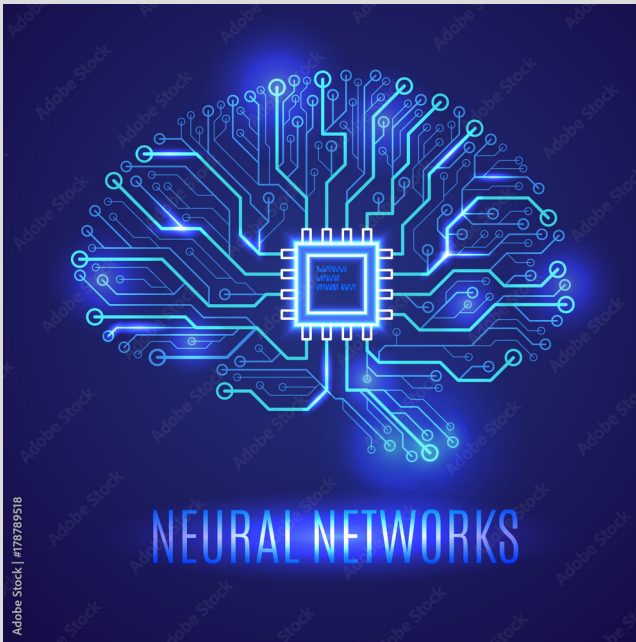
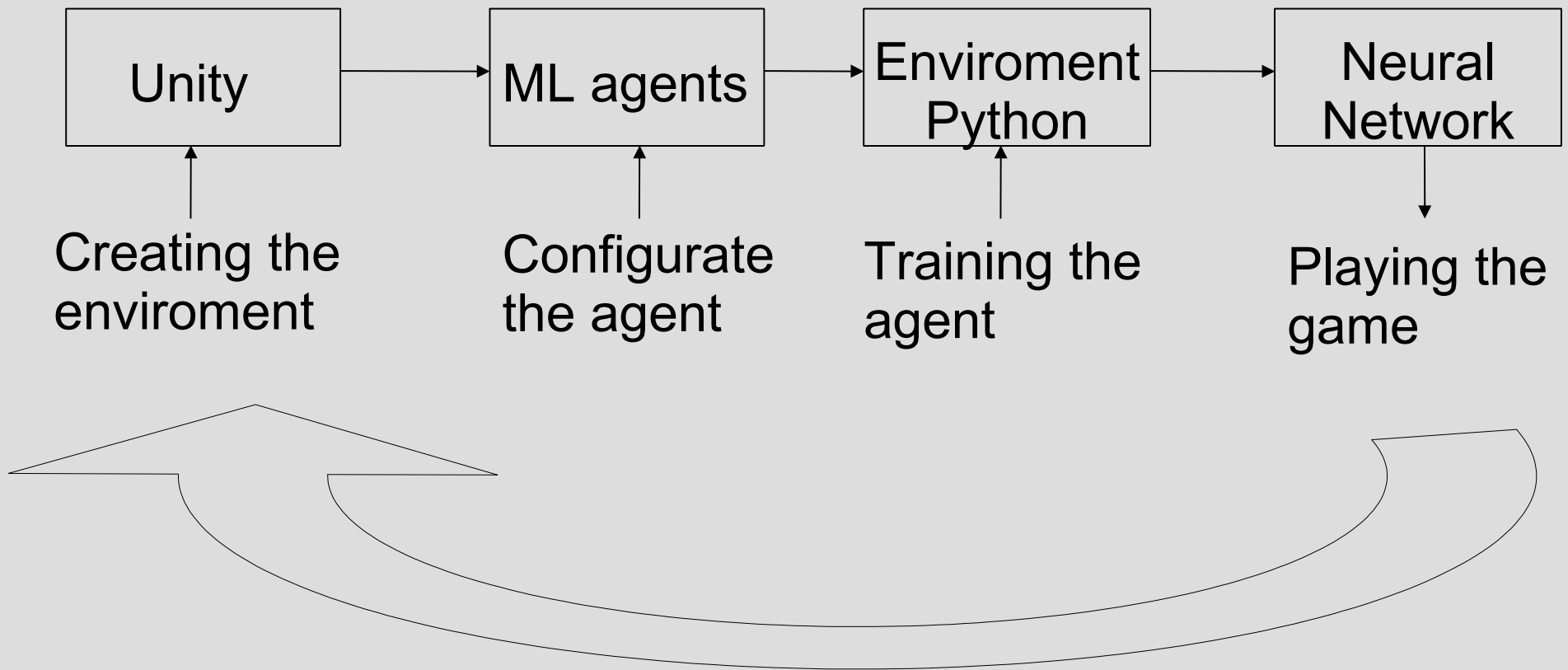


