

**Faculty of Information Technology**

**Department of Mechanics, Information Technology, and Robotics**

Specialization of Robotics

**Patryk Stradomski**

s16716

**Sarsamax implementation**

**with OpenAI Gym and Unity engine**

B.Eng Thesis

Thesis supervisor  
MSc.Eng. Piotr Gnyś

Warsaw, May 2021

Table of Contents

[1 Introduction 3](#_Toc72883457)

[1.1 Abstract 3](#_Toc72883458)

[1.2 Goals 4](#_Toc72883459)

[2 Reinforcement Learning 4](#_Toc72883460)

[2.1 Introduction 4](#_Toc72883461)

[2.2 Components 4](#_Toc72883462)

[2.3 History 5](#_Toc72883463)

[2.4 Markov Decision Processes 6](#_Toc72883464)

[2.5 Exploration versus Exploitation 8](#_Toc72883465)

[2.5.1 - Greedy policy 8](#_Toc72883466)

[3 Q-learning 9](#_Toc72883467)

[3.1 Value Functions 9](#_Toc72883468)

[3.2 Temporal Difference Learning 10](#_Toc72883469)

[3.3 Sarsamax 10](#_Toc72883470)

[3.4 Q-table 11](#_Toc72883471)

[4 OpenAI 12](#_Toc72883472)

[4.1 Introduction 12](#_Toc72883473)

[4.2 Mission 12](#_Toc72883474)

[4.3 History 12](#_Toc72883475)

[5 OpenAI Gym 13](#_Toc72883476)

[5.1 What is Gym? 13](#_Toc72883477)

[5.2 Environments 13](#_Toc72883478)

[5.2.1 Algorithms 13](#_Toc72883479)

[5.2.2 Classic control and Box2D 13](#_Toc72883480)

[5.2.3 Atari 13](#_Toc72883481)

[5.2.4 Mujoco and Robotics 14](#_Toc72883482)

[5.3 Interface 15](#_Toc72883483)

[6 Unity ml-agents 17](#_Toc72883484)

[6.1 Unity game engine (or Unity Real-Time Development Platform) 17](#_Toc72883485)

[6.2 Machine learning with Unity 17](#_Toc72883486)

[6.3 Environment examples 17](#_Toc72883487)

[6.3.1 Crawler 17](#_Toc72883488)

[6.3.2 Sorter 18](#_Toc72883489)

[6.3.3 GridWorld 19](#_Toc72883490)

[6.4 Robotic simulation 19](#_Toc72883491)

[*7* Q learning implementation using ml agents – ai gym hybrid.? 20](#_Toc72883492)

[7.1 Basic environment 20](#_Toc72883493)

[8 Experiments 22](#_Toc72883494)

[Equal distance to the rewards. 23](#_Toc72883495)

[9 Conclusions 26](#_Toc72883496)

[10 References 26](#_Toc72883497)

[11 Appendix 27](#_Toc72883498)

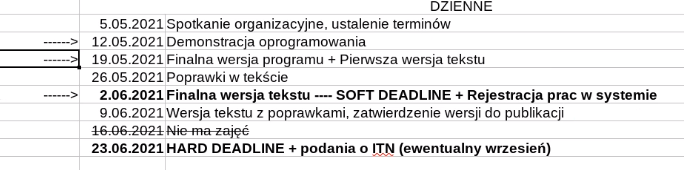
[11.1 How to create a unity executable – to appendix 28](#_Toc72883499)

[11.2 Using Ai gym with unity executable – to appendix 28](#_Toc72883500)

[11.2.1 UnityEnvironment 28](#_Toc72883501)

[11.2.2 UnityToGymWrapper 28](#_Toc72883502)

[11.3 Algorithm implementation – to appendix 28](#_Toc72883503)



# Introduction

## Abstract

This paper is focused on presenting a Q-Learning solution to a simple reinforcement learning problem that has been implemented in Unity Game Engine. The method has been written as a python script which communicates with the unity executable with the help of Unity ML-agents package through an interface that was first implemented by OpenAI Gym. The paper begins with familiarizing the reader with reinforcement learning; more specifically, the aspects of it that are necessary to have a clear understanding of the proposed solution. Please note that reinforcement learning is an ever-growing field and not all relating topics are covered in the aforementioned description. Then a brief description of OpenAI, Gym, Unity Game Engine and Unity ML-Agents is given in order to provide more context on how the solution works and how said parts are integrated. After that the titular solution is explained and effects of experiments are shown.

The paper presents the solution implemented using Python and Unity-ML agents on Unity-made environment as easy to do and straightforward. It is proposing that the framework brings significant value by considerably quickening the modeling process of reinforcement learning solutions.

## Goals

Researchers working on reinforcement learning should be able to spend most of their time and resources on implementing the most efficient solution rather than on preparing environments and tools. The goal of this paper is to present the ease of integration of reinforcement learning solutions with an environment prepared in Unity Game Engine through Gym by OpenAi by implementing a Q-learning algorithm and running it on a prepared executable.

# Reinforcement Learning

## Introduction

Reinforcement learning is a field of machine learning that focuses on maximizing rewards while performing a certain task or navigating an environment.

Reinforcement learning should feel very familiar and intuitive to most people since it is a way of learning most similar to that of a person or animal. Sutton in his book [1] mentions as an example a child learning by interacting with its environment without any tutorage. Lacking guidance said child would have to rely on its actions to determine how the world will react to it and how it affects the child itself. This is a good example of how familiar the trial and error, learning from experience should be. However, this does not fully describe reinforcement learning.

A reinforcement learning problem would consider a certain **agent** performing **actions** that directly influence its **environment** in order to maximize the **reward** that results from that environment. The child from the aforementioned example might be an agent acting in an environment but to fully set it in a reinforcement learning framework the child would need to have a defined goal. For example, if the child – representing an agent – would desire to learn how to ride a bike on its own, the possible actions could be defined as shifting its balance or steering the wheel whilst the reward would be determined by how far the bike has travelled. For the sake of creating the model of that environment the only things we would like to consider could be the child and its bike since the model should only be wide enough to contain the problem. [2]

## Components

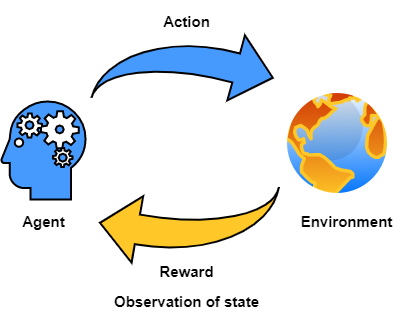


Fig. 1 An illustration of the agent-environment interaction. It shows how an agent performs actions on an environment and in turns receives a reward and an observation of the environment after that action. Then the process repeats.

**Environment** is a certain situation we want to model. It has its own set of rules describing the scope, possible actions and the reward function associated with those actions.

**State** describes a snapshot of the environment, agent, and their relation to each other in a given moment . It includes all parameters of the environment and it can change through time along with values corresponding to those parameters. For example, in the bicycle model [section 2.1] the state would consider the position of the agent and all considered physical forces acting on it.

**Observation**  of state is the information that can be accessed from the state. For the purposes of this paper, it will be assumed that . However, it has to be noted that there exist models where this is not true and applying such models directly to real-life scenarios would be difficult if not impossible. [3] In such cases the observable part of the environment can be modeled as probabilistically related to the true state. [4]

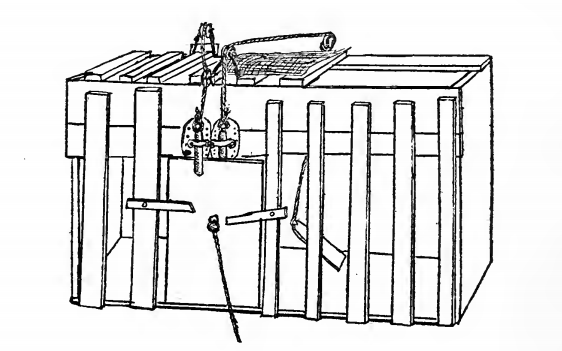
**Action**  can be performed on the environment in a given state. It may alter the state of the environment resulting in different actions being available All possible actions of an environment are called the action space and all actions available at a given state are denoted as . Reward function of an environment provides a specific reward based on the performed action.

