CMPT766 - Computer Animation

Programming Assignment 4:

Deep Reinforcement Learning (10 points)

Due date: **23:59 Thursday Dec 5, 2024**

In this assignment, you will be using Unity's ML-Agent package to train an agent with deep reinforcement learning. The coding environment will be in Unity, but you will also need to install Python for training.

Unity Setup

To begin, download and extract **CMPT766-Assignment4.zip**. Open the extracted folder **CMPT766-Assignment4** as a Unity project in Unity Hub and select **2022.3.46f1** as the editor version. Later versions should also work, but if you run into issues then use the listed version. It should take a while to open the project for the first time. Once the project is opened, open the scene in **Assets -> Scenes -> Assignment4**.

Python Setup

First, download anaconda or miniconda. Once you've installed conda, open the terminal and create a virtual environment via the command:

```
conda create -n assignment4 python=3.10.12
```

This sets up a virtual environment named **assignment4** with python 3.10.12. Once the environment is created, activate it by typing the following command in the terminal:

```
conda activate assignment4
```

The terminal should show (assignment4) after you typed in the command indicating the virtual environment has been activated.

Then from the same terminal where you've activated the virtual environment, install PyTorch by first going here to find the install command for your OS, then type that into your terminal. You'll also need to install the mlagents python package:

```
pip install mlagents protobuf onnx
```

Project Script files

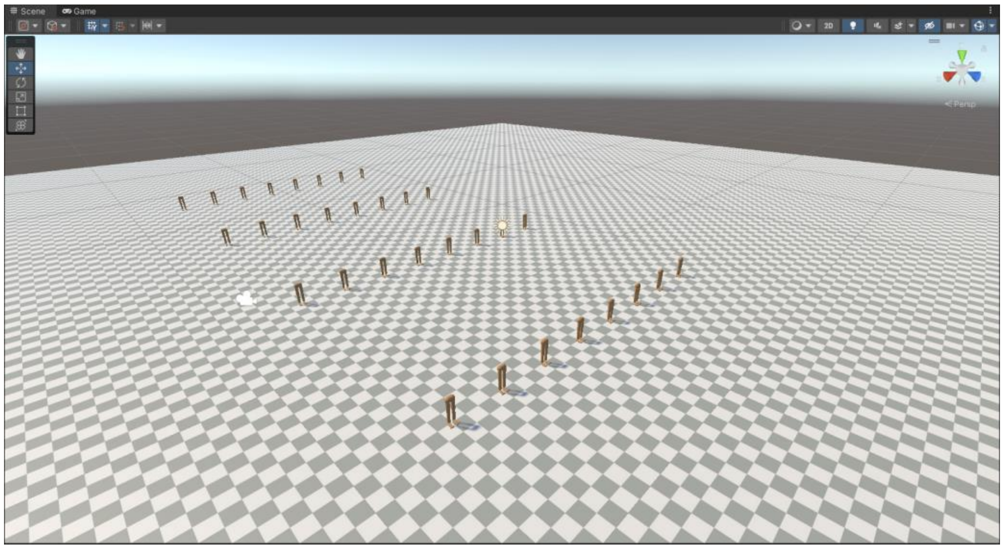
All the C# script files are contained in **Assets/Scripts** folder. There is only one file you'll need to modify:

1) **MarathonManAgent.cs** The only file you need to modify.

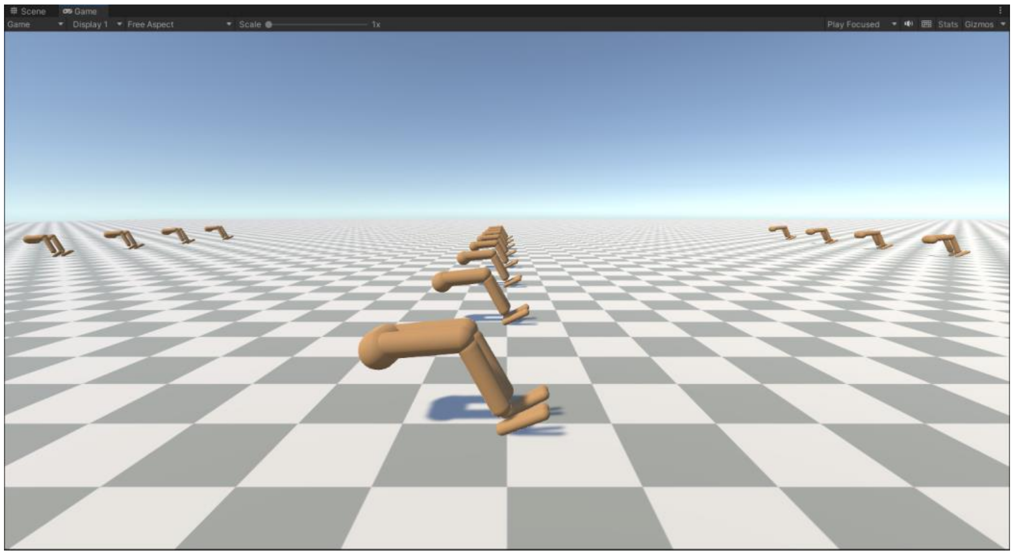
   This script dictates how the agent should interact with the environment. Including getting the observation state, assigning actions, and calculating rewards.