Deep Reinforced Learning Project



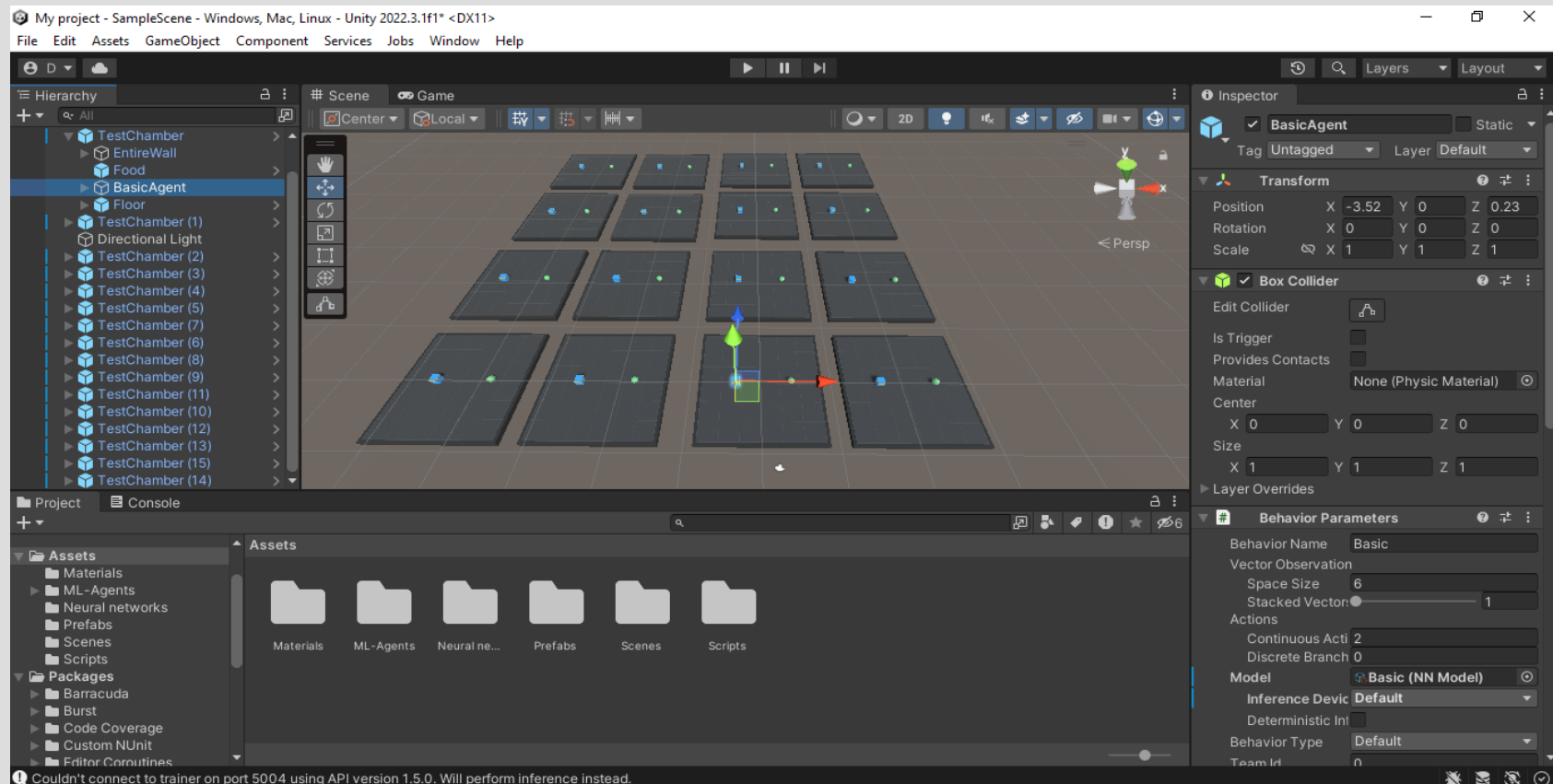
Created by Miguel Pinel Martínez

Structure of the project



Unity

Unity is a game engine design to be user-friendly and to allow anyone to design a videogame. We are going to use it to create an environment for our agent to learn.