We define an **Agent** as an entity capable of performing actions that influence the environment or its position. Each step it is performing an action based on a policy which is an instruction that defines the Agents behavior in a given state. Reinforcement learning aims to provide a policy that maximizes the sum of all rewards after arriving at a terminal state.

## History

Reinforcement learning as a subject is a convergence of two distinctive and -at the first glance – unrelated fields of science. Those fields being [1] [2]:

* Study of psychology of animal learning with an outstanding examples being the well-known Pavlovian conditioning experiments [5] and the research on trial and error conducted by Edward Thorndike. In years 1897-1898 Thorndike made a series of experiments on cats, dogs, and chickens that revolved around placing them inside a box with several mechanisms and providing them with discomfort in a form of hunger and feeling of confinement with the purpose of making the animal try to escape the box. What he has found was that cats and dogs that managed to escape the contraption would generally perform the same ask faster depending on which consecutive attempt it was. Reducing time spent in cage for an exemplary cat from 160 seconds on the first try to just 7 seconds on the 24th. [6]

  
Fig. 2 "Puzzle-box" used by Thorndike in experiments on cats.

* Optimal control theory. The field has been greatly influenced by research of Richard Bellman who proposed a functional equation for solving dynamic optimization problems. [7] The function later became known as Bellman equation or dynamic programming equation [1].

The two subjects start interpolating with a rise of the method of Temporal Difference Learning. [8] The procedure was unconventional because instead of updating the policy based on the difference between the predicted and actual outcome, it did so, based on the new, more accurate prediction. This came after a research paper by Sutton and Barto [9] concerning conditioned response in animals appearing before the actual stimuli, very much like the Pavlov’s dogs starting to salivate without receiving the stimulus in the form of food [5]. What those two papers have in common is how both in animal conditioning and TD, the reaction is produced because of the prediction of future reward rather than the actual reward itself. Temporal Difference learning has been later used by Watkins in his PhD thesis to introduce Q-Learning. [10]

The next prominent step in reinforcement learnings history came with popularization of Deep Learning, a type of machine learning methods utilizing artificial neural networks, which in combination with contemporary reinforcement learning methods created Deep Reinforcement Learning. This new field proved invaluable in widening the range of decision-making tasks that were previously out of reach. [11] With major modern achievements including programs outcompeting human world champions in games like Go [12] or Dota 2 [13].

## Markov Decision Processes

Reinforcement learning tasks are modeled using Markov Decision Processes. MDPs are described using the components introduced in section 2.2.. Where is the set of all possible states, is the action space, describes the reward that is associated with transition from to and is the probability of a given action changing the state from to . It needs noting that and not always definite values, but rather often – random variables. Most modeled environments are stochastic, hence the probabilities in the decision process. To expand on a previous example: if an agent controlling the bicycle would decide that the best action to take would be to pull the break, it does not necessarily mean that the bike would stop – if the road were icy, it could start skidding. [2]

For any given state and action Sutton and Barto [1] denote the following *Transition model*:   
.  
The transition model fully describes the dynamics of the environment. Markov decision processes are a natural extension of Markov chains for modelling decision making. The model therefore has to satisfy the Markov Property, meaning that outcome of each particular action is not dependent on the outcome of any of the previous actions. This can be observed in the transition model since the next state and the reward depend solely on the state and action pair *)* [14] [1] [2].

We can take an example of a Martian rover exploring the red desert. For purposes of demonstration, I will provide a simplistic model that does not consider most problems of a space rover. The rover is equipped with wheels, a battery powering the wheels, and a mechanism which charges the battery when resting. The goal of the robot is to travel as far is can, however it needs to consider its battery which depletes when it is traveling. The battery recharges when the rover rests but if the rover decides to travel with a low battery, it may die. We can assume that the rover can be in three states:

,

and for each state it can perform the following actions:

,,.

The dead battery state is a terminal state, meaning that no action can be taken in it.  
If the battery has a high charge and the robot decides to travel there is a probability that the battery will go into low charge. If it decides to rest, then the charge will always stay high. If the battery is already in low charge, it can be recharged on rest with the probability . However, if the rover travels with a low battery charge there is a probability that it will completely run out of energy, rendering it inoperable. The reward should be representative to the number of meters the rover has travelled. I will denote it as and it will be available whenever the rover chooses to travel. If it is resting the reward will be 0 and if the battery dies the reward will be -100. A transition table can be produced in order to present the constructed model:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| High charge | Low charge | travel |  |  |
| High charge | High charge | travel |  |  |
| High charge | Low charge | rest | 0 | 0 |
| High charge | High charge | rest | 1 | 0 |
| Low charge | High charge | rest |  | 0 |
| Low charge | Low charge | rest |  | 0 |
| Low charge | High charge | travel | 0 |  |
| Low charge | Low charge | travel |  |  |
| Low charge | Dead battery | travel |  |  |

Table 1: Transition probabilities and expected rewards for the Markov decision process of the travelling rover.

Table 1 depicts all valid combination of state , next state and possible action . For each combination there is a probability of transitioning to from given and the corresponding reward for such transition.

A good way of visualizing a Markov decision process is a transition graph. The big nodes are the possible states of the rover, I have decided to mark the terminal state in red. The small black nodes are possible actions from the action space of the state. The lines between small and big nodes are transitions, next to the transition the probability and reward are shown. It only depicts possible transitions; ones where .

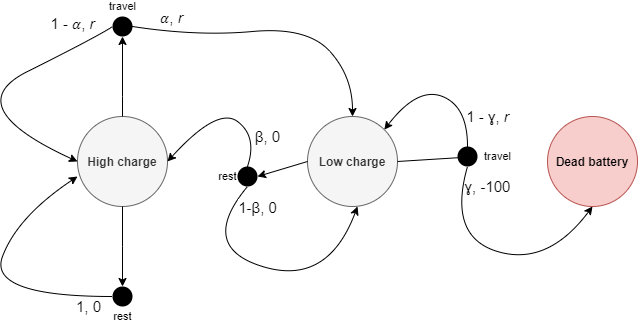


Fig. 3 Transition graph for the Markov decision process of the travelling rover

Markov decision process that also considers the observation of state is called *partially observable Markov decision process*. If action and state spaces of an environment are finite, then the process is called *finite Markov decision process.* This papers deals only with finite, fully observable Markov decision processes. [1]

## Exploration versus Exploitation

As in real life, in reinforcement learning an agent must first learn before it can successfully perform a task. This process of learning the environment is called **exploration**. If an agent is exploring the environment, then it is trying out actions to discover how they affect the environment and possible reward. It also lets the agent discover all the available actions. However, the agent will not achieve the goal of finding the optimal policy by simply stumbling around in the environment. The knowledge accumulated in the process helps the agent in the **exploitation**. In this phase the agent is acting upon its policy, exploiting what it knows about how the actions influence the environment. [1]

The challenge of balancing the exploitation and exploration has been fundamental to the optimization. [15] On one hand if the exploitation is favored without sufficient exploration, the agent will be selecting suboptimal actions without because of not discovering how beneficial the other actions may be. Alternatively, with too much time being spent on exploration the agent may not have the ability to act upon its knowledge in the short term since an explorative action can bring a negative reward.

### - Greedy policy

One of the simplest, commonly used solutions to the exploration/exploitation dilemma is stochastically exploring some of the time. Each timestep the agent would either choose the best-known action according to its policy or – with some probability – choose to perform a random action from the available action space.

