Getting Started in Unity with ML-agents

An Applied Tutorial

Machine Perception and Cognitive Robotics Lab

MPCR LABS, FLORIDA ATLANTIC UNIVERSITY

HTTPS://GITHUB.COM/MPCRLAB/BRAIN/BLOB/MASTER/README.MD

This research was done under the supervision of Dr. William Edward Hahn and Dr. Elan Barenholtz.

First release, Fall 2020

Contents

1 Introduction 5

1.1 Objective 5

1.2 Useful Links 5

1.3 The Workspace 5

1.3.1 Download Unity for Windows or Mac 6

1.3.2 Download the ML-agent API 6

1.3.3 Download Unity for Ubuntu 6

1.3.4 Setting Up Unity Hub 6

2 Creating the Model 8

2.1 Create a Scene 8

2.2 Implement Agent 8

2.3 Integrate Scripts 10

2.4 Checking the Scene Set-up 11

2.5 Training the Model 11

3 Basics of ML-Agent 13

3.1 Model Outline 13

3.2 Planning 13

3.3 Scene Components 14

3.4 Agent Endowment 15

3.5 Training 17

4 Conclusion 19

5 References 20

1. Introduction

1.1 Objective

Artificial Intelligence research is coming to an exciting turning point were hardware and neuroscience are finally working together. Simulation engine, Unity, is making this possible more than ever with their new ML-agent API. The combination of tools described in this getting started workbook allows researchers to build novel simulations and investigate intelligent agents in the blink of an eye! This guide is only meant to condense the amazing documentation on Unity's official web page and the opinions presented in here are solely our own. The goal of this text is to get you building your very own AI agents and environments now!

1.2 Useful Links

These are some links from Unity. You don't need to read them now, but keep them in mind if you get stuck. https://docs.unity3d.com/Manual/upm-ui-install.html https://github.com/Unity-Technologies/ml-agents/blob/release$_2$/$docs/Python-API.md$ https://github.com/Unity-Technologies/ml-agents/blob/release$_2$/$docs/Learning-Environment-Create-New.md$

1.3 The Workspace

To be successful in this tutorial, you will need a working computer with osx, linux, or windows operating system, a reliable internet connect (for download), and a considerable amount of disk space to install Unity and the required packages and save your work. These instructions were generated while using Ubuntu OS, but should be similar for other OS. You also need python3.6 or higher installed and tensorboard

```
pip3 install tensorboard
```

.

R Start thinking of what you want to build first! I recommend you choose something fairly simple with room to improve upon. If you can't think of anything right now, just do the tutorial as is. You can come back later and easily make modifications.

1.3.1 Download Unity for Windows or Mac

- Go to this link: https://store.unity.com/plans-individual
- Under the student or personal package, click "Sign Up" or "Getting Started".
- Follow the clickable instructions. As a student you will have to navigate through several pages until you get to "First time users" Click "Start Here" and follow download instructions.
- Open the executable download file in Windows or similarly for Mac and.
- Install the program and open Unity Hub.

1.3.2 Download the ML-agent API

- Clone this repository:

```
git clone --branch release_1 https://github.com/Unity-Technologies/ml-agents.git
```

- Install the API directly by

```
pip3 install mlagents
```

1.3.3 Download Unity for Ubuntu

- Go to this link: https://store.unity.com/plans-individual
- Under the student or personal package, click "Sign Up" or "Getting Started".
- Follow the clickable instructions. As a student you will have to navigate through several pages until you get to "First time users" Click "Start Here" and follow download instructions.
- Open a terminal and navigate to where the file downloaded (probably in Downloads).
- Make the file executable by typing

```
chmod +x UnityHub.AppImage
```