ML agents package

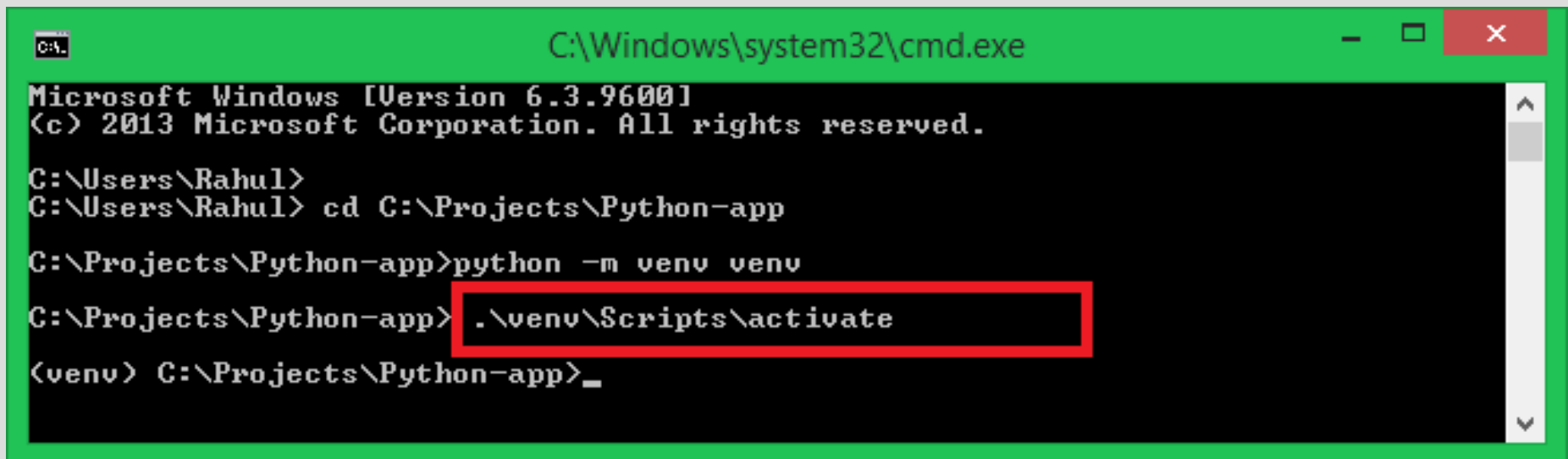
ML agents is a package of Unity that allow us to create machine learning agents controled by a neural network, to give does agents observations of the enviroments and to receive response from the agent we could transform into actions in the enviroment.

This package as the rest of Unity works in C#, but it will need a virtual enviroment of python in order to train the neural network.



Virtual Python Enviroment

The neural network will be created and trained using pytorch and a library of ML-agents for python. This virtual enviroment will receive the configuration information and the observations directly from Unity and we shouldn't need to interact directly with this enviroment. All the code for the project is in C# and we will control all the process of this enviroment directly from Unity.



```
C:\Windows\system32\cmd.exe

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Rahul>
C:\Users\Rahul> cd C:\Projects\Python-app

C:\Projects\Python-app>python -m venv venv

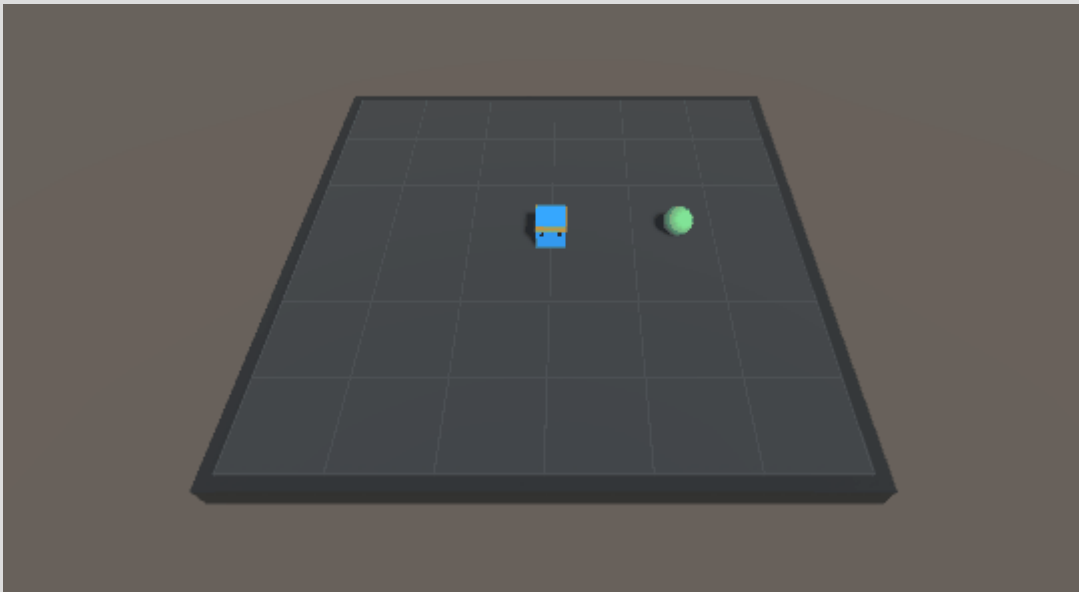
C:\Projects\Python-app> .\venv\Scripts\activate

(venv) C:\Projects\Python-app>_
```