A policy where the agent is only selecting the best next action is called ***greedy***, hence the name . This method has been proved to positively affect the agent’s performance [1]

Some argue that in the long run running after the agent has had time to discover all profitable actions is detrimental to the growth rate of the reward sum. Even if the agent is nearing the perfect policy, it still will be forced to choose potentially non-optimal action instead. One of the solutions would be to have decrease over time with a certain rate. This would result in less random actions the more time the agent had to study its environment. [16] [2]

# Q-learning

## Value Functions

From any state in a Markov decision process a value function can be calculated for a given policy . The value function describes the expected discounted cumulative reward an agent would obtain if it started in state and followed the policy until it arrived at a terminal state. The function can be written as follows:

Where is the discount rate – this parameter implies that the rewards received further into the future are less valuable than if they were received immediately. [1]

Similarly, we can define such a function for each state and action pair . A function which describes the expected cumulative reward starting in state , performing action and then following the policy is called the *action-value function for policy*, and is defined as [1]:

As mentioned in previous sections, reinforcement learning is concerned with finding the optimal policy for a given task. To discern which policy is better one just has to compare their value functions. If the expected discounted reward sum of a policy is greater than the expected discounted reward sum of then . A value function following the optimal policy is called the optimal value function. Similarly optimal policy will also have an optimal action-value function . We can then establish that the value function will be equal to the best action value function from that state [1] and with that and the previous equations, it can the following recursive function can be constructed:

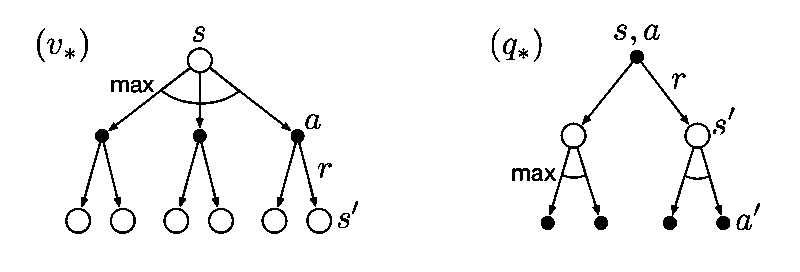


Fig. 4 Backup diagrams of the optimal value and optimal action-value functions. Backup diagrams are used to summarize algorithms and the relation of the parameters described in it. [1]

## Temporal Difference Learning

Temporal difference is in heart of most popular reinforcement learning algorithms. As briefly mentioned in section 2.3 Temporal Difference learning is what connected the two history roots of the field. The general idea behind TD is to rely on new estimates instead of waiting to discover the real value. To describe this further we will have to consider an estimate where is a set of all non-terminal states.

The general form of the update rule for an estimate is:  
 .

If we were to traverse the whole Markov decision process, for example using a Monte Carlo method we could describe the estimate updating as:

Where is the learning rate – parameter determining how much does the new information override the estimation. Please note that, as described in section 3.1, is the discounted sum of all rewards, meaning that the whole chain has to be traversed and all the rewards discovered in order to calculate its value.

In Temporal Difference Learning, instead of using the actual value of , we simply use the estimate of the next state . The formula for the simplest temporal difference method – [17]– is:  
This way the estimate can be updated after the episode ends but instead – every single step [1].

## Sarsamax

Sarsamax or Q-learning is a reinforcement learning algorithm first introduced by Christopher Watkins in his PhD thesis entitled “Learning from delayed rewards.” [10] Q-learning is a Temporal Difference Learning algorithm that utilizes the idea of following the optimal action value function from section 3.1. It follows an update rule similar to that of from section 3.2. Each step the estimate of the value function is being updated using the prediction of the value of the next step. Since the algorithm follows the optimal policy, the next chosen action will always be the one that provides the most valuable [18] [2] [1]

Where:

* – represents the learning rate, it affects the effect of the new prediction on the previous estimation.
* – represents the reward after performing the action in state .
* – is the discount factor. If its smaller than one the then rewards received later are valued less than those received earlier.
* is current, non-terminal state.
* – represents the estimate action value function for the next state with the most optimal action.

Fig. 5 Backup diagram representing the Q-learning. The alternative name, Sarsamax, was later constructed from the elements of the algorithm, visible clearly on this diagram.

## Q-table

In a standard implementation of the Q-learning algorithm a lookup table for the values is created. Such a matrix containing estimates of the action value function for all possible and pairs is called a Q-Table. [19] Values in a q-table can be initialized using 0’s to imply that there is no known value for any pair. Alternatively, it can be filled out with unnaturally high values – those are called “optimistic initial conditions” [1]. The latter method encourages exploration since actions that have already been chosen will always have smaller values than those still in the starting state.

An initial Q-Table for the Martian rover example could look like this:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 200 | 200 |
|  | 200 | 200 |
|  | 200 | 200 |

Now the Q-learning formula can be followed. We are assuming that the rover starts in a state and decides to . The reward, selected at random, was equal to 20 and the agent is now in . The Q-table can be updated using that information. We can set the learning rate to 1 and the discount factor to 0.5 for ease of calculation. We can assume that out of 2 available action from state, the rover will choose the first one, since both have an estimated value of 200.

The new estimated value for is 120, we can alter the Q-Table accordingly:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 120 | 200 |
|  | 200 | 200 |
|  | 200 | 200 |

Next time the rover is faced with a state action pair it will assume that the best policy is to .

The obvious limitation to the Q-table is that it can only hold discrete states and actions. If the process were continuous – either discretization would have to be used, or a different algorithm.

# OpenAI

## Introduction

OpenAi is an organization that focuses on AI research. It was created in December 2015 by a group of AI researchers and entrepreneurs including Elon Musk, Sam Altman, Ilya Sutskever and more. Together, the investors managed to accumulate a funding of 1 billion dollars. [20] It started as a non-profit research company trying to develop an ethical AI solutions that would serve everybody. It has since developed several products both open-source and proprietary.

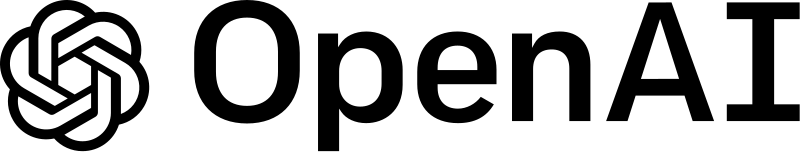


Fig. 6 OpenAI logo as of May 2021

## Mission

OpenAI’s mission is ensuring that artificial intelligence serves all of humanity. To meet this goal the organization intends to develop and help advance beneficial autonomous systems with its long-term safety in mind. The founders were concerned that the AI race may become so competitive that some could be willing to cut corners when it comes to safety precautions. They pledge that if some organization with similar views will come close to creating an artificial general intelligence before OpenAi, the company will stop competing against them and instead start supporting their project to ensure that no conflict of interest would jeopardize the systems’ safety. [21]

## History

OpenAi published its introductory post on December 11th, 2015 [20] and soon after started attracting talents from all over the world. The first project released to the public was the beta of OpenAi Gym. The intent was to help the reinforcement learning community by providing standardization and diversity of environments so that future researchers can reliably compare their results. [22]Through the years they pursued the field with several popular milestones like writing a bot that beats professional players of DOTA 2 [13], a competitive online multiplayer RTS game, or developing a novelty language model based on Transformers and unsupervised learning. The model was later succeeded by well-known GPT-2 [23] and GPT-3 [24], with the latter boasting 175 billion parameters.   
By this time, however, not all solutions created by OpenAi were open source, in March 2019 OpenAi LP was created with the parent company introducing is as follows:

*“We’ve created OpenAI LP, a new ‘capped profit’ company that allows us to rapidly increase our investments in compute and talent while including checks and balances to actualize our mission.”* [25]

Under the LP they licensed GPT-3 to over 300 companies, including Microsoft. The company assures however that nothing it does should interfere with their core mission of bringing about advanced, safe AI. [26]

# OpenAI Gym

## What is Gym?

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides a set of standardized problems which can be solved with the use of reinforcement learning. The problems, referred to as *Environments*, encapsulate a simulation with an agent. The toolkit exposes a concise interface that wraps the environment allowing for an agent to easily influence it. After each tick, the interface can provide us with the following: The current state of the environment – the representation of which varies, the latest reward obtained by the agent, information on whether it is time to end the current episode, and diagnostic information that can be used to further understand what happened during the last step. The agent can make use of the first three to then provide an action that will influence the environment in the next step. [27] [28]

## Environments

Gym boasts a considerable number of premade environments.

