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 just 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 1 point. When it touches a bomb or any of the side walls it gets –1 point. 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 and for a bomb or an outer wall it turns red.

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. Additionally, I am also preventing having a fruit and a bomb be spawned in the same spot.

Additional attempts or ideas

First Attempt:

To improve on my AI, I 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. After all my attempts the AI was turning out quite bad having little knowledge in achieving its goal.

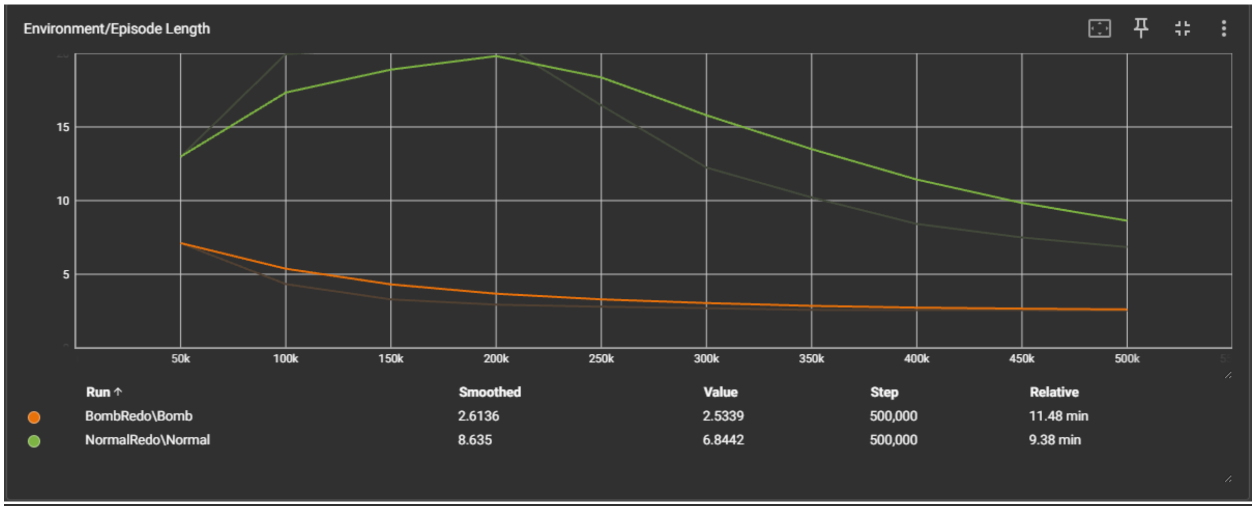
Second Attempt:

I decided to drastically improve my AI, so I started out from the begging to make sure I properly understand what helps its learning process and what doesn't. So, I recreated the move to goal AI (an ai that just need s to reach a preset target). This also helped me in making sure that the AI was learning and that there wasn't something wrong with my setup. From this I learned that the AI shouldn't have too high of a speed, so that it can be better visualized when learning and that the smaller that platform the quicker it learns as it has less possible paths it can take. I recreated my previous tests with fewer possibilities of places where items can spawn and with a way smaller platform. The difference was immediately obvious as the AI finally looked like it was learning.

