Relevant Links

- **ML Agent repo:** https://github.com/Unity-Technologies/ml-agents
- **Unity tutorial:** https://learn.unity.com/course/ml-agents-hummingbirds
- **INVIVO repo:** http://gitlab.invivo.ca/invivo/ml-agents-hide-and-seek
- **Anaconda:** https://www.anaconda.com/products/individual

Getting started

1. Install any version following the Unity 2018.4 release
2. In the editor select Window → Package Manager → ML Agents

Making your agent

Make a new c# script. Have the class inherit from the 'Agent' class then Import ML Agent.

```
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
```

Before we go any further lets place this new class on a new empty game object that will become our agent. The agent will require **three** components, YourNewClass.cs, DecisionRequester.cs, and BehaviorParameters.cs. Expect to need a rigid body and collider as well, you can use whatever model you want to. Save this new game object as a prefab.

Filling out the class

📎 ChaseSpirit.cs 10.3KB

I have added an agent class example for a simple target chasing agent. Feel free to use it to follow along. There are a few key methods you will need to override from the Agent Class to get our agent functioning. They are...

1. public override void OnEpisodeBegin()
2. public override void CollectObservations(VectorSensor sensor)
3. public override void OnActionReceived(float[] vectorAction)

4. public override void Heuristic(float[] actionsOut)

On Episode Begin is called after a given number of decisions are made by the ML agent neural network. This number of steps is a configuration in the inspector attached to the agent and is used for training purposes.

Use this method to reset the environment, agent, and any other state specific information.

Collect Observations is called any time the agent is rewarded or punished with the *AddReward(float reward)* method. In this method create new observations to add to the VectorSensor that will be correlated to the reward received, and decisions made to inform future decisions.

Use *sensor.AddObservation()* to add any of the following as observations.

1. **EInumerable<float>**

2. **Quaternion** - 4 observation(s)

3. **Vector2** - 2 observation(s)

4. **Vector3** - 3 observation(s)

5. **bool** - 1 observation(s)

6. **float** - 1 observation(s)

7. **int** - 1 observation(s)

Keep track of how many observation you make with these, you will need it in the configuration step.

NOTE: The neural network has an easier time dealing with numbers between 0 and (-)1, you may find learning is faster if all information is normalized or made a ratio always less than 1.

On Action Received is called every time a new decision is made by the neural network. The network will provide an array of either discrete or continuous numbers for our use. As seen in the example we can generate a movement vector to apply to our agent. We can request as large of an array as we want for more complicated actions and decision making, this can be done in the configuration step.

It is important to use each input uniquely so that observations made inform only one decision and not several.

Heuristic is only called if your agent is set to do so in the configuration step. This method is used for manual override of control over an agent. This can be used for testing or even competitive learning for your agent. For our purposes, it is not very important.

Max Step determines how long the simulation will run before **OnEpisode()** is called. Each new episode uses the prior experiences to weight the decision making used in **OnActionReceived(float[] vectorAction)**. Feel free to experiment with this setting, I found less is better for simpler agents.

Behavior Name is used by the configuration file we will create momentarily, know that this name will need to match the one specified in this file.

Vector Observation is set based upon the number of observations made in **CollectObservations(VectorSensor sensor)**, the number needs to be added total of observations made.

NOTE: If observations are only made conditionally or require a conditional piece of information, create a check that adds an empty array at the same size as the missing observations.

Vector Action determines the size of the vector action array use in **OnActionReceived(float[] vectorAction)**. Be sure to only set space size to the amount you are using as not to pad the network with unused information.

Trainer configuration

📎 trainer_config.yaml 2.0KB

You can place this file anywhere you would like, just know that you will need to be able to access it later. I keep mine in my unity project under the ml_agents folder.

Each agent will require a unique configuration to optimize its learning rate and ability. Here is a trainer configuration file I created for the sample agent above. Understanding all the parameters is not necessarily required to get a functional agent but I encourage learning more about how the hyperparameters affect an agent. Increasing the **max_step** will increase how long your agent will train before stopping automatically. Other settings such as **buffer_size** and **hidden_layers** will increase the complexity of the agent and therefore make training take longer, but in theory, allows for more complicated decision making (not necessarily better).