The envs registry can be found in *gym.envs.registry*, import gym and run the snippet below to see it.

>>> from gym import envs

>>> print(envs.registry.all())

### Algorithms

Simple computations or logic actions such as addition or reversing of symbols provided by the environment. While such a task would be trivial for a computer, the goal is to have the agent learn by examples, very much like a person would.

### Classic control and Box2D

A set of reinforcement learning problems previously proposed in literature. Notable examples include the cart pole problem(the reverse pendulum) and the mountain car example which can be found here https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.pdf

### Atari

This is an integration of the Arcade Learning Environment [29], numerous Atari 2600 game emulations can be found in this package with all offering the on-screen image as the environment’s observation. An agent performing in such an environment would be no different to a person playing the very same game on the console, that is, having visual observation as an input and pressing appropriate buttons as actions. Every environment comes in two versions, in one the observation is the RGB image of the screen but in the other – the observation of the environment is the actual RAM of the simulated Atari console. The Atari’s ram consisted of 128bytes which is a miniscule amount compared to personal computers we use today. It is still however an imaginable number if we were to imagine bit arrangements as states in a Markov decision process. These tasks are especially interesting for deep learning researchers since several of the emulated games were used in Deep mind researchers’ paper “*Playing Atari with Deep Reinforcement Learning*” [30].

A good example would be the Breakout-v0 environment. The agent controls a pallet that can move either left or right, or not move at all. The goal is to hit the incoming ball in such a way that it returns to one of the blocks on top destroying it and resulting in points. The observation is either an array of shape (210, 160, 3) – representing the on-screen image – or the 128 bytes of ram.

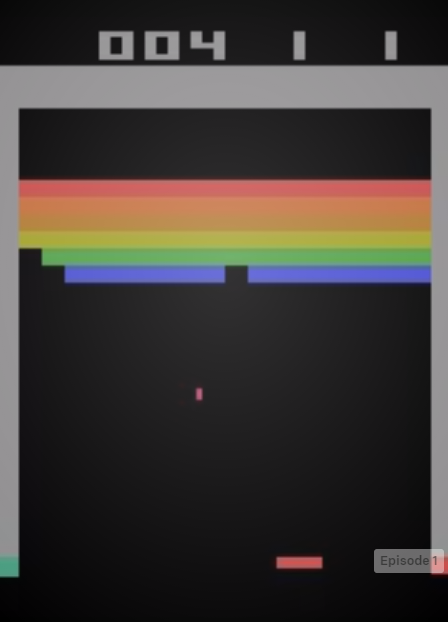
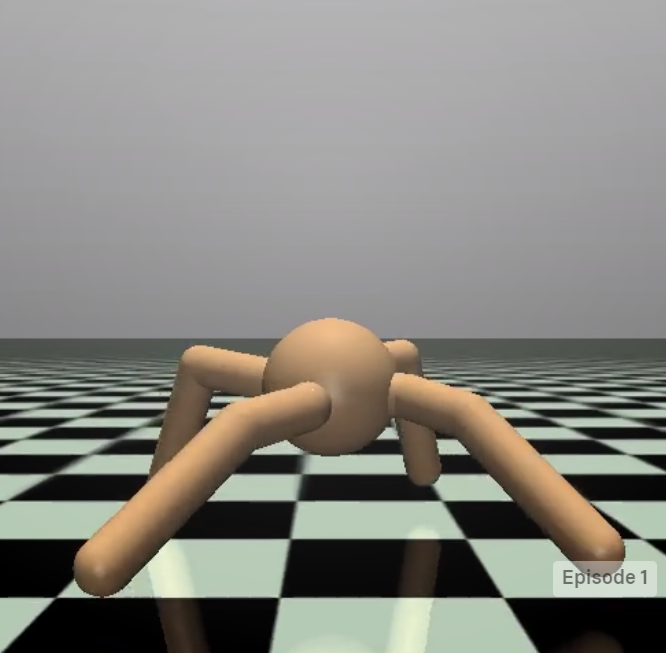


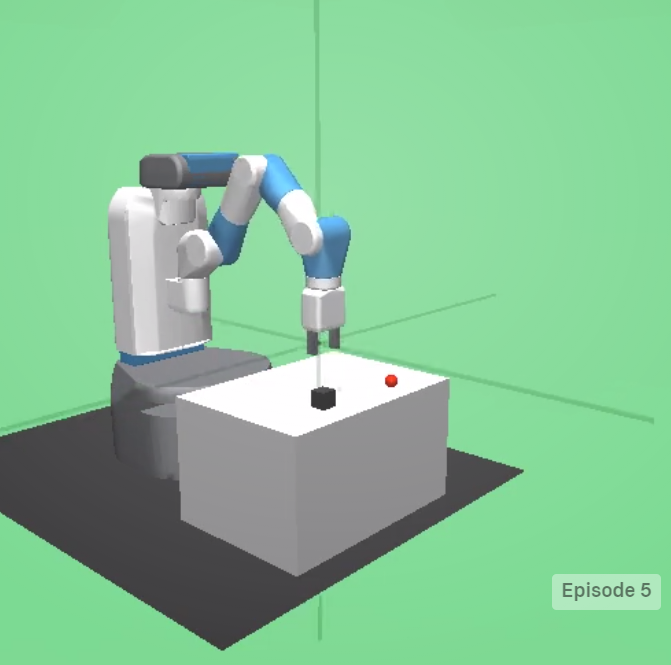
Fig. 7 Breakout-v0's on-screen image

### Mujoco and Robotics

Gym lists MuJoCo category as their environments of choice for continuous control tasks. It takes the name after a physics simulation engine created by Emo Todorov for the Movement Control laboratory of the University of Washington. MuJoCo is an acronym of Multi-Joint dynamics with Contact. The engine is now owned by Roboti LLC. [31]



The environments take advantage of MuJoCo’s fast engine to simulate agents’ limbs as they try to move with the highest velocity. The engine is also used in the Robotics category environments dedicated to, as the name suggests, robot control. The tasks under this category are goal based and include manipulating a robot arm to interact with a block or manipulating a robotic hand to position its digits or objects held within it in a desired position.

  
While the MuJoCo’s physics engine may be accurate, unlike OpenAI Gym it is not open source. In order to use the product for research a license has to be bought from the owners.**[ https://github.com/openai/mujoco-py#obtaining-the-binaries-and-license-key]** However understandable, this may come as a great hinder to many researchers who might need to explore other options.

## Interface

Gym provides a simple interface for their environment.  
Having started the environment *env*, an order to perform an action can be issued to the agent with the following method:

observation, reward, done, info = env.step(action)

The action is performed, provided that it matches the environments action space, and in turn the environment returns four variables:

* Observation: *List* - agent's observation of the current environment
* Reward: *Float* – the value that has been awarded to the agent after the performed action
* Done: *Boolean* – specifies whether the environment is ready to be reset. Returned as true in cases like the agent arriving at its final destination or failing beyond fixing.
* Info: *Dict* – diagnostic information on the environment, irrelevant from the algorithms point of view.

Gym introduces a concept of *Spaces* which describe the format of input and output of the environment, namely the actions and observation. Each environment has an action space and an observation space, invoked by: *env.action\_space* and *env.observation\_space.*OpenAI’s documentation presents the following example. https://gym.openai.com/docs/

import gym

env = gym.make('CartPole-v0')

print(env.action\_space)

#> Discrete(2)

print(env.observation\_space)

#> Box(4,)

From the snippet we can learn that environment “CartPole-v0” has two discrete actions. The values are following non-negative number so for this example they will be 0 and 1.

The observation space of the cart pole environment is a *Box(4,)*. A *Box* represents an array so in this case the observation will be a list of 4 elements.