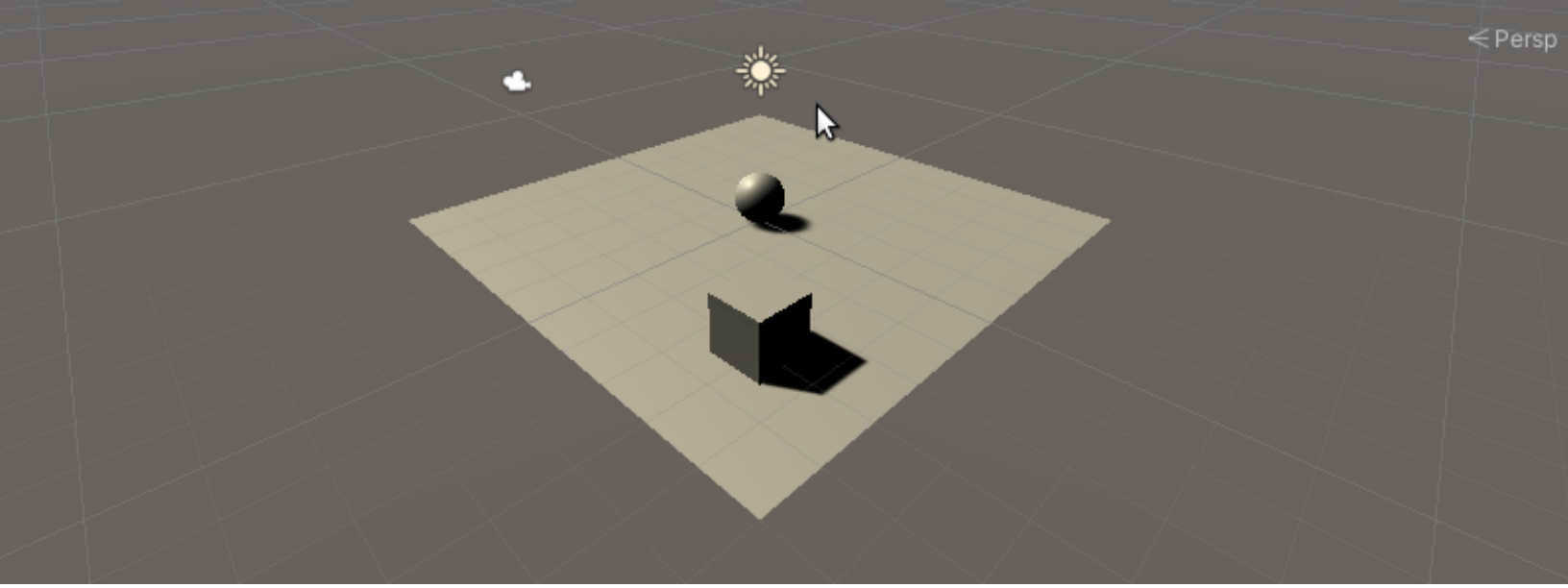
- It may be useful to make a copy of the UnityHub.AppImage and put it on your desktop (copy paste) or wherever you would like to access it from.
- Open the file manager and navigating to the UnityHub.AppImage file. Double click to open it.
- Unity Hub should open and now you can continue the instructions bellow.

1.3.4 Setting Up Unity Hub

- Once Unity Hub is open, if you don't have a previously installed version of Unity, you'll need to click on the "Installs Tab" and then click "Add".
- You'll be prompted to select a version of Unity and support packages (e.g. Microsoft Visual Studios on Windows) and install them (this will take a while depending on your internet speed). Restart your computer when this is done.
- Open Unity Hub and click on the "Projects" tab and select "New". Select the 3d model and give the project a name and location. The Unity interface should open (don't worry if it looks intimidating, we'll cover the basics soon).

- At the top of the interface, click on the tab called "Window" and select "Package Manager". A new interface should open.
- Click on the top tab next to the plus button and select "Unity Registry" or "All Packages".
- Scroll down and select "ML Agents". Click "Install" and once the install is complete, close the package manager window.
- The package we just installed should automatically be added to the project. Check this by going to the Project panel under the Packages folder and making sure you see "ML Agents".

You are ready to create your model and get your agents learning!



2. Creating the Model

The first thing to do in making your own simulated model with Unity and ML-agent is to make the environment. During environment making, you will have to specify the agent, behaviors, and actions. In this chapter, we will walk through creating the environment, and thus the model, in a step-by-step fashion.