Toy Problem: GridWorld

Environment: A square platform of 6x6 tiles that you can't leave and a piece of food

Goal: Eat the food.



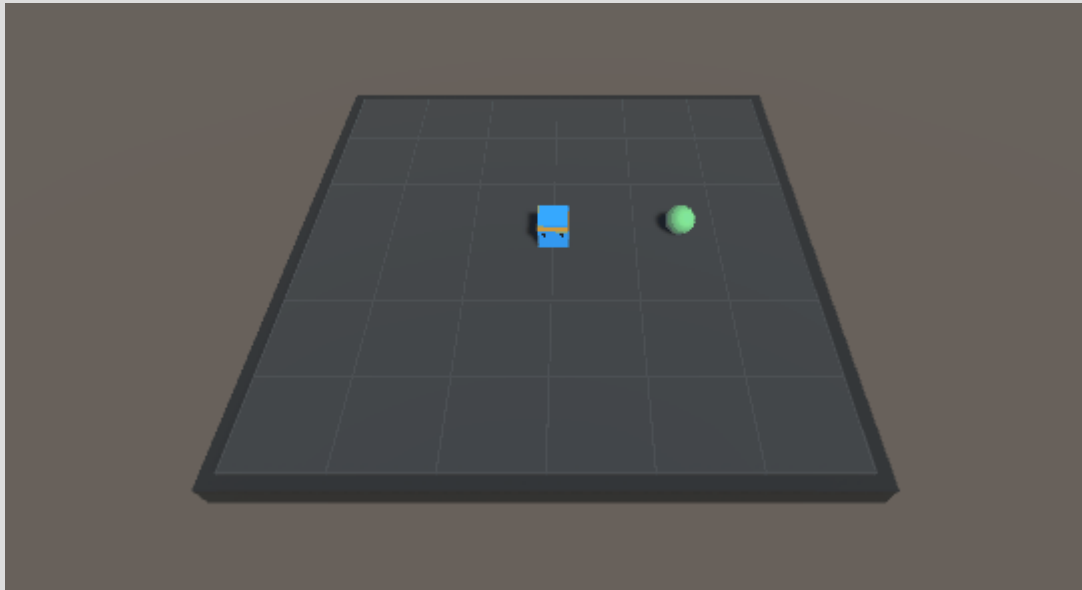
Human Gameplay

Possible actions:
Movement in the four
cardinals directions.

Rewards:
+1 for the food
-1 for leaving the
platform

Solution to Toy Problem

We are going to give the agent its position at every moment and the position of the food.



The agent learn to move to the food, even though not in the fastest way as velocity is not being rewarded in this problem

How is the agent learning?

We are solving this problem with a model-free approach. The agent will receive a serie of observations of the enviroment that will determined the actual state of it, then the neural network will give deterministically a action as response to that state and then the agent will forget all information about the state of the problem. The base architecture of the neural network will be two hidden layer with 120 neurons each.

But how are we training our neural network?

Proximal policy optimization (1)

Proximal policy optimization is a policy gradient algorithm. We have study that we want to calculate the following gradient to update our policy:

$$\nabla_{\theta} G = E(\nabla_{\theta} \log \pi_{\theta}(A|S) Q(S|A))$$

We are going to change a little bit the notation and we will change Q for A , the advantage, to create this function, the policy objective function, which we will try to maximize :

$$L^{PG}(\theta) = E_t(\log \pi_{\theta}(a_t|s_t) A_t)$$

$$A(s, a) = Q(s, a) - V(s)$$

Proximal policy optimization (2)

Unfortunately the function we have seen can give us really big changes in the policy. For this we are going to use the following function instead:

$$L^{CLIP}(\theta) = E_t(\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t))$$