Introduction to the project

When you first open the project, you'll be greeted with 32 individual torsos standing on the checkerboard plane. Each of the torsos will be our agents.



When you press the play button, you'll see them start falling onto the ground. And once their body touches the ground, they'll reset back to their initial states.



Our goal is to train the agents so they can move in a forward direction without falling onto the ground.

Note that using 32 agents achieves the fastest training on TA's machine and it may not be the best number to use on your machine. Feel free to experiment with it if you want to improve your training speed.

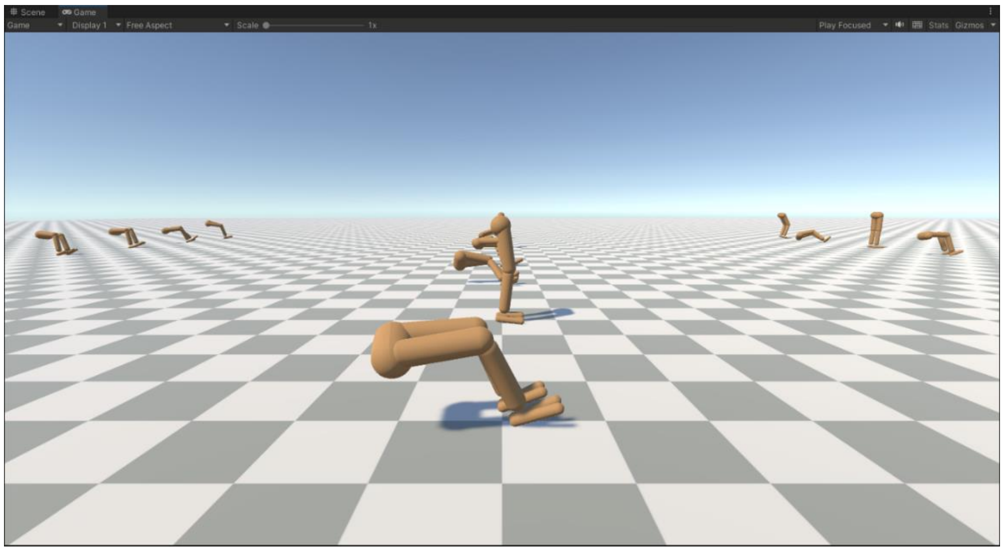
Training the Agent

To begin training, open the terminal and activate your virtual environment. Then, navigate to the directory where you extracted your Unity project and type in the following command:

```
mlagents-learn CMPT766-Assignment4/Assets/Config/MarathonMan.yaml --run-id=MarathonMan --force
```

This starts a server which allows Unity to interact with ML-agent's python training modules. When a message "Start training by pressing the Play button in the Unity Editor" is displayed on the terminal, you can press the Play button in Unity to begin training in the Editor.

Once it starts training, you should see the characters start moving and they will fumble onto the ground at the beginning.

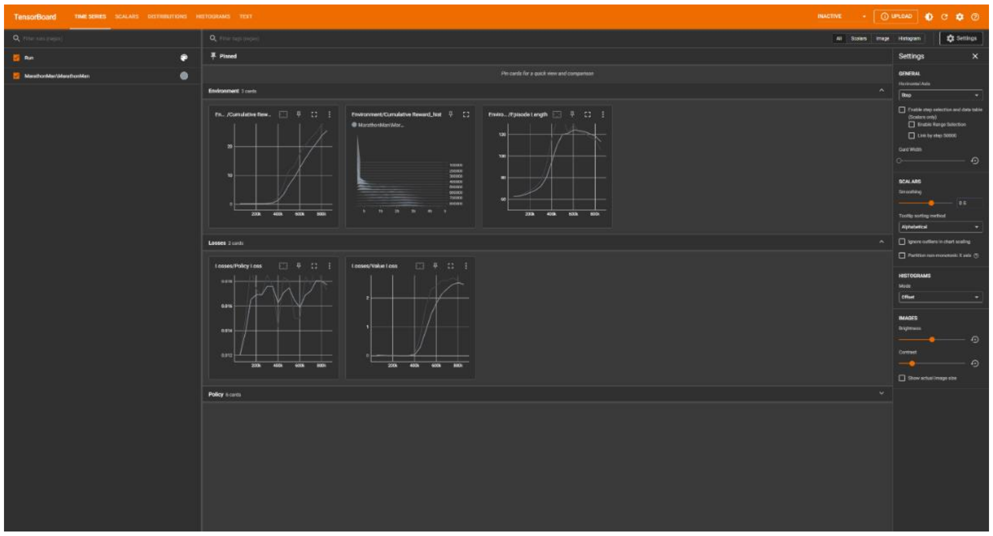


The agents will be trained for 10000000 steps, and it took roughly 1.5 hours on TA's machine, which has an Intel Xeon 6248R CPU and an NVIDIA GeForce Quadro RTX 6000 graphics card. If you have installed PyTorch with CUDA and have a CUDA device, Unity will automatically select your GPU for training, which is usually faster than training on CPU.