2.1 Create a Scene

- Under the "Hierarchy" tab click the plus button and select "3D object" then select "Plane". A new gameobject will appear in the scene and in the list of gameobject under the hierarchy tab.
- Click on the 'Plane" object. In the "Inspector" tab the object's properties will appear. Rename this Floor by right-clicking and selecting rename.
- Click the plus button (or right-click in the empty space) under hierarchy tab and select 3D object then cube. Rename this Target.
- Click and drag the cube in the scene to move its positon. Or click on the cube in the hierarchy tab and manually set the position in the inspector tab.
- Add another gameobject, 3d sphere. Under the inspector tab at the bottom, click "Add Component" and search in the search bar that appears for "rigid body" to add. Rename this RollerAgent.
- For this tutorial we want the sphere and cube to be far apart. So set cube positions 3, .5, 3 and sphere positon 0, .5, 0.
- We now have an environment floor (plane) target (cube) and agent (sphere).
- "Create Empty" in the hierarchy tab and click on the GameObject. You can change the name by right-clicking on GameObject and selecting rename. This will become our training area.
- Drag the agent, target, and floor into the training area gameobject.

2.2 Implement Agent

- Click on the agent and add a script component by searching for "New Script" and naming it.

- Double click on the script name under the name of the script in the inspector tab. This will open the script to be edited.
- Delete everything in the script and replace it with the following

```
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;


public class RollerAgent : Agent
{
    private Rigidbody rBody;
    void Start()
    {
        rBody = GetComponent<Rigidbody>();
    }

    public Transform Target;
    public override void OnEpisodeBegin()
    {
        if (this.transform.localPosition.y < 0)
        {
            // If the Agent fell, zero its momentum
            this.rBody.angularVelocity = Vector3.zero;
            this.rBody.velocity = Vector3.zero;
            this.transform.localPosition = new Vector3(0, 0.5f, 0);
        }

        // Move the target to a new spot
        Target.localPosition = new Vector3(Random.value * 8 - 4,
                                           0.5f,
                                           Random.value * 8 - 4);
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        // Target and Agent positions
        sensor.AddObservation(Target.localPosition);
        sensor.AddObservation(this.transform.localPosition);

        // Agent velocity
        sensor.AddObservation(rBody.velocity.x);
        sensor.AddObservation(rBody.velocity.z);
    }
```

```
    public float speed = 10;
    public override void OnActionReceived(float[] vectorAction)
    {
        // Actions, size = 2
        Vector3 controlSignal = Vector3.zero;
        controlSignal.x = vectorAction[0];
        controlSignal.z = vectorAction[1];
        rBody.AddForce(controlSignal * speed);

        // Rewards
        float distanceToTarget = Vector3.Distance(this.transform.localPosition, Target.l

        // Reached target
        if (distanceToTarget < 1.42f)
        {
            SetReward(1.0f);
            EndEpisode();
        }

        // Fell off platform
        if (this.transform.localPosition.y < 0)
        {
            EndEpisode();
        }
    }

    public override void Heuristic(float[] actionsOut)
    {
        actionsOut[0] = Input.GetAxis("Horizontal");
        actionsOut[1] = Input.GetAxis("Vertical");
    }
}
```

- The agent is now integrated with the ML-agent package, we have described general behaviors of the model, allowed the agent observations, and applying action and reward information.

2.3 Integrate Scripts

- Click on the agent and add the "Decision Requester" script via the add component button.
- Change the "Decision Period" slide in the inspector tab under Decision Requester to 10.
- Click and drag the target under the agent in the hierarchy tab into the RollerAgent inspector tab under the Roller Agent (Script) box into the "Target" "none" box. The box should change to Target (Transform).
- Click on the agent and in the inspector tab, add "Behavior Parameters" script with Add Component.

- Change the name of the behavior to match the project name, vector observation size of 8. Change the vector action space type and space size to continuous and 2.
- We just connected the Agent to the environment and target.

2.4 Checking the Scene Set-up

- In the agents inspector tab, change the Behavior Type to "Heuristic Only".
- Press the play button at the tip middle of the scene window or use ctrl + p. Now use the keyboard to move the sphere. Make sure you don't get any error in the project console tab. When you first enter play mode, it may take a couple of seconds to switch the view. Once it switches view you can take control of the agent.
- You should see a warning that you can't connect to trainer on port... this is normal because we haven't connected a learning algorithm yet.
- Any changes you make while in play mode might not apply properly. Stop play mode by pressing the pause button or ctrl+shft+p.