# Unity ml-agents

## Unity game engine (or Unity Real-Time Development Platform)

Unity is a game engine created in \_ by \_

It has over 80 different case studies listed on their site (<https://unity.com/case-study>) created by companies and organizations from a range of industries including: Gaming, Engineering, Automotive, Film and more.

The team behind unity wants to empower everyone with a universal kit that will let them focus on their goal rather than on the essential tools. (https://unity3d.com/whitepapers/adopting-unity)

## Machine learning with Unity

<https://github.com/Unity-Technologies/ml-agents>

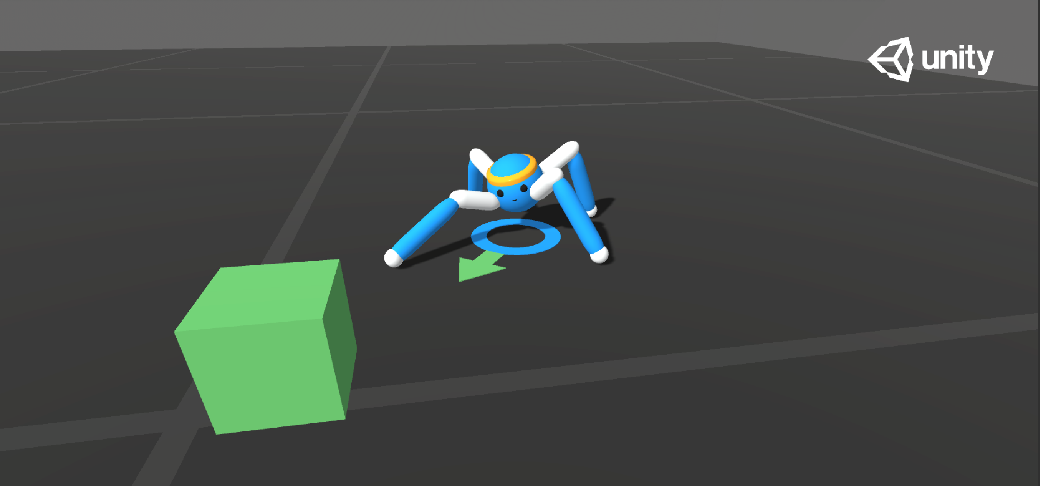
https://blogs.unity3d.com/2020/05/12/announcing-ml-agents-unity-package-v1-0/

The Unity Machine Learning Agents is a toolkit that empowers the creation of games and simulation for the purpose of using them as environments for intelligent agents. It comes with a number of sample environments prepared with sample machine learning solutions that could utilize them. Alternatively, new solutions can be tested against those either by creating policies for the agents or by utilizing a provided python API. It uses. The toolkit first appeared in Beta on Sep 19, 2017[https://github.com/Unity-Technologies/ml-agents/releases/tag/v0.1] and as of April 21, 2021 it is on its 16th stable release. The project is open source and has a growing community.

## Environment examples

The ml-agents package comes with 17 example environments each designed with a different task, reward function and proposed solution. Notable examples include:

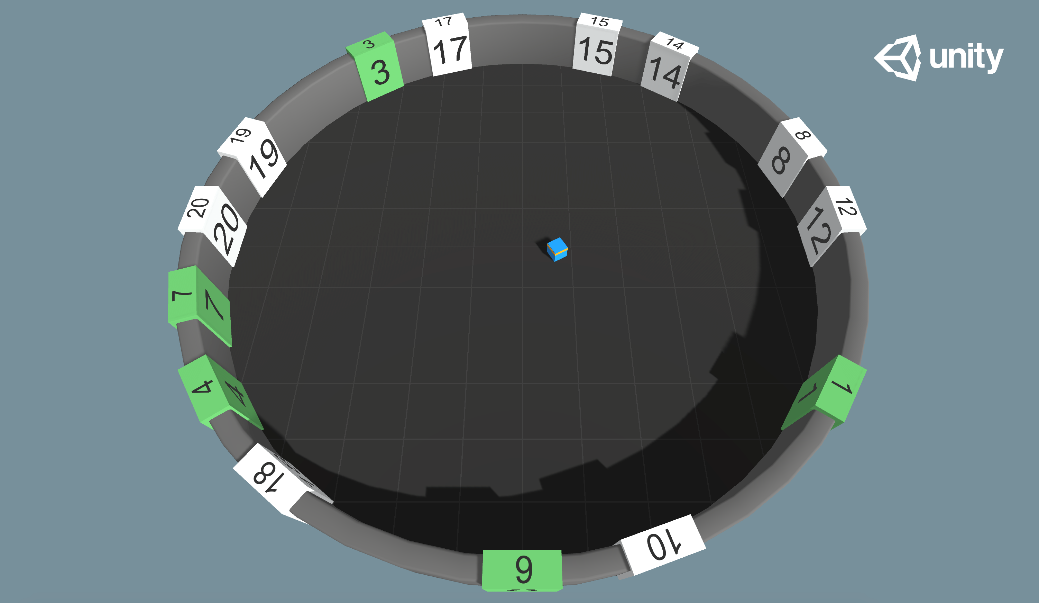
### Crawler

<https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md#crawler>  
  
This environment contains a spider-like agent with 4 sets of limbs connected by two joints. The agent has to move towards a randomly selected goal and receives rewards for a product of its velocity towards the goal and the alignment of its head.

Observation space: 172 values describing the state of agent’s acceleration, its limbs, their velocity, angular velocities etc.

Action space: 20 continuous inputs corresponding to desired rotation of the joints.

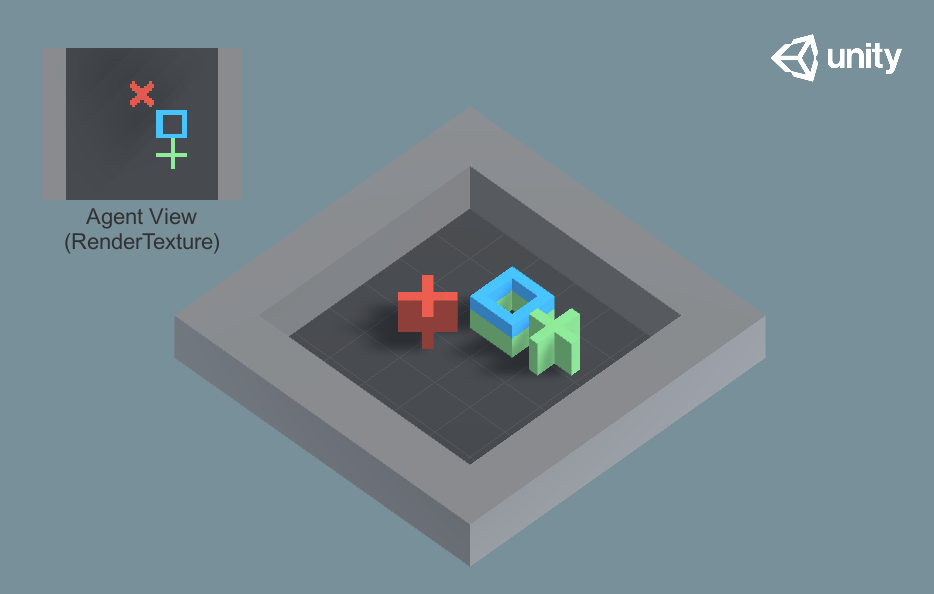
### Sorter

https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md#sorter  
   
Agent is placed in a room with 20 randomly placed numbers, the goal is for the agent to learn to touch the numbered tiles in an ascending order. The numbers placement is randomized each episode and their tiles change color when touched. The reward is received for touching the correct tile, penalty is given if an incorrect tile was touched, there is also existential penalty.

Observation space: Values describing the agent’s position, the positions of tiles, their numbers and whether they have been touched already.

Action space: Inputs specify the agent’s movement and rotation.

### GridWorld

<https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md#gridworld>  


This environment has the agent reach the goal without touching the obstacle.

Observation space: A rendering of the top-down view of the environment.

Action space: 5 possible actions representing movement in horizontal/vertical direction or not moving.