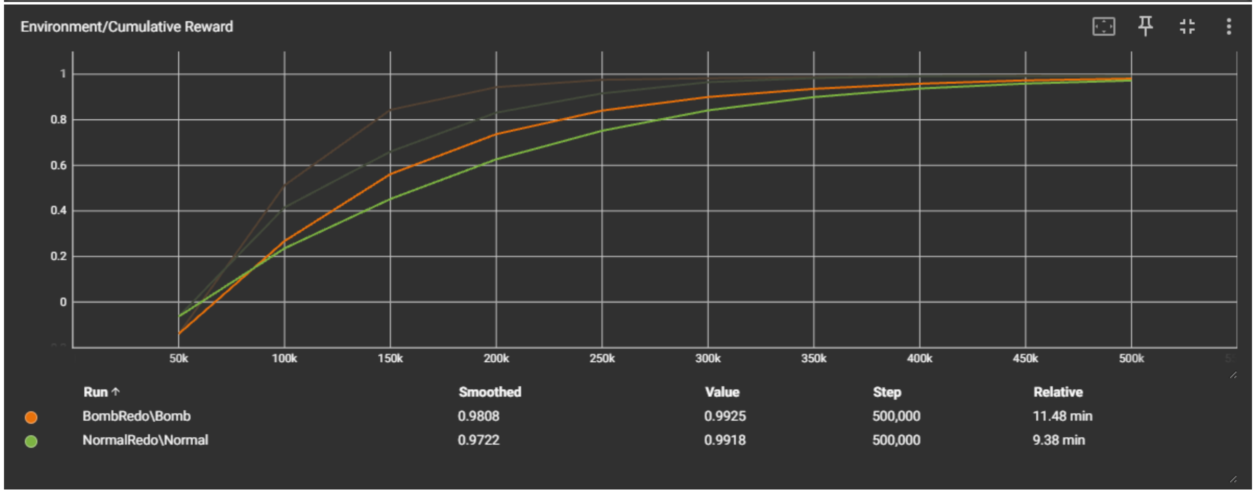
Charts

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



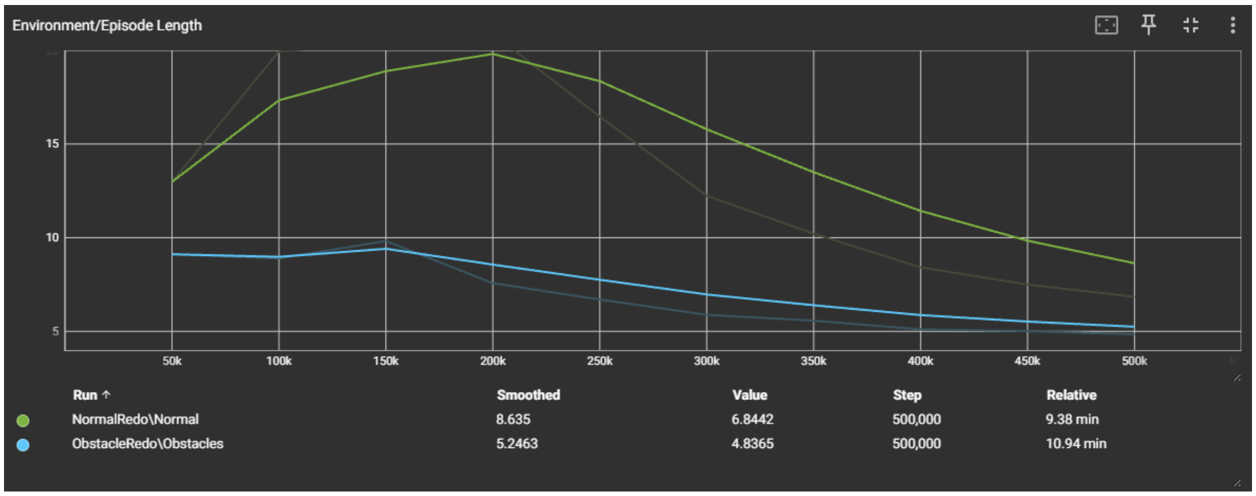
Run ↑	Smoothed	Value	Step	Relative
🟠 BombRedo\Bomb	2.6136	2.5339	500,000	11.48 min
🟢 NormalRedo\Normal	8.635	6.8442	500,000	9.38 min



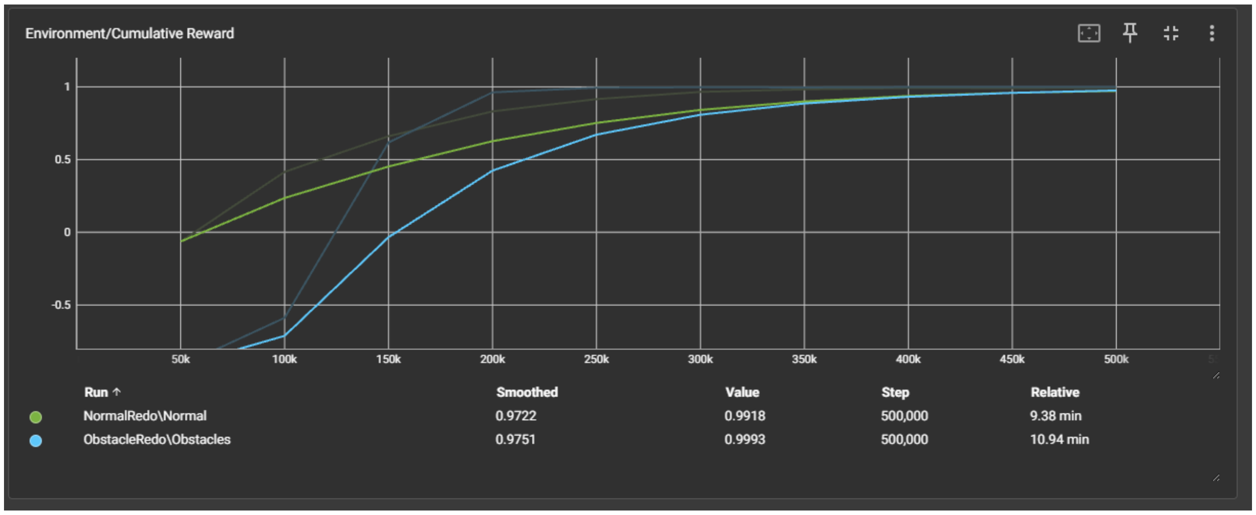
Run ↑	Smoothed	Value	Step	Relative
🟠 BombRedo\Bomb	0.9808	0.9925	500,000	11.48 min
🟢 NormalRedo\Normal	0.9722	0.9918	500,000	9.38 min