2.5 Training the Model

- Open a terminal and navigate to the ml-agents repository we cloned earlier. Go into the config folder and we need to add a yaml file with hyperparameter options.

```
cd ml-agents/config
```

- Make the yaml with the following terminal command:

```
cat > rollerball_config.yaml
```

- Now you can enter the training hyperparameters by pasting or typing the follow into the same terminal:

```
behaviors:
  RollerBall:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      beta: 5.0e-3
      buffer_size: 100
      epsilon: 0.2
      lambd: 0.95
      learning_rate: 3.0e-4
      learning_rate_schedule: linear
      num_epoch: 3
    network_settings:
      normalize: false
      num_layers: 2
      hidden_units: 128
```

```
  use_recurrent: false
  memory_size: 128
reward_signals:
    extrinsic:
      strength: 1.0
      gamma: 0.99
time_horizon: 64
summary_freq: 10000
max_steps: 5.0e4
```

- Press ctrl+shft+d to save the text into the yaml.
- In the terminal in the ml-agents folder run:

```
tensorboard --logdir results --port 6006
```

- In another terminal in the ml-agents folder un:

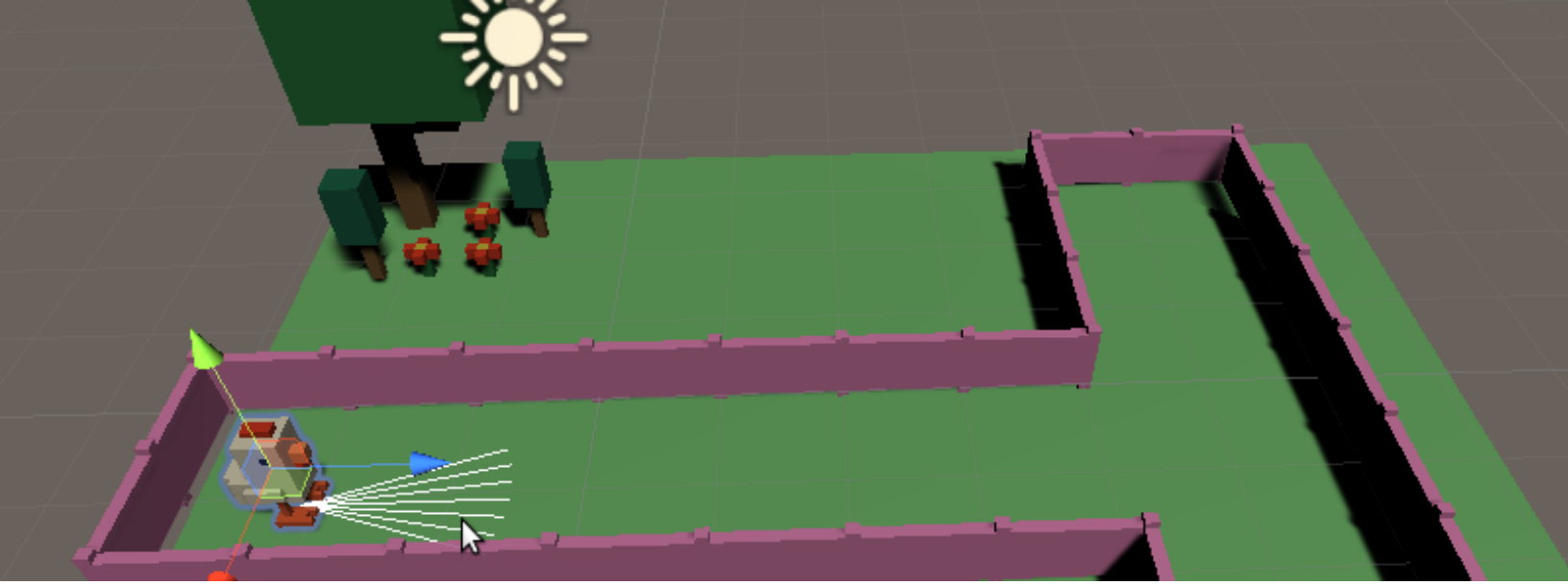
```
mlagents-learn config/rollerball_config.yaml --run-id=RollerBall
```

- You should see giant letters spelling out Unity in the terminal, keep it open.
- Quickly go to the Unity editor and hit play once. You should begin to see your model training in the unity editor. Note that if you have to run the training command above, you will need to specify a new run-id.