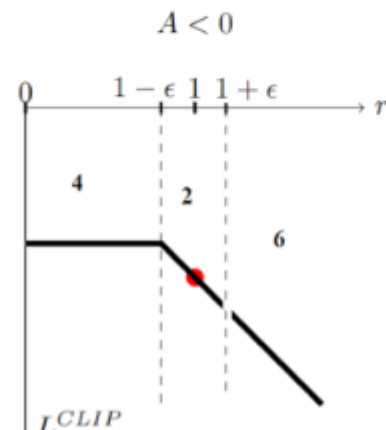
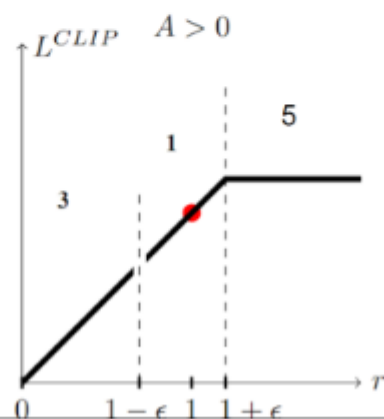
$$r_t(\theta) = \frac{\pi_{\theta}(a_t|a_s)}{\pi_{\theta_{old}}(a_t|a_s)}$$

The idea of this new function is that we will discard the actions that are drastically different from the previous policy and so only update the policy in small steps. However, when the new policy is worse then we allow big change to correct errors in the change of the policy

Proximal policy optimization (3)

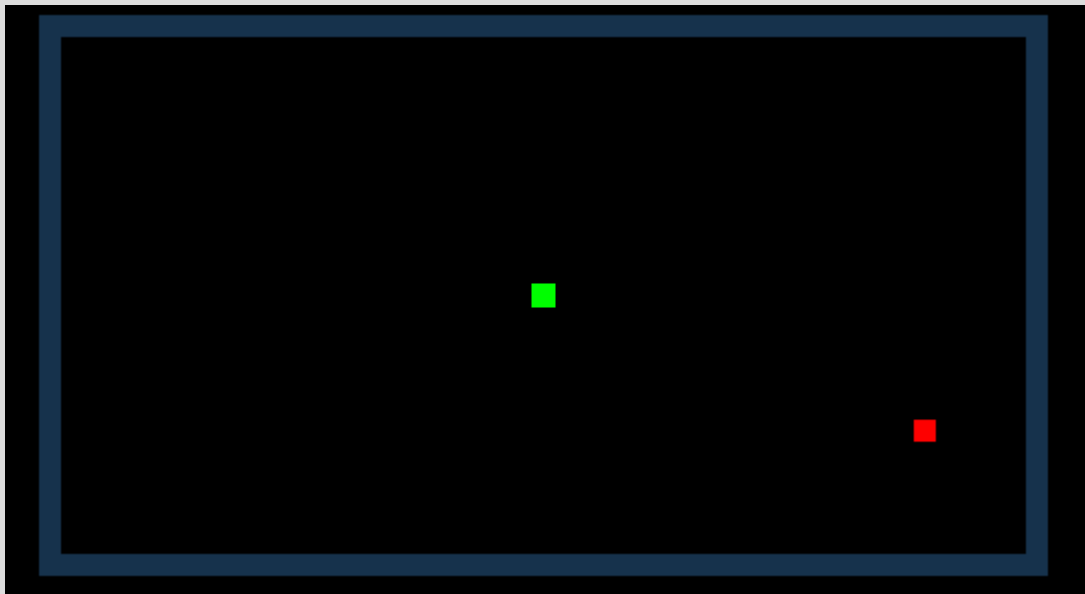
To help visualize the function, here p represents the ratio of the policies

	$p_t(\theta) > 0$	A_t	Return Value of \min	Objective is Clipped	Sign of Objective	Gradient
1	$p_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$	+	$p_t(\theta)A_t$	no	+	✓
2	$p_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$	-	$p_t(\theta)A_t$	no	-	✓
3	$p_t(\theta) < 1 - \epsilon$	+	$p_t(\theta)A_t$	no	+	✓
4	$p_t(\theta) < 1 - \epsilon$	-	$(1 - \epsilon)A_t$	yes	-	0
5	$p_t(\theta) > 1 + \epsilon$	+	$(1 + \epsilon)A_t$	yes	+	0
6	$p_t(\theta) > 1 + \epsilon$	-	$p_t(\theta)A_t$	no	-	✓



Problem statement: SNAKE

Environment: A rectangle of 43x23 tiles surrounded by walls that will kill you when touched and a piece of food that will move to a random location each time is collected.



Human Gameplay

Goal: Eat as many food as possible.