A number example configurations and further explanation on the parameters can be found https://github.com/Unity-Technologies/ml-agents/tree/0.10.1/config as of 8/6/2020. If by chance the repository changes search 'ml agent Github' and the first result should yield the repository. Check out the most recent branch and browse the files either in ml-agents, config, or demos for other examples.

Adding External Sensors

A simple way of adding to your agent's observations is to add a unity Ray Perception Sensor 3D component.
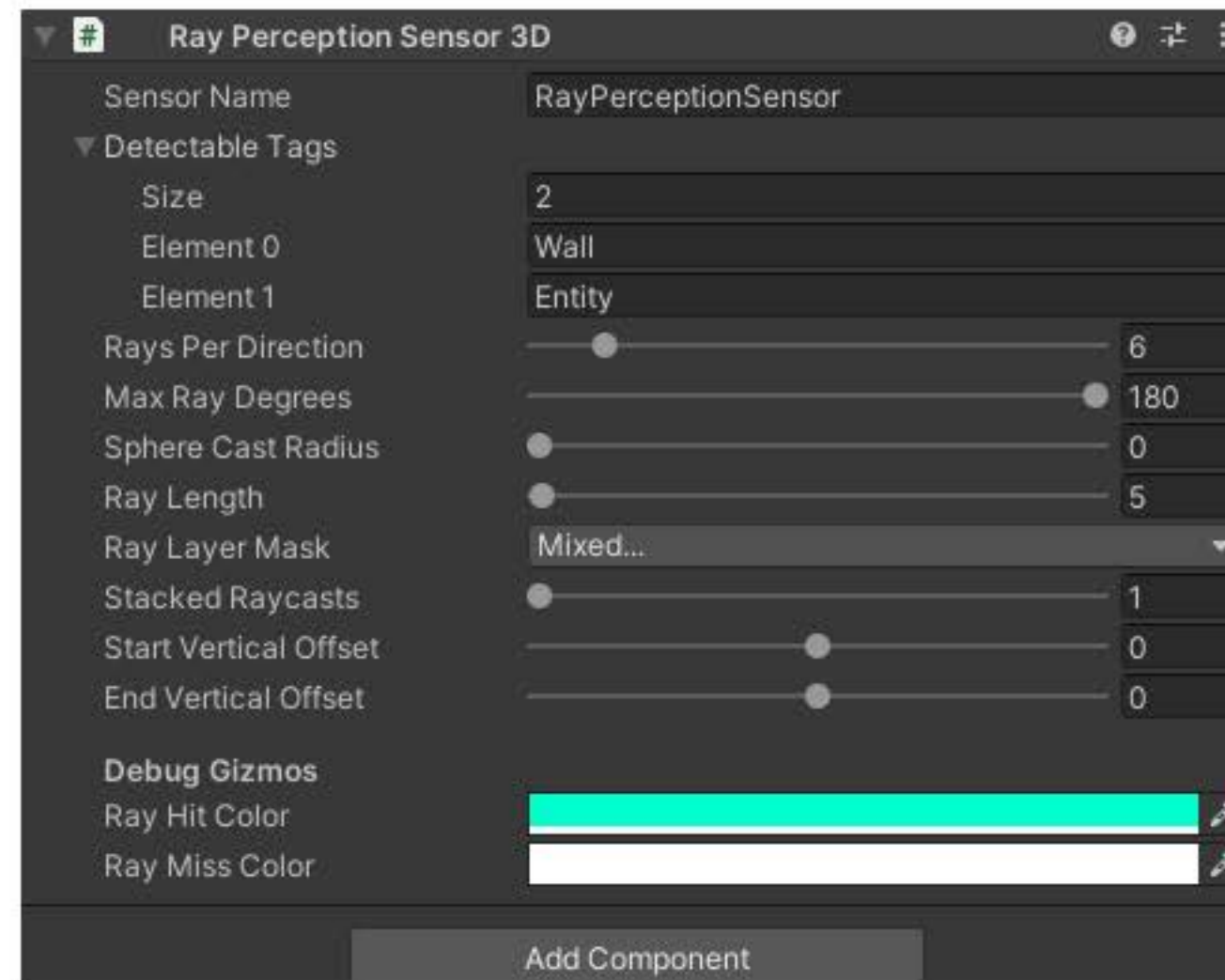
This component will shoot ray-casts out according to the displayed settings, to have these detect specific Game Objects you will need to manually create and add tags to said Game Objects then specify them under the 'Detectable Tags' array. Check off 'use child sensors' on your agent component to enable the use of the new observations.