As expected at first the bomb AI has much shorter episodes since it has way more possibilities of dying, However, the bomb AI is quicker to reach the optimal itme length per episode (the case in which the agent goes straight to the reward). This means that the presence of the bomb somehow made the AI learn faster tha n without the bomb. To prove my point that the bomb AI is learning faster I provided a second graph with the cummulativce reward, where it can be seen that the bomb AI reacheds max rewards faster. If left to train more the normal AI should also reach the same episode length as the bomb AI.

Comparing the normal AI with the obstacles AI



	Run ↑	Smoothed	Value	Step	Relative
●	NormalRedo\Normal	8.635	6.8442	500,000	9.38 min
●	ObstacleRedo\Obstacles	5.2463	4.8365	500,000	10.94 min



	Run ↑	Smoothed	Value	Step	Relative
●	NormalRedo\Normal	0.9722	0.9918	500,000	9.38 min
●	ObstacleRedo\Obstacles	0.9751	0.9993	500,000	10.94 min

The obstacle AI has a way shorter episode as its longest episode because it sort of has less ways to kill itself in longer ways. The way the obstacles are placed actually ends helping the agent by guiding it away from the outer walls. After passing the obstacles the agent just needs to move up or down to find the reward.

Analysis

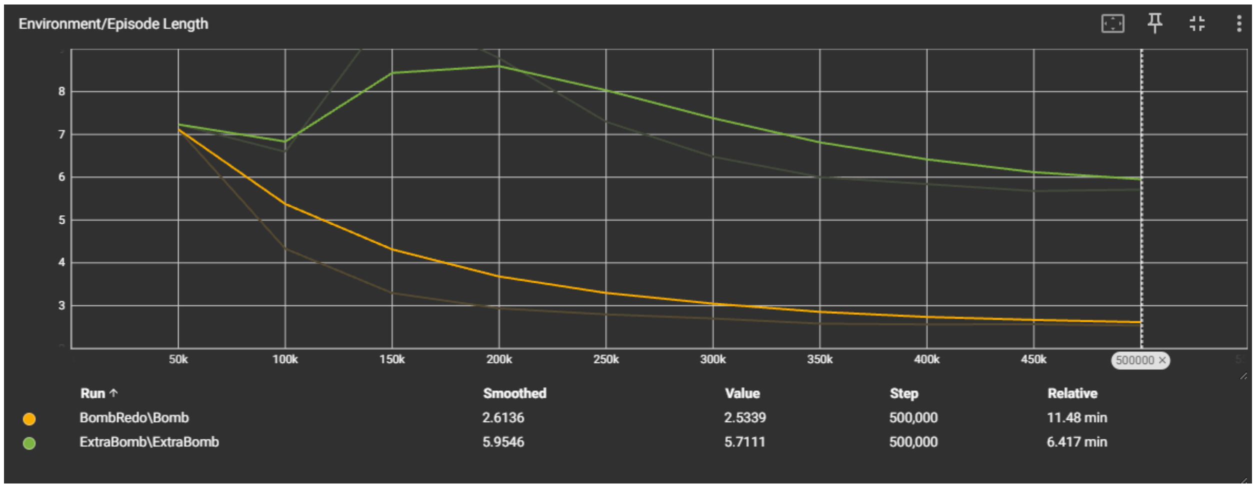
The highest point in the normal graph represents the moment when the finishes trying the longet paths to the reward. It starts out with less time per episode when it is activaly killing its self in the outside walls and continues to find out where else it can walk. After it learn where the outer walls are it starts going t the reward more often.