Possible actions:
Movement in the four cardinals directions.

Rewards:
+1 for each food
-1 for when dying

First small problem: Infinite loop

First problem we face, the agent learn to loop in circles to avoid hitting walls indefinitely. The main cause of this is that if the food appear near the edge it will take a lot of time for the agent to randomly reach it.

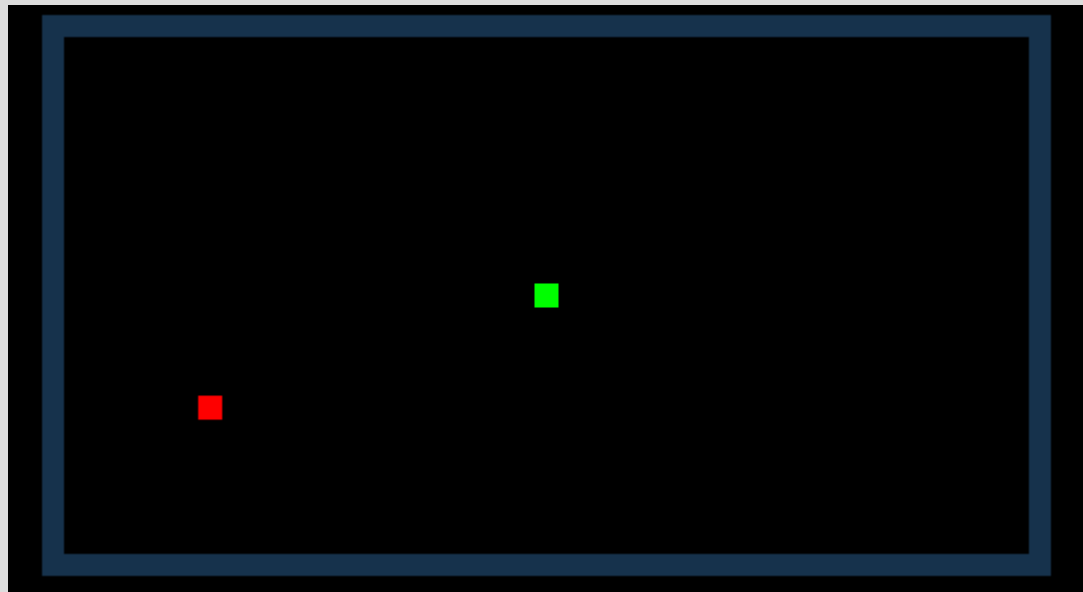


Solution: First we add a timer so that the player also die if it takes too long to reach the food. Secondly we randomize the position of the food after each death.

First approach: Blind Snake

We are going to give our agent two observations:

- Position of the head of the snake
- Position of the food



Results:

- Mean reward of 2.5

Problem:

- The agent has no way of knowing the tiles where there are parts of the body of the snake, so it die by hitting itself

Second approach: Full map view

We are going to give the agent a vector containing all the information of the map. Unfortunately we can't give a matrix as an observation in the ML agents package. So the map will be a vector of boolean values, where each tile will be represented by four booleans, differentiating the four possible objects: wall, food and body or head of the snake.

Results:

- No better than a random walk

Problem:

- The neural network is incapable of infer its position and the one of the food from all the observations(4500 total booleans)

Third approach: Attention is all you need

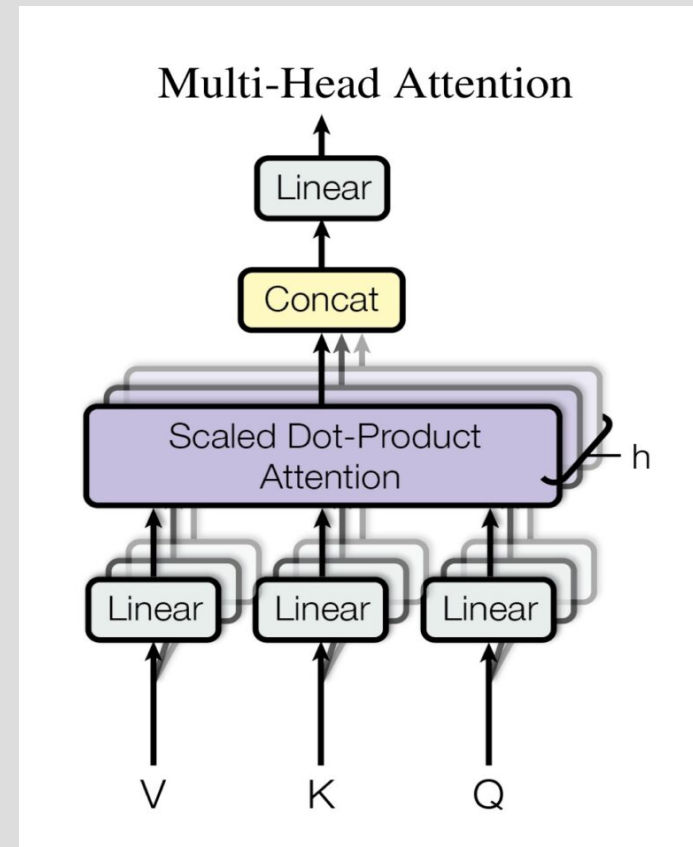
We are going to use the BufferSensor module to give the agent a variable number of observations, giving the position of the food, the head of the snake and all the parts of the body of the snake.