CONGRATULATIONS!

You just made and trained your very own RL model from scratch in Unity3D with ML-agent!

Keep going! Let's investigate what we just did by going through the important steps. You can use the information in the next section to start thinking of how to modify and expand these techniques for your own unique project.



3. Basics of ML-Agent

In this chapter we will use the previous model we built as a means to understanding the basic components of ML-agent. The motive for this is to learn when and what we need to change for future projects.

3.1 Model Outline

- Plan your project.
- Make objects of the scene.
  - Design agents, targets, and the environment.
- Implement the agent.
  - Give the agent physics emulators, agent script, behavior decisions, and behavior parameters.
- Set up the training configurations

3.2 Planning

In the above section we had the planning completed for us. In general, you will want to think ahead how you will design your model. You need to know what agent(s) will interacting with the environment and target(s). You also need to think of the reward function, type of learning algorithms, and hyperparameters necessary to complete the task.

In our model above we wanted to teach an agent how to find a target. In this case we used a sphere as the agent and a cube as the target. The environment these two objects engage on was the a plane we used as the floor.

Next we had decided the agent should roll to get navigate the environment. This rolling behavior would be continuous and terminate in two conditions: rolling off the floor or reaching the target. We also needed to know what should happen to the agent should it not reach the target. In this case it

either timed out or fell off the florr and was reset to a random position on the floor.

The decision of rolling would be determined by the agents observation, which we decided would be the agent and target positions and the the velocity of the agent.

We then planned for our agent to have two actions. The agent needed to control its velocity and position. Note how the actions and observations are intertwined. In order for the agent to be able to control its position and velocity, it needs to observe its position and velocity.

In order for the agent to learn, it needs a proper reward system. We determined this by only giving the agent a reward for reaching the target.

Finally, decisions on the learning algorithm and hyperparameters were selected. These are sometimes difficult to decide, but in general we can start small and simple and scale as we grow. In our case, we used the PPO algorithm and the standard hyperparmeters in the config file. While you may not need to know all of your hyperparameters before you start, having and idea of what decisions you need to make when it comes time to implement them will help. However, it is sometimes important to choose the learning algorithm before you start since different algorithms are built for specific tasks.

At this point we were ready to start the tutorial and jumped right in.

3.3 Scene Components

- Set up your editor. Make sure to import all the packages and assets you would like to use. For our model we only needed to import the ML-agent package using the package manager.
- Design the environment first. Set up all the necessary background objects, which was only the floor in our case. Sometimes you will need a floor and sometimes not. If you want to create any objects, you will most likely use 3d cube or sphere (even for walls and stuff like that). Sometimes you might need to create cs scripts for objects that let you build more quickly. There are many online video tutorials on how to manipulate objects efficiently in the environment.
- While designing the environment you'll need to put in your target(s). In our case, we just added the cube and moved it where we needed it. Target creation can get more advanced as you learn to create more complex environments. You can import assests that are pre-built gameobjects (sometiems called models) from the assest store or sprites you create yourself.
- Creating the agent is the most important part. This is last stop on the scene design. In our tutorial model we used a sphere. A sphere was a good choice for our design because we wanted it to have rolling behavior. So create your agent with its motion and behavior dynamics in mind.
- Remember grouping your training area often simplifies the later processes. It can also keep your hierarchy tab manageable once the environment gets more complex.

3.4 Agent Endowment

- Now that your agent is designed, you may need to give it some basic principal components. For us, we added "rigid body" to allow the sphere to move according to a physics engine simulator. This allowed the continuous rolling we later described in behaviors. There's many options here that can be investigated by looking through the add component options.
- The next step is where things can get complicated if you're not paying attention: implementing the agent. Almost any project will need a new script using

```
Unity.MlAgents
Unity.MlAgents.Sensors
```

  .
- We also need to define the public class as the agent. In our model we did this by changing

```
MonoBehavior
```

  to

```
Agent
```

  . Remember the class name and the agent's name should be identical.
- The script also needs to be aware of the physics (and any other necessary) components. We accomplished this by calling the variable

```
Rigidbody rBody;
```

  and utilizing it upon the starting frame

```
rBody = GetComponent<RigidBody>();
```

- You can connect the target to the agent by making a public field to point to the target

```
public Transform Target;
```

  Once this is called, the Unity editor will have a target field in the inspector window asking you to point to the target. Which we did by dragging the target from the hierarchy window into the target box in the inspector window.
- The rest of defining the script pertains to calling these three essential methods:

```
OnEpisodeBegin()
CollectObservations(VectorSensor sensor)
OnActionReceived(float[] vectorAction)
```

  . How you call these will define how the agent interacts with the scene.