The most intersting result of this analsys is that the bomb ai was by far the most efficient in learning. This means that the more guided the AI is with positive or negative rewards the quicker it learns.

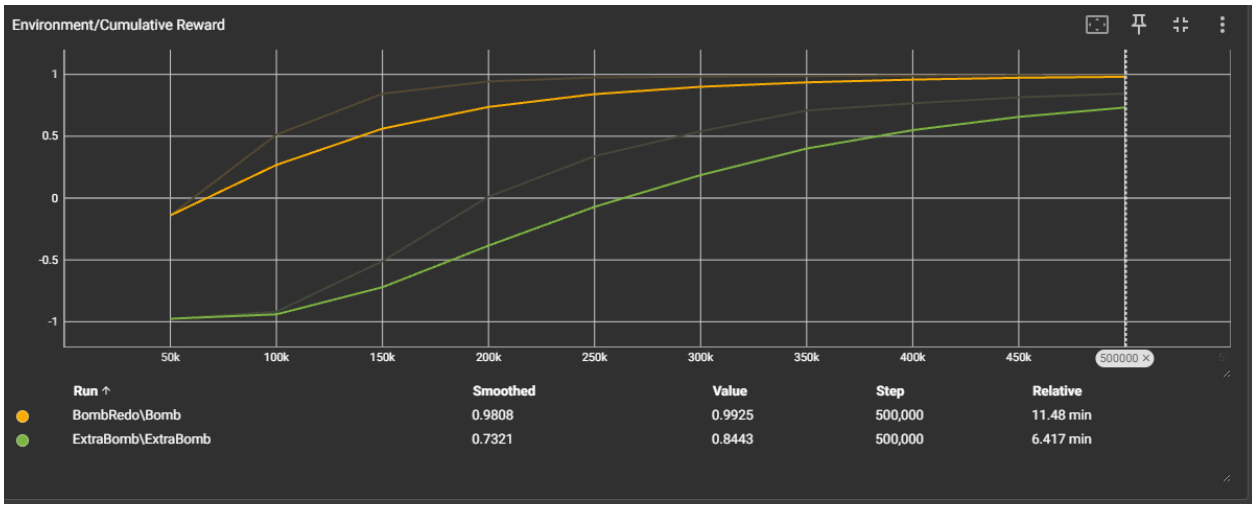
New tests

After conducting the previews tests I went to obtain some feedback on them. I was advised to conduct more tests in which I see if more bombs help or not the AI and in which I try out an obstacle made to obstruct the agent.

Two Bombs vs one bomb



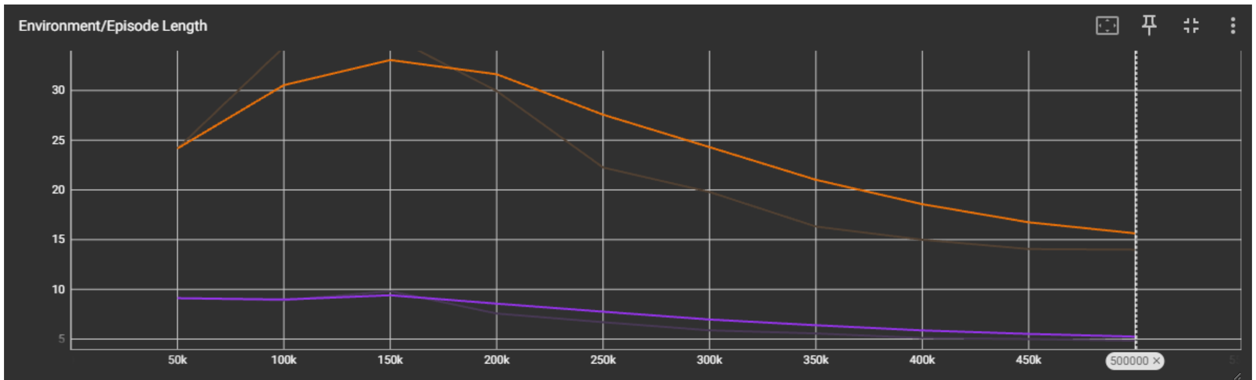
Run ↑	Smoothed	Value	Step	Relative
● BombRedo\Bomb	2.6136	2.5339	500,000	11.48 min
● ExtraBomb\ExtraBomb	5.9546	5.7111	500,000	6.417 min