All example environments have different setups and can be solved in different ways. The examples include motion control, remembering, learning algorithms, multi-agent environments and more. Most researchers should find a benchmark environment here matching their project.  
However, the great thing about ml-agents is that the provided environments are only a part of what can be achieved with the toolkit. The documentation provides a thorough guide on creating new environments. [https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Create-New.md] This gives people the freedom to design simulations fit to their research.

## Robotic simulation

Though not included in the base ml-agents package, the team behind it has created a simulation of Universal Robotics UR3e robot. The demo of the UR3 can be checked out from Unity technologies repository. From there it can be built as a unity executable and after that it should be ready to work with ml-agents like any other environment.

The agent would have the ability to rotate 6 joints using discrete values as well as open or close the pincher. The observation would be the rotations of joints and the position of an object.  
In the provided demo the goal of the agent is to touch the object.

This simulation is provided under Apache 2.0 license meaning that. unlike MuJoCo engine, this simulation would be free to use for research purposes. [32]

# Q learning implementation using ml agents – ai gym hybrid.?

## Basic environment

I have chosen the ‘Basic’ environment provided by the ml-agents framework contains a single cube that can move left, right, or not move at all. There are two rewards in the level: one small but close and one far bigger but a little further away. The goal is to obtain the most reward state.

The agent, represented by the blue cube, start is placed in a 1-dimensional space that holds 20 positions. The agent starts at the eleventh position from the left while the small and big reward are on the positions 8 and 18, respectively.

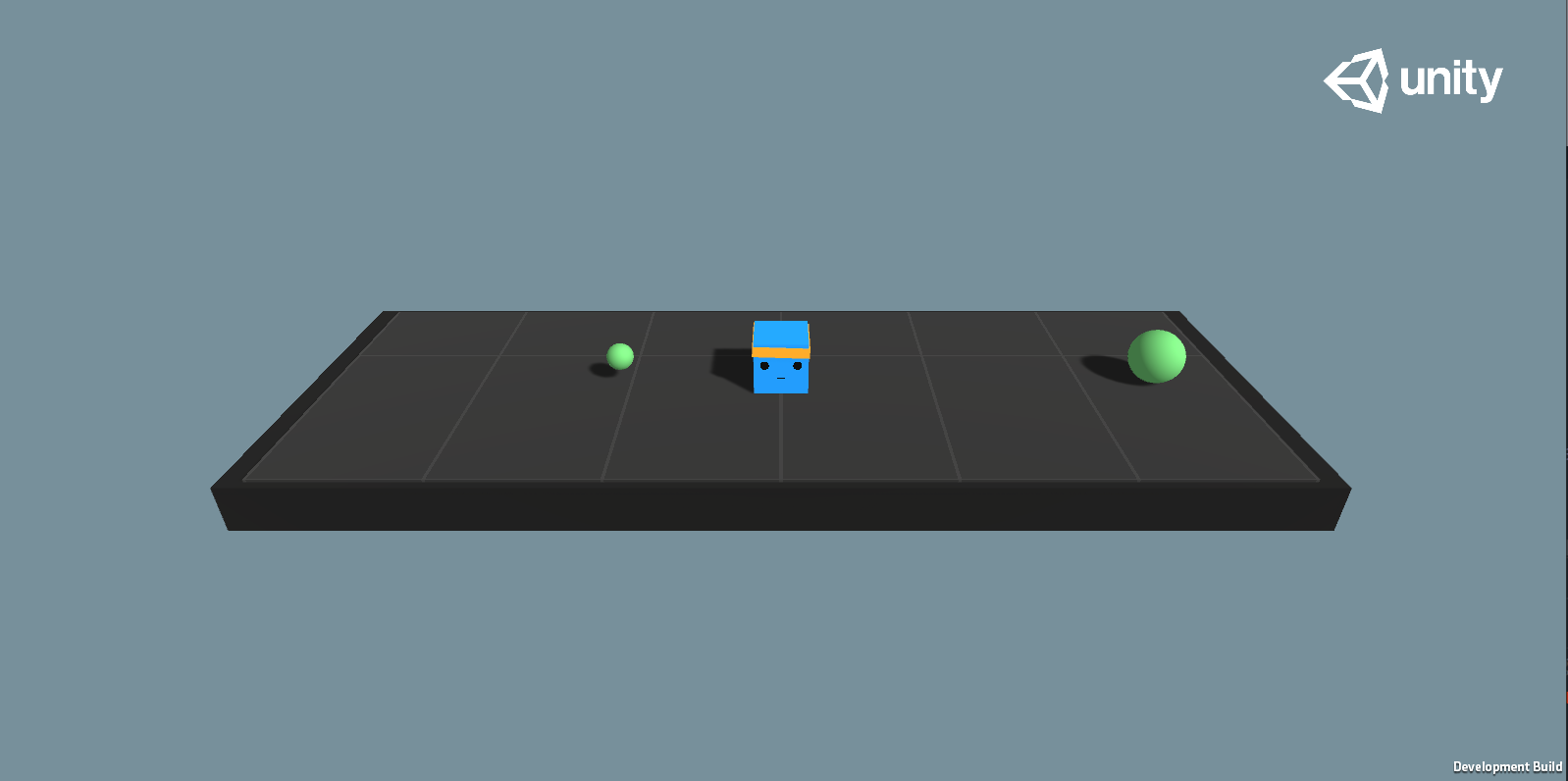


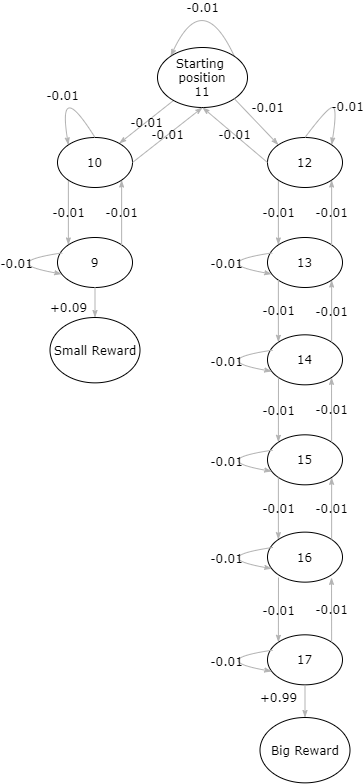
Fig. 8 View of the Basic environment in its initial state.

Reward function for the environment:

* +1.0 on touching the bigger reward.
* +0.1 on touching the smaller reward.
* -0.01 on every action.

The documentation(**citation**) lists the benchmark Mean reward as 0.93. According to this benchmark we should expect the agent to arrive at the bigger reward state after 7 steps.  
To achieve the lower reward however, the agent needs to only perform 3 steps. The path is more than 2 times shorter, but the reward is ten times smaller. This simple composition demonstrates one of the more challenging aspects of reinforcement learning – taking actions that will maximize long term rewards even if it an actions seems better in the short term. [33]

Markov decision process representation of the Basic environment

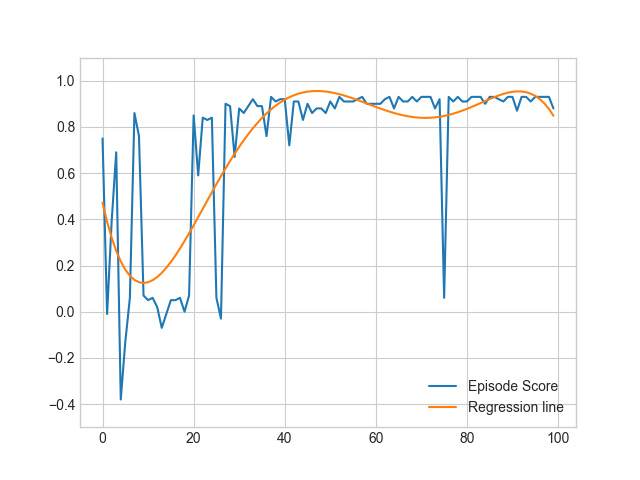


# Experiments

The original run had the following arguments:

* Learning rate = 0.99
* Discount Factor = 0.99
* Epsilon decay = 0.975 [34]

And resulted in this:



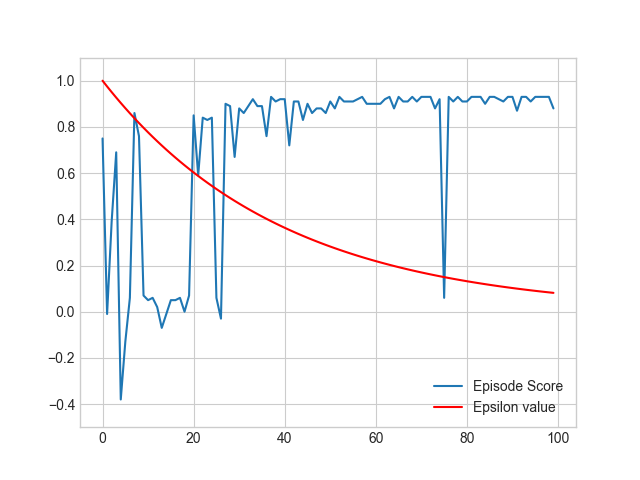


Table 2 The Q-Table after 100 episodes

|  |  |  |  |
| --- | --- | --- | --- |
|  | Do nothing | Move left | Move right |
| 1 | 0. | 0. | 0. |
| 2 | 0. | 0. | 0. |
| 3 | 0. | 0. | 0. |
| 4 | 0. | 0. | 0. |
| 5 | 0. | 0. | 0. |
| 6 | 0. | 0. | 0. |
| 7 | 0. | 0. | 0. |
| 8 | 0. | 0. | 0. |
| 9 | 0.0791 | 0.09 | 0.068309 |
| 10 | 0.83626195 | 0.0791 | 0.85481005 |
| 11 | 0.85481005 | 0.83626195 | 0.87354551 |
| 12 | 0.87354551 | 0.85481005 | 0.89247021 |
| 13 | 0.89247021 | 0.87354551 | 0.91158607 |
| 14 | 0.91158607 | 0.89247021 | 0.93089502 |
| 15 | 0.93089502 | 0.91158607 | 0.95039901 |
| 16 | 0.95039901 | 0.93089502 | 0.97010001 |
| 17 | 0.97010001 | 0.95039901 | 0.99000001 |
| 18 | 0. | 0. | 0. |
| 19 | 0. | 0. | 0. |
| 20 | 0. | 0. | 0. |

Equal distance to the rewards.

In the ‘Basic’ environment provided by Unity the agent has to learn to choose the reward even though it is twice as far as the small reward. This can lead to a situation where the agent gets stuck on the local maximum and decides to always go for the small reward.

However, if the goal were to check the very obvious assumption that the actor will always go for the bigger of two rewards then it could be more reasonable to prepare an environment that will check just that.

Unity provides us with a way to easily create new environments and alter existing ones. Using the intuitive UI, I was able to quickly alter the position of the rewards, their color, and add a description of this mode.

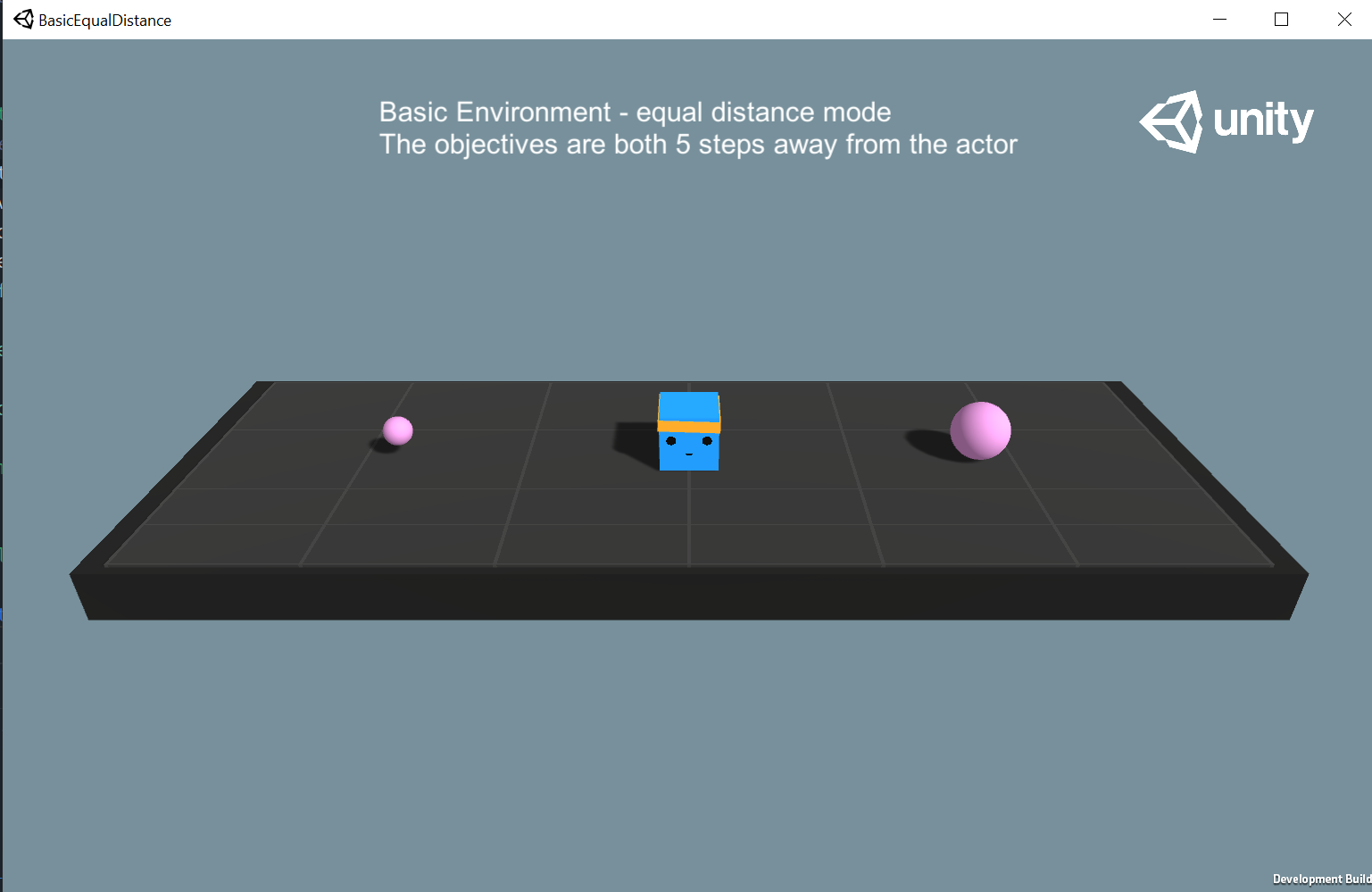
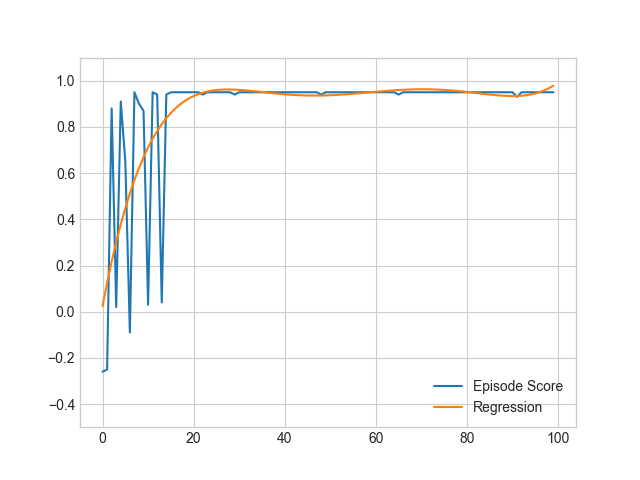


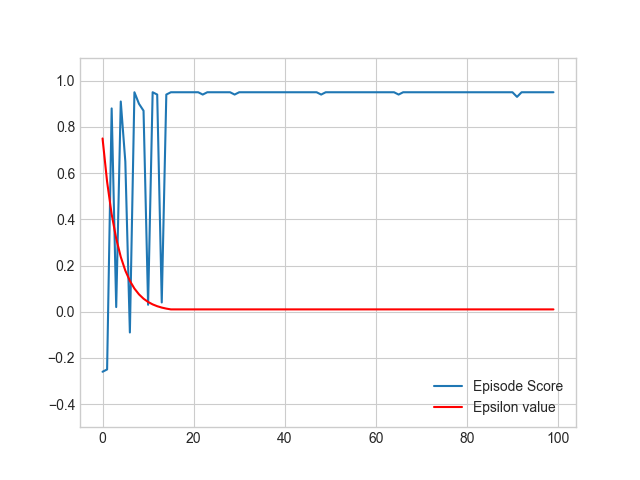
Fig. 9 The view of BasicEqualDistance environment in starting position. The rewards are at an equal distance from the agent.