Other requirements and Rewards

Your agent and environment will need support scripts to manipulate objects in the environment and the agent itself for resetting the simulation.

The agent class will need to be **Rewarded** for its behave to yield any results. This can be done in any number of ways but likely through a collision or trigger method that responds with a positive *AddReward(1f)* or negative *AddReward(-1f)*, depending on what it interacts with.

NOTE: The goal of the simulation should be to eventually average a score of +1 or higher. You will need to experiment heavily with how heavily you punish or reward the agent for its actions to get the results you seek. I have found that positive reinforcement leads to more curious and desirable behavior than heavy negative reinforcement, both should still be used, however.

Agent configuration

Settings that matter for our purposes are; Max Step, Behavior Name, Vector Observation, and Vector Action.

Preparing your python environment

ML Agent uses a python script to construct and teach the neural network. Before you can start to train an agent we will need to install python and the ml agent packages for python. I recommend using Anaconda as it is a simple solution to all our needs. Everything beyond this point will be in the context of Anaconda.

1. Download Anaconda individual edition for your system,
   https://www.anaconda.com/products/individual

2. Run Anaconda prompt

3. Create a new environment with python 3.7 installed with the command:
   conda create -n myEnvName python=3.7

4. Activate that environment with the command:
   conda activate myEnvName

5. Now that the python environment is created and activated we can install the ml-agents packages the pip command:
   pip install mlagents

These should be all the steps needed to set up your python environment and we can test if it is functioning down bellow.

Running your agent

First, let us test if the environment is set up correctly. Your Anaconda prompt once activated should look something like this.

```
(base) C:\Users\bryan.chafee>conda activate myEnvName

(myEnvName) C:\Users\bryan.chafee>_
```

Command to test ml agents intallation

- mlagents -learn

Anaconda prompt should populate with this display. Ignore cudart dlerror if you have it.

Go to the directory you placed the **trainer_configuration.yaml** file earlier, copy the address. Then in your Anaconda prompt (hit Ctrl-c to escape the agent session you started as a test) enter into that directory using the command:

- cd C:\Users\bryan.chafee\example

Now that we are inside the same folder as the configuration file we can begin training our agent! Ensure that your unity project is open and your agent and the scene is entirely prepared before we begin. Use the command:

- mlagents -learn ./trainer_config.yaml —run-id myAgent_00

You will then be prompted to hit play in your unity editor to start training. You can name your agent whatever you choose, just keep in mind every time you use the -learn command a new -id is required (hence the _00).

Common issues

- Make sure not to click inside of the anaconda prompt once the agent is running, it may freeze

- Behavior type on the agent component is not set to default

- Forgot to correctly add and set the sensor vector observations space size

- Make sure not to click inside of the anaconda prompt once the agent is running, it may freeze

- Behavior type on the agent component is not set to default

- Forgot to correctly add and set the sensor vector observations space size

Once you have verified that your agent is running in your editor feel free to duplicate your training area as many times as your CPU can handle.

Tensorboard

So your agent is running, but you are struggling to tell how well it is performing. ML Agents provides a powerful tool called **Tensorboard** that will display your agent analytics in real-time on any given web browser. To access this open a new Anaconda prompt and enter your previously setup environment with ml agents installed. Type the commands:

- cd C:\Users\bryan.chafee\example

- tensorboard —logdir results

You may then go to http://localhost:6006/, or the displayed host address to view various graphs for your agent's rewards and behaviors.

Example of three generations with only differences in reward values

Cumulative Reward
tag: Environment/Cumulative Reward