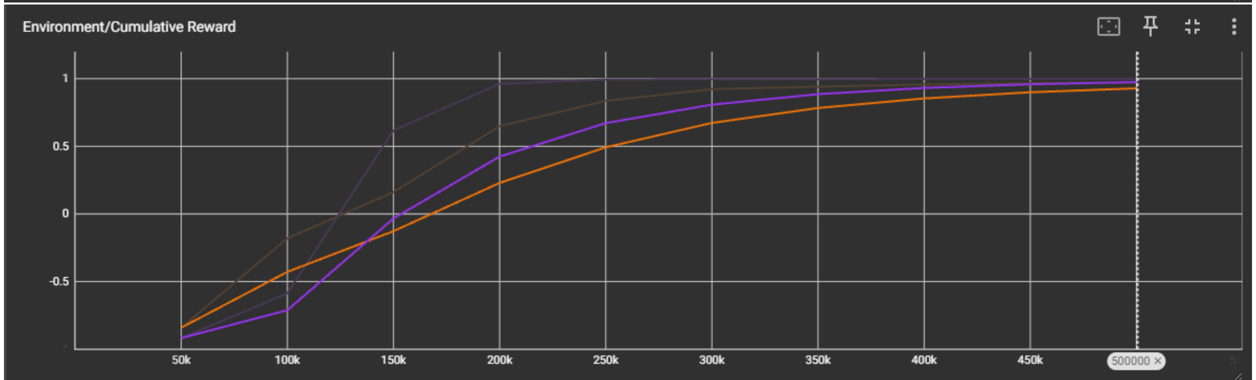
Run ↑	Smoothed	Value	Step	Relative
● BombRedo\Bomb	0.9808	0.9925	500,000	11.48 min
● ExtraBomb\ExtraBomb	0.7321	0.8443	500,000	6.417 min

Adding an extra bomb ma de the episode length increases a bit at first. Somehow the AI became more takes some extra time in finding the reward. The quick drop at the beginning is probably the AI realizing that going to quickly in a direction might kill him. After wards he seems to try to find new paths with extra care.

Obstructive obstacle vs helping obstacles

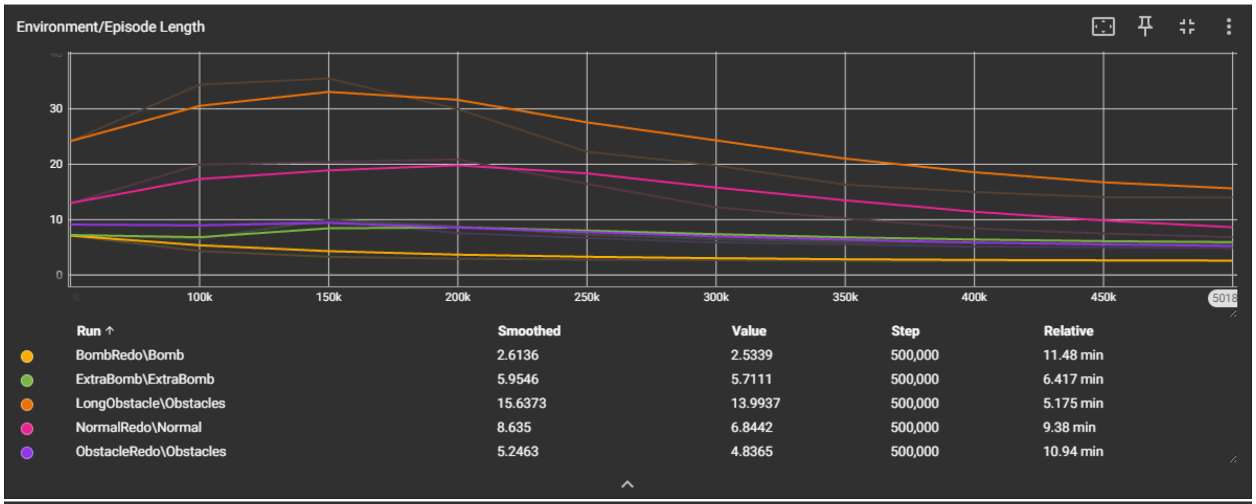


	Run ↑	Smoothed	Value	Step	Relative
●	LongObstacle\Obstacles	15.6373	13.9937	500,000	5.175 min
●	ObstacleRedo\Obstacles	5.2463	4.8365	500,000	10.94 min