– OnEpisodeBegin() is called whenever the agent's task needs to start. In our model, we needed to randomize where the agent would spawn on the floor at the beginning of each task to increase its generalizability during training. Setting up this method involves specifying any operations that should be performed during the reset of each task episode. For us, we just needed to tell the agent and target to take a random position on the floor.

```
public override void OnEpisodeBegin()
{
    if (this.transform.localPosition.y < 0)
    {
        // If the Agent fell, zero its momentum
        this.rBody.angularVelocity = Vector3.zero;
        this.rBody.velocity = Vector3.zero;
        this.transform.localPosition = new Vector3( 0, 0.5f, 0);
}

// Move the target to a new spot
Target.localPosition = new Vector3(Random.value * 8 - 4,
                                   0.5f,
                                   Random.value * 8 - 4);
}
```

– Now that the agent and target are placed on the scene at the beginning of the episode we need to tell the agent what to observe. This is done with

```
public override void CollectObservations(VectorSensor sensor)
```

. In our model, we let the agent see the position of the target and the position of itself with

```
// Target and Agent positions
sensor.AddObservation(Target.localPosition);
sensor.AddObservation(this.transform.localPosition);
```

we also let the agent know its own information from the physics component. In this case, velocity

```
// Agent velocity
sensor.AddObservation(rBody.velocity.x);
sensor.AddObservation(rBody.velocity.z);
```

Notice this observation vector will be of size 8 since position has x,y,z coordinates.

– Finally we can give action options and rewards with

```
public override void OnActionReceived(float[] vectorAction)
```

In which we made a controlSignal that would parse vectorAction later decided by the ml-agent learning algorithm and given to the physics rbody component

```
Vector3 controlSignal = Vector3.zero;
controlSignal.x = vectorAction[0];
controlSignal.z = vectorAction[1];
rBody.AddForce(controlSignal * speed);
```

We then specified a simple reward of 1 if the agent reached the target within some small distance

```
float distanceToTarget = Vector3.Distance(this.transform.localPosition,

// Reached target
if (distanceToTarget < 1.42f)
{
  SetReward(1.0f);
  EndEpisode();
}
```

Realize that we must call EndEpisode to terminate the current run for the task. This will allow OnEpisodeBegin to set up a new trial for the task if the max epochs has not been reached. We also specified our alternative terminal event for the episode, falling off the floor

```
if (this.transform.localPosition.y < 0)
{
    EndEpisode();
}
```

- We then finished connecting the set-up by adding decision requester to state when to ask for action decisions and the behavior parameter components when we described the size of the observation and action vector spaces.
- This is where you want to test your model set-up manually. We did that by implementing heuristic only behavior type and adding input to the agent.cs file to accept keyboard movements as rbody action input:

```
    public override void Heuristic(float[] actionsOut)
 {
actionsOut[0] = Input.GetAxis("Horizontal");
actionsOut[1] = Input.GetAxis("Vertical");
 }
```