Results:

- Unable to train

Problem:

- The attention block is way too slow to reach a conclusive number of training steps.



Last approach: Occam's razor

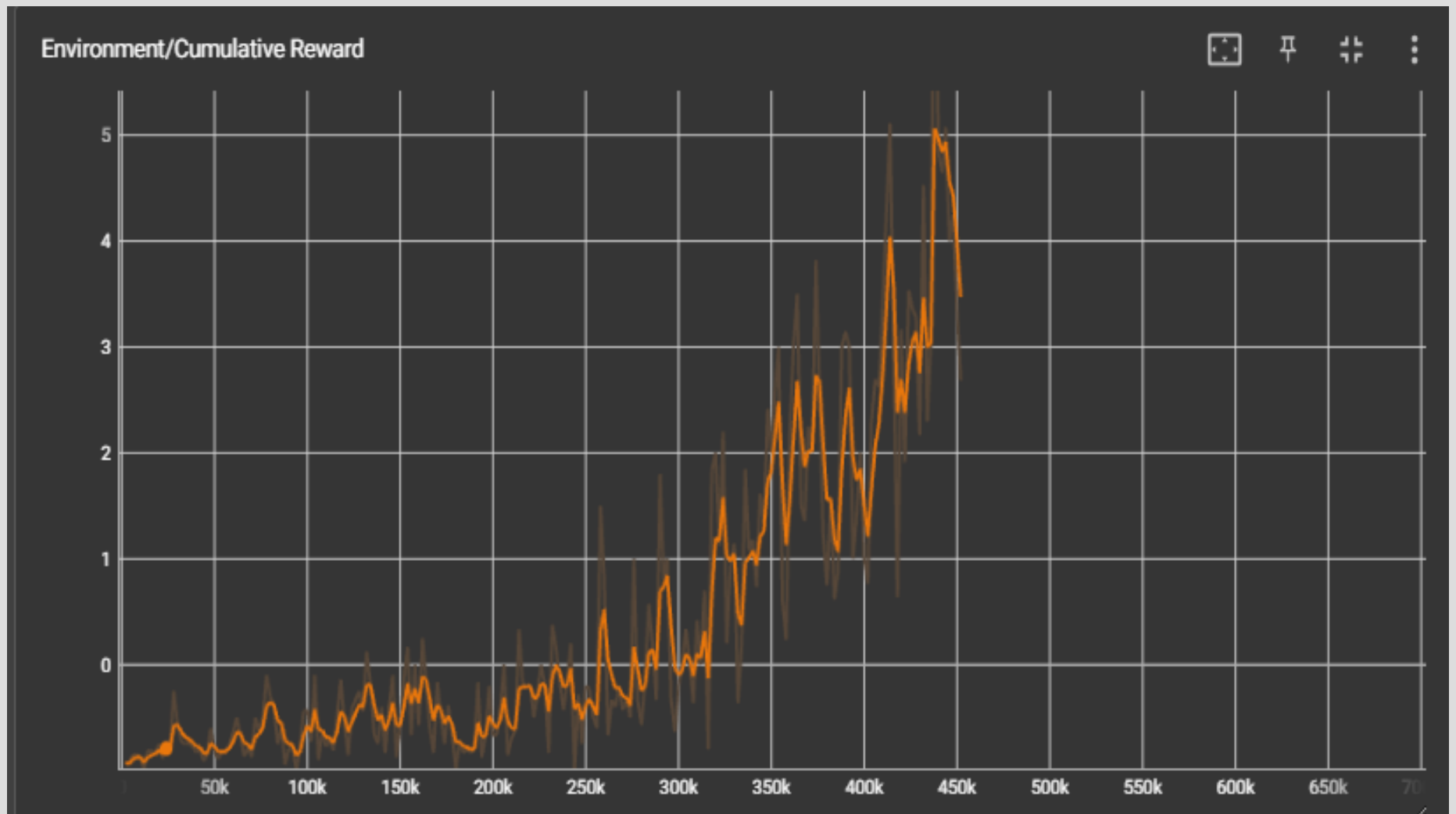
We are going to give our agent the position of the food and the head of the snake and also the distance in the four cardinal directions to the nearest wall or part of its body.