To view the plots of your cumulative rewards and training loss, open the terminal and navigate to where your project is extracted, then type in the following command:

```
tensorboard --logdir results
```

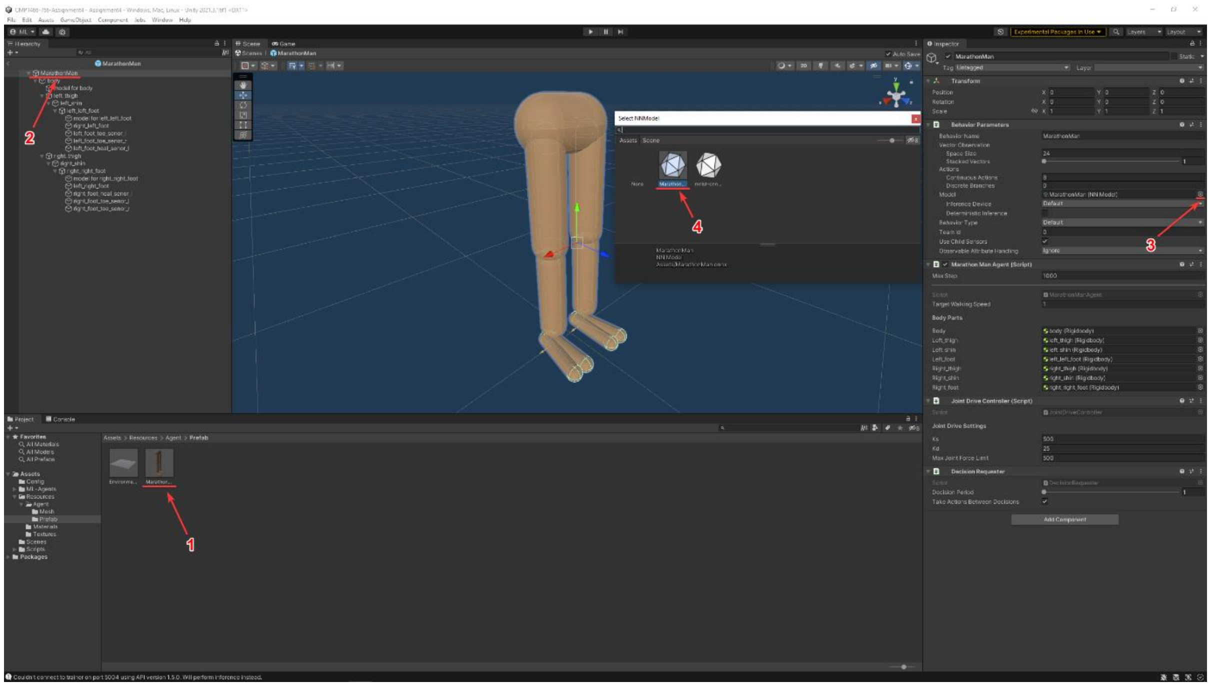
Open tensorboard at http://localhost:6006/, you'll then be able to see the plots.