After building the executable I performed the Sarsamax training with the following attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Discount factor | Learning rate | Beginning epsilon | Epsilon decay | Minimum epsilon |
| 0.95 | 0.1 | 1 | 0.75 | 0.01 |

The Q-table has been initialized with all zeros and the training run for 100 episodes.





# Conclusions

# References

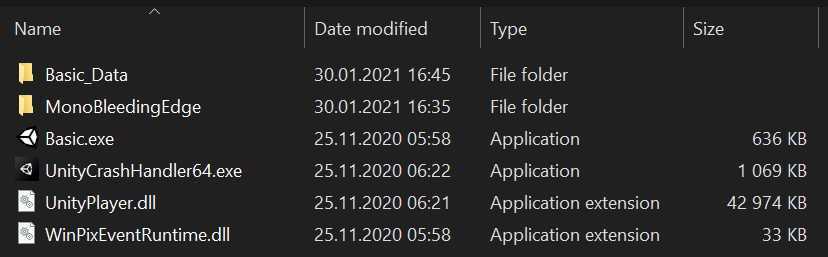
|  |  |
| --- | --- |
| [1] | R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, Massachusetts: The MIT Press, 2014, 2015. |
| [2] | P. Winder, Reinforcement Learning, O'Reilly Media, Inc., 2020. |
| [3] | K. J. Åström, "Optimal Control of Markov Processes with Incomplete State Information," *Journal of Mathematical Analysis and Applications,* vol. I, no. 10, pp. 174-205, 1965. |
| [4] | R. D. Smallwood and E. J. Sondik, "The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon.," *Operations Research,* vol. 21, no. 5, p. 1071–1088, 1973. |
| [5] | I. Pavlov, Conditioned reflexes., London: Oxford University Press, 1927. |
| [6] | E. L. Thorndike, "Animal intelligence: An experimental study of the associative processes in animals.," *The Psychological Review,* vol. II, no. 4, 1898. |
| [7] | R. E. Bellman, Dynamic Programming, Dover, 1957. |
| [8] | R. S. Sutton, "Learning to Predict by the Method of Temporal Differences," *Machine Learning,* vol. 3, pp. 9-44, 1988. |
| [9] | R. S. S. Andrew G. Barto, "Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element," *Behavioural Brain Research,* vol. 4, no. 3, pp. 221-235, 1982. |
| [10] | C. Watkins, Learning from Delayed Rewards, King's College, 1989. |
| [11] | V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare and J. Pineau, "An Introduction to Deep Reinforcement Learning," *Foundations and Trends® in Machine Learning,* vol. 11, no. 3-4, 2018. |
| [12] | D. Silver, A. Huang, C. Maddison and e. al., "Mastering the game of Go with deep neural networks and tree search," *Nature,* p. 484–489, 2016. |
| [13] | OpenAI, Dota 2 with Large Scale Deep Reinforcement Learning, 2021. |
| [14] | W. Feller, "The Exponential Density," in *An Introduction to Probability Theory and Its Applications, vol 2*, New York, John Wiley & Sons Inc., 1971, pp. 9-19. |
| [15] | M. Tokic, "Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences," *Lecture Notes in Artificial Intelligence,* pp. 203-210, 2010. |
| [16] | P. Auer, N. Cesa-Bianchi and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning,* vol. 47, no. 2-3, pp. 235-256, 2002. |
| [17] | I. H. Witten, "An adaptive optimal controller for discrete-time Markov environments," *Information and Control,* vol. 34, no. 4, pp. 286-295, 1977. |
| [18] | H. v. Hasselt, Reinforcement Learning: State-of-the-Art, Springer Science & Business Media, pp. 207-251. |
| [19] | C. Gaskett, D. Wettergreen and A. Zelinsky, "Q-Learning in Continuous State and Action Spaces," in *Advanced Topics in Artificial Intelligence*, 1999. |
| [20] | OpenAI, "Introducing OpenAI," 11 December 2015. [Online]. Available: https://openai.com/blog/introducing-open. [Accessed 25 May 2021]. |
| [21] | OpenAI, "Charter," 9 April 2018. [Online]. Available: https://openai.com/charter/. [Accessed 25 May 2021]. |
| [22] | OpenAI, "OpenAI Gym Beta," 27 April 2016. [Online]. Available: https://openai.com/blog/openai-gym-beta/. [Accessed 25 May 2021]. |
| [23] | A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, Language Models are Unsupervised Multitask Learners, 2019. |
| [24] | OpenAI, "Language Models are Few-Shot Learners," 28 May 2020. |
| [25] | OpenAI, "OpenAI LP," 11 March 2019. [Online]. Available: https://openai.com/blog/openai-lp/. |
| [26] | OpenAI, "GPT-3 Powers the Next Generation of Apps," 25 March 2021. [Online]. Available: https://openai.com/blog/gpt-3-apps/. [Accessed 25 May 2021]. |
| [27] | "OpenAi Gym Documentation," [Online]. Available: https://gym.openai.com/docs/. [Accessed 24 March 2021]. |
| [28] | "OpenAI Gym whitepaper," [Online]. Available: https://arxiv.org/abs/1606.01540. |
| [29] | M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research,* vol. 47, 2013. |
| [30] | DeepMind Technologies, "Playing Atari with Deep Reinforcement Learning," 2013. |
| [31] | "MuJoCo advanced physics simulation," Roboti LLC, 2018. [Online]. Available: http://www.mujoco.org/index.html. [Accessed 26 May 2021]. |
| [32] | Unity Technologies, "Articulations Robot Demo," 29 July 2020. [Online]. Available: https://github.com/Unity-Technologies/articulations-robot-demo/tree/mlagents. [Accessed 25 May 2021]. |
| [33] | S. Levine, "Deep Reinforcement Learning lectures," [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse-fa19/. [Accessed 21 05 2021]. |
| [34] | T. Nieuwdorp, "Dare to Discover: The Effect of the Exploration Strategy on an Agent's Performance," Radboud University Nijmegen, 2017. |
| [35] | A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar and D. Lange, *Unity: A General Platform for Intelligent Agents,* 2 ed., 2020. |

# Appendix

Write a tutorial on what u did.

## How to create a unity executable – to appendix

After building the project we receive the following directory:



## Using Ai gym with unity executable – to appendix

After creating a unity executable with our environment we’re able to import it into our runtime by creating an object of ***mlagents\_envs.environment.UnityEnvironment*** and wrapping it into a Gym with the help of ***UnityToGymWrapper*** from ***gym\_unity.envs***.

### UnityEnvironment

https://github.com/Unity-Technologies/ml-agents/tree/main/ml-agents-envs

On initiation, this object runs the unity environment executable under a provided path and establishes a connection between python runtime and the environment through an unsecured socket. By default, the connection is established on port 5005.

### UnityToGymWrapper

## Algorithm implementation – to appendix

Should I put code here or just describe it?