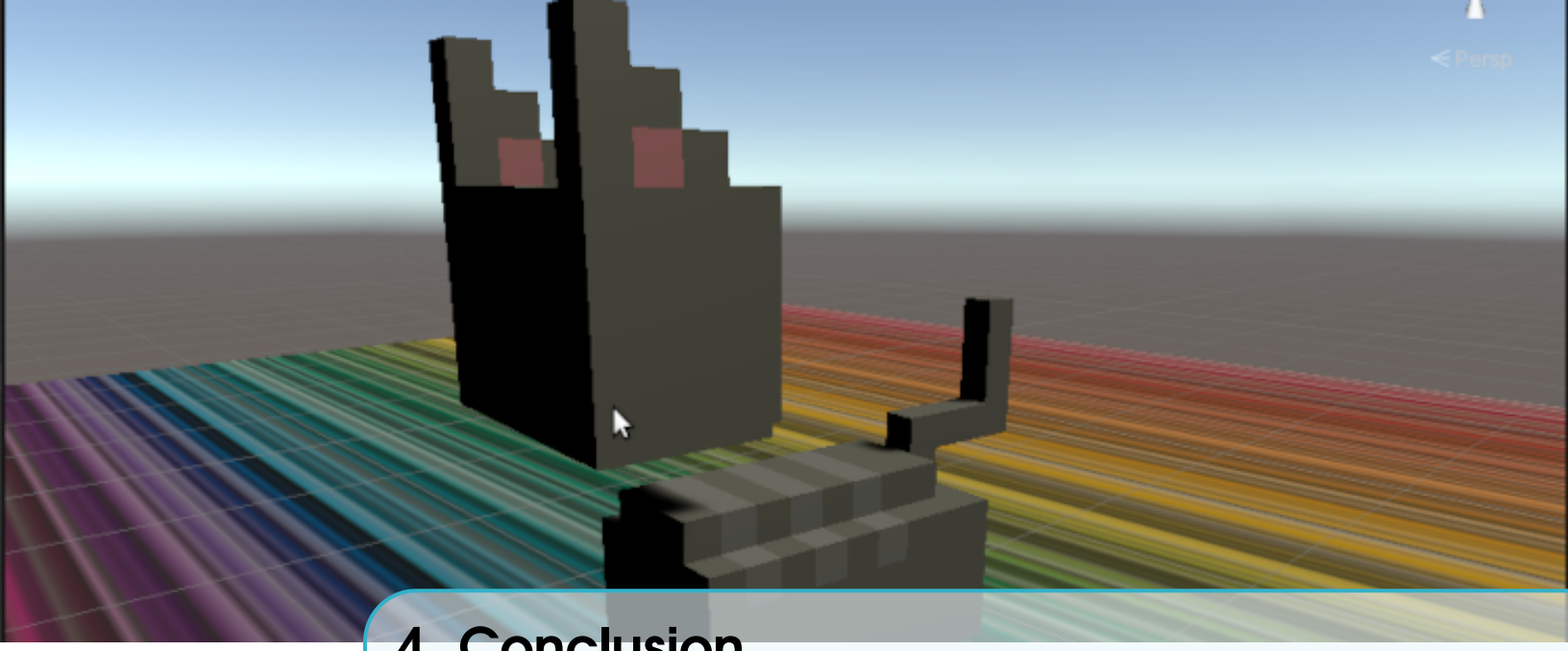
3.5 Training

- In order to get training, we needed to make the yaml file and place it in the ml-agents cloned directory config folder.
- In that yaml file we stated many different hyperparameters including which learning algorithm to use from ml-agent. We can also specify to use our own neural net this way or directly

accessing the api in a python environment by importin the unity environment.

- We then attacked tensorboard so we could watch our model train and started training by running the ml-agent train command

```
mlagents-learn config/rollerball_config.yaml --run-id=RollerBall
```



4. Conclusion

It's that simple! If you made it this far, you've created your own model in Unity3D and learned the fundamentals of making more complex and unique environments to keep your agents learning!

There's tons of resources online to help you figure out all the design and specific program aspects to implement you idea. Since ML-agent is a new and highly used API, make sure you are keeping up your dependencies and versions are compatible. Also beware that the rapid growth in the projects versions may make some docs outdated. If you get stuck, just keep trying!

5. References

All code was taken from the official Unity3d ML-agent github repository. I've tried to simplify the process without loosing the learning component of the tutorial. I hope you found this useful, especially for people like me who want to get the model working ASAP and then focus on learning the skill.

Here are the main pages I used to create this text:

{https://docs.unity3d.com/Manual/upm-ui-install.html}
{https://github.com/Unity-Technologies/ml-agents/blob/release_2/docs/Python-API.md}
{https://github.com/Unity-Technologies/ml-agents/blob/release_2/
docs/Learning-Environment-Create-New.md}

A special thank you to the creator of this beautiful latex template, Mathias Legrand, which can be found here http://www.latextemplates.com/template/the-legrand-orange-book].