bombs to the AI drastically decreased its learning efficiency and lowered the episode length. I believe this is means that the agent would quickly just kill himself and not have time to learn more about

obtaining a reward. The obstacle agent has a high increase in episode length, leading me to believe that the AI would learn the path (the positions of the obstacles) and then focus on finding the fruit.

Conclusion

It is more efficient to teach the AI one simple task first. Adding obstacles either drastically increased or decreased the episode length, which are two extremes that aren't necessarily good for having an AI learn something. One doesn't want an AI to loiter and learn that doing nothing is good, nor to have the AI instantly gain negative points thus ending the episode.