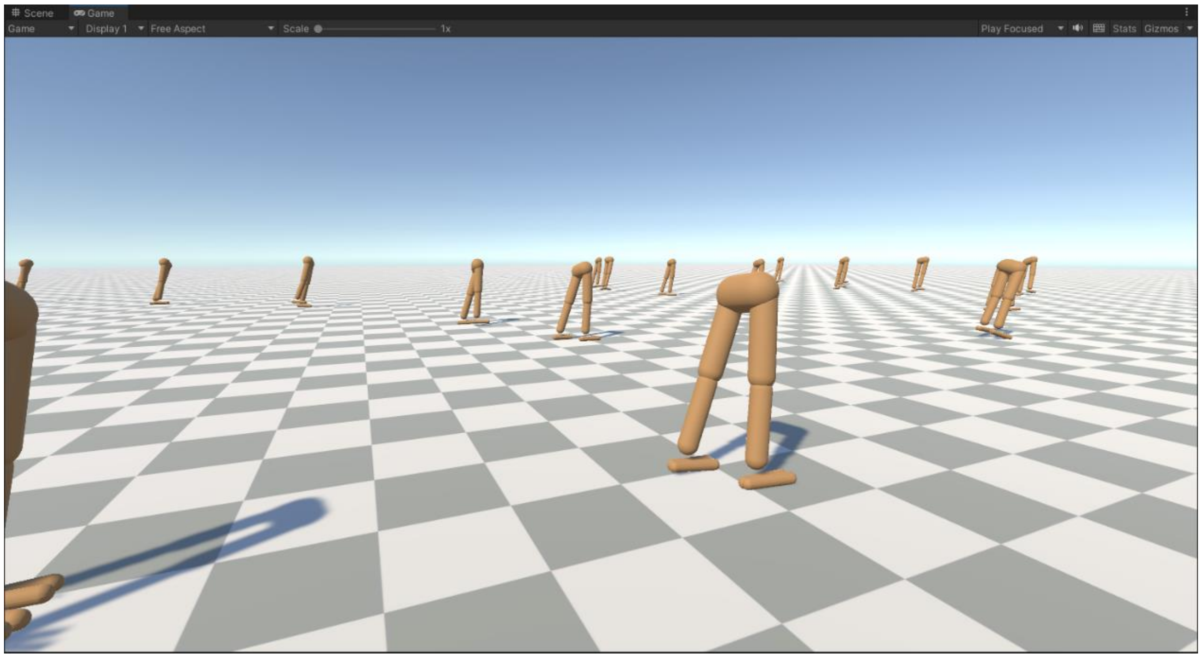
Once the training is finished, a **results/** directory will be created at where you extracted your project folder, and the trained model should be saved to **results/MarathonMan/MarathonMan.onnx**. Make sure to rename it to **MarathonManStandard.onnx** or **MarathonManVariation.onnx** to ensure you don't override and lose an existing trained model while testing the different reward functions.

Playing the Agent

Once you have a trained model, drag the trained **.onnx** file into the Assets folder. Navigate to **Assets/Resources/Agent/Prefab** in the Unity explorer, open the MarathonMan prefab, and select the MarathonMan GameObject in the Hierarchy window, then click into the model attribute in the inspector tab and select the **.onnx** file you want to test.



Now you can press the play button to see the trained controller in action.



What to Code?

Recall that given an observation state, a policy outputs actions that transition the agent from the current state into the next state. To quantify how well the policy performed, we need to assign a reward to the action that was taken. You are tasked to design the reward function.

(7/10 points) **Stable forward locomotion cycle**

Complete the following function:

private float computeRewardStandard()

This function returns a reward of the current state after performing an action sampled from the policy. You can return any reward you like, but to facilitate training, it is a good idea to assign a higher reward on more desirable states and less reward on less desirable states. For example, if you want to keep the agent moving in a straight direction, one idea is to assign the reward based on how close the agent's moving direction aligns with the target direction. Similar ideas can be applied to control the moving speed etc. You can also combine multiple rewards by summing or multiplying them together.

Some tips for designing the reward:

- Keep the reward between 0 and 1, where 1 is the maximum reward and 0 is the minimum.
- Have multiple rewards where each of them represents a goal the agent is trying to achieve. Then sum the rewards together as a weighted sum up by assigning higher weights on rewards that matter more.
  - Your reward function should allow the agents to move forward in +z direction with a stable locomotion cycle after training.

- After you've designed the reward function, explain what your design choices are and why you think that'll help facilitate training in your **answer.txt**. If you've taken any ideas from outside sources, make sure to cite them in your text file.
- Submit your best **MarathonManStandard.onnx** trained model. Your model will be graded on how well your agent performs.

(3/10 points) **Variations in locomotion style, direction, etc.**

Complete the following function:

private float computeRewardVariation()

For this reward function, you're allowed to explore variations that lead to different results from the standard reward function you've implemented above.

Some possible variation ideas you can explore:

- **Different locomotion style:** E.g., if your standard reward function results in the agent walking, you can try to make the agent run with this reward function.
- **Different moving direction:** You can explore making the agent move in any direction other than +z.

- Your reward function should allow the agent to move according to the direction, speed, or other goals you have decided for your variation.
- After you've designed the reward function, clearly explain what variation you chose, what your design choices are for reward computation, and why you think that'll help facilitate training in your **answer.txt**. If you've taken any ideas from outside sources, make sure to cite them in your text file.
- Submit your best **MarathonManVariation.onnx** trained model. Your model will be graded on how well your agent performs.

Make sure to call the correct reward function (computeRewardStandard or computeRewardVariation) in the OnActionReceived function while training the models.

Submission

Please create a zip named as **studentid.zip** (e.g. 123456789.zip) containing your **MarathonManAgent.cs**, **answer.txt, MarathonManStandard.onnx,** and **MarathonManVariation.onnx** files. At the top of your files, please leave a comment that includes both your **name** and **student id**.

Submit your zip file on CourSys.