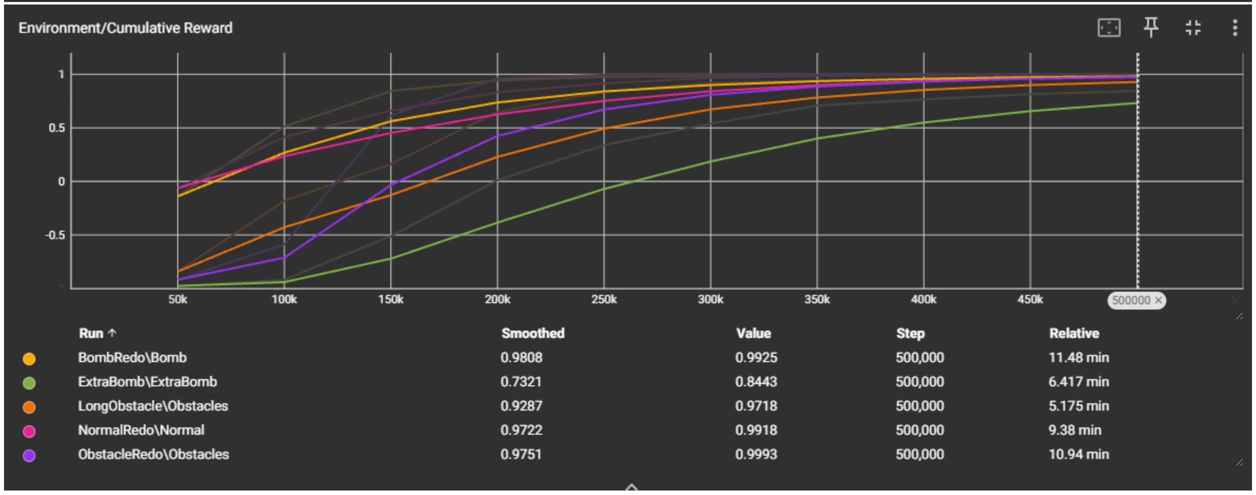


	Run ↑	Smoothed	Value	Step	Relative
●	LongObstacle\Obstacles	0.9287	0.9718	500,000	5.175 min
●	ObstacleRedo\Obstacles	0.9751	0.9993	500,000	10.94 min

As expected adding a more obstructive obstacle made the AI have a harder time in reaching the goal in a short period of time.



	Run ↑	Smoothed	Value	Step	Relative
●	BombRedo\Bomb	2.6136	2.5339	500,000	11.48 min
●	ExtraBomb\ExtraBomb	5.9546	5.7111	500,000	6.417 min
●	LongObstacle\Obstacles	15.6373	13.9937	500,000	5.175 min
●	NormalRedo\Normal	8.635	6.8442	500,000	9.38 min
●	ObstacleRedo\Obstacles	5.2463	4.8365	500,000	10.94 min



	Run ↑	Smoothed	Value	Step	Relative
●	BombRedo\Bomb	0.9808	0.9925	500,000	11.48 min
●	ExtraBomb\ExtraBomb	0.7321	0.8443	500,000	6.417 min
●	LongObstacle\Obstacles	0.9287	0.9718	500,000	5.175 min
●	NormalRedo\Normal	0.9722	0.9918	500,000	9.38 min
●	ObstacleRedo\Obstacles	0.9751	0.9993	500,000	10.94 min

After conducting all of these training sessions with different variables my conclusion is that using tools like bombs or walls can either be your friend or enemy in making your AI more efficient. One bomb really helps the AI while 2 bombs already shows an increase in the episode length. Obstacles can help your AI reach your goal or greatly increase its time. According to the last charts in my case all my cases were better than the normal expect for the long obstacle.

Conclusion

The learning speed of an AI is improved based on your guidance. Additionally, adding obstacles can also benefit your AI as it might lower the possible bad paths an AI can take.