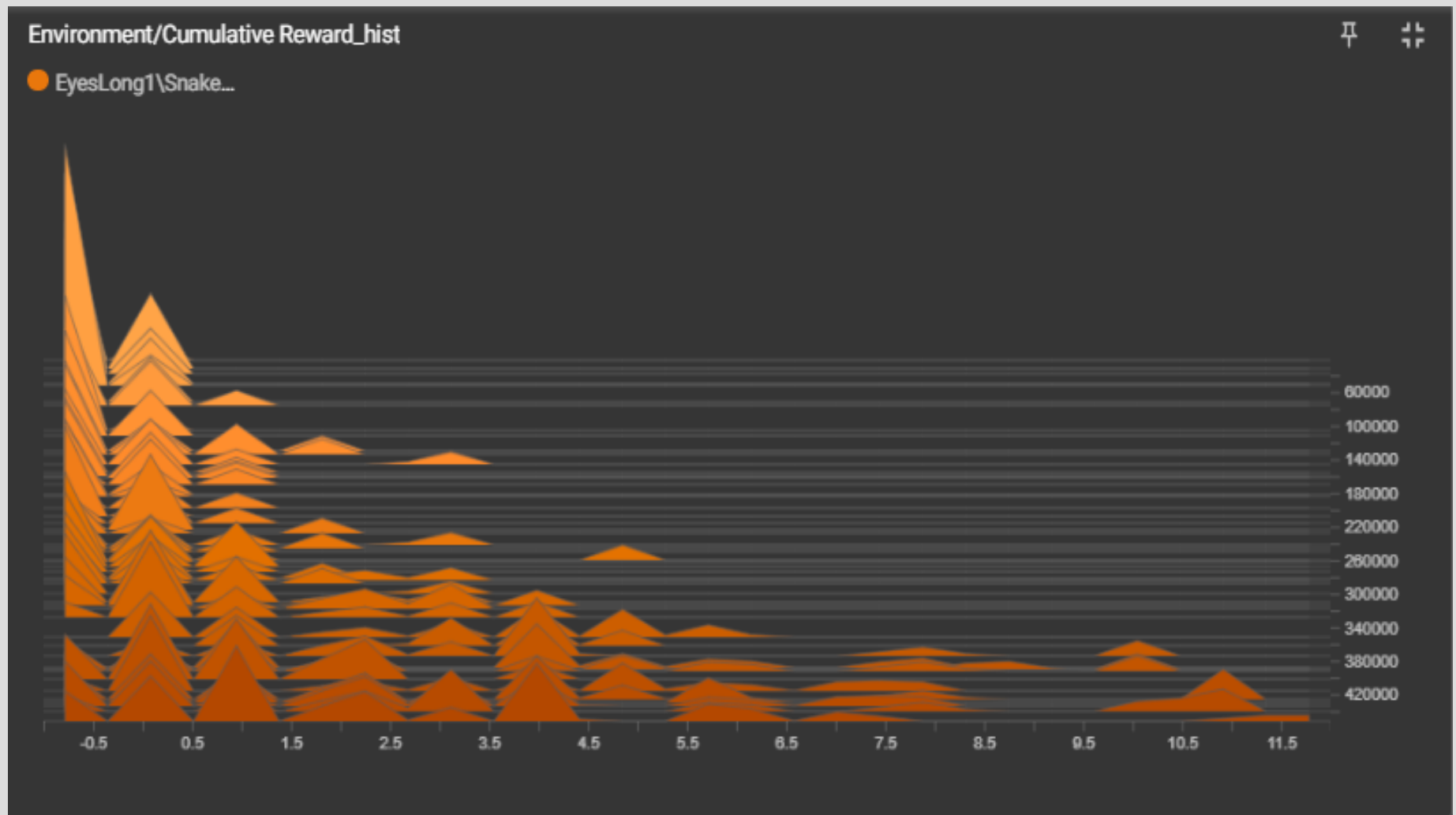
Results:

- Mean reward: 5.0
- Up to 11 reward

Statistics (1)



Statistics (2)



Statistics (3)

