

**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](#_Toc72611653)

[1.1 Abstract 3](#_Toc72611654)

[1.2 Goals 3](#_Toc72611655)

[2 Reinforcement Learning 4](#_Toc72611656)

[2.1 Introduction 4](#_Toc72611657)

[2.2 Components 4](#_Toc72611658)

[2.3 History 5](#_Toc72611659)

[2.4 Decision Processes (Markov chain) 6](#_Toc72611660)

[2.5 Explore v. Exploit 6](#_Toc72611661)

[2.6 Epsiolon-Greedy policy 6](#_Toc72611662)

[3 SARSA & Q-learning 6](#_Toc72611663)

[3.1 What is SARSA 6](#_Toc72611664)

[3.2 What is Q-learning 6](#_Toc72611665)

[3.3 Q-table 7](#_Toc72611666)

[4 OpenAI 7](#_Toc72611667)

[4.1 Introduction 7](#_Toc72611668)

[4.2 Mission 7](#_Toc72611669)

[4.3 History 7](#_Toc72611670)

[5 OpenAI Gym 8](#_Toc72611671)

[5.1 What is Gym? 8](#_Toc72611672)

[5.2 Environments 8](#_Toc72611673)

[5.2.1 Mujoco 9](#_Toc72611674)

[5.3 Interface 9](#_Toc72611675)

[6 Unity ml-agents 11](#_Toc72611676)

[6.1 Unity game engine (or Unity Real-Time Development Platform) 11](#_Toc72611677)

[6.2 Machine learning with Unity 11](#_Toc72611678)

[6.3 Environment examples 11](#_Toc72611679)

[6.3.1 Crawler 11](#_Toc72611680)

[6.3.2 Sorter 12](#_Toc72611681)

[6.3.3 GridWorld 13](#_Toc72611682)

[6.4 Robotic simulation 13](#_Toc72611683)

[*7* Q learning implementation using ml agents – ai gym hybrid.? 15](#_Toc72611684)

[7.1 Basic environment 15](#_Toc72611685)

[8 Experiments 17](#_Toc72611686)

[Equal distance to the rewards. 18](#_Toc72611687)

[9 Conclusions 21](#_Toc72611688)

[10 References 21](#_Toc72611689)

[11 Appendix 22](#_Toc72611690)

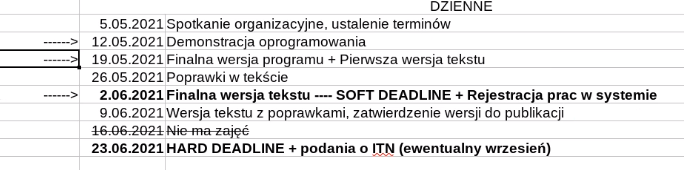
[11.1 How to create a unity executable – to appendix 22](#_Toc72611691)

[11.2 Using Ai gym with unity executable – to appendix 22](#_Toc72611692)

[11.2.1 UnityEnvironment 22](#_Toc72611693)

[11.2.2 UnityToGymWrapper 23](#_Toc72611694)

[11.3 Algorithm implementation – to appendix 23](#_Toc72611695)



# 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

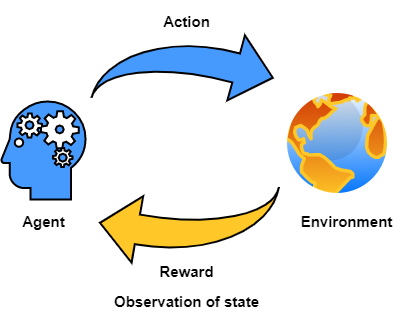


Fig. 1 An illustration of the agent-environment interaction. It shows how an agent performs actions on an environment and in turns receives a reward and an observation of the environment after that action. Then the process repeats.

**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]

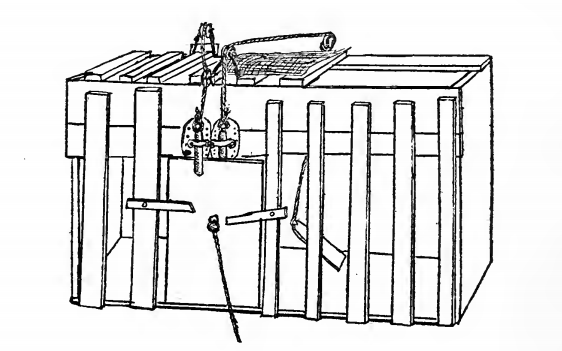
**Action**  can be performed on the environment in a given state. It may alter the state of the environment resulting in different actions being available All possible actions of an environment are called the action space and all actions available at a given state are denoted as . Reward function of 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 any 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 emphasis is put on the prediction of future reward rather than the 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 outcompeeting human world champions in games like Go [12] or Dota 2 [13].

## Decision Processes (Markov chain)

The components described in section 2.2 can be used to build a Markov decision process. A reinforcement learning task can be represented by such model The model has to satisfy the Markov Property meaning that outcome of each particular action is not dependend on the outcome of a previous action. [14] [1] This may prove difficult when modeling real-life scenarios.

## Explore v. Exploit

As in real life, in reinforcement learning an agent must first learn before it can succesfully perform a task. Th The process of learning the environment should be defined as **exploration**. It’s the time where agent can

## Epsiolon-Greedy policy

One of the simpliest solutions to the exploration/exploitation dillema is stochastically exploring some of the time.

We can take a

**Hereby, a stateaction value denotes the expected cumulative reward Rt for following π by starting in state s and selecting action a Q π (s, a) = Eπ {Rt|st = s, at = a} = Eπ (X∞ k=0 γ k rt+k+1|st = s, at = a ) , (1) where γ is a discount factor such that 0 < γ ≤ 1 for episodic learning tasks and 0 < γ < 1 for continuous learning tasks.** **http://tokic.com/www/tokicm/publikationen/papers/AdaptiveEpsilonGreedyExploration.pdf**

# SARSA & Q-learning

## What is SARSA

SARSA stands for State–action–reward–state–action, it is a reinforcement learning algorithm. Q learning is SARSA but with the assumption that the policy for updating the Q-value is based on the maximum possible reward for available actions while for SARSA itself it could be a different policy, for example taking the mean value.

## What is Q-learning

Q-learning or SARSAMAX is an algorithm relying on the same principle as SARSA but its function for choosing the best value is to choose the max value.

Definition first then after explaining the equation you can show the equation – it has to be put in context so that you can understand it on the spot. F ex New Q value Is determined by …

[15] [2]

Where:

* – represents the learning rate.
* – represents the reward.
* – is the discount factor. If its smaller than one the then rewards received later are valued exponentially(?) less than those received earlier.
* is current state.
* represents the estimate Q value after the most optimal action.

## Q-table

Q-Learning utilizes an idea of a Q-Table – a dataset assigning a so-called Q-value to each pair in a cartesian product of action and state. When an agent finds itself in a given state, with a correctly discovered values in its Q-Table, it should pick an action based on the q-value corresponding to that action. In sarsamax this will be the action which has accumulated the highest q-value.  
**Przyklady**

# OpenAI

## Introduction

OpenAi is an organization that focuses on AI research. It was created in December 2015 as 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.

## 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.

## History

OpenAi published its introductory post on December 11th, 2015 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. **https://openai.com/blog/openai-gym-beta/**Through the years they pursued the field with several popular milestones like writing a bot that beats professional players of DOTA 2, 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 and GPT-3, with the latter boasting 175 billion parameters. <https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf>. <https://arxiv.org/abs/2005.14165>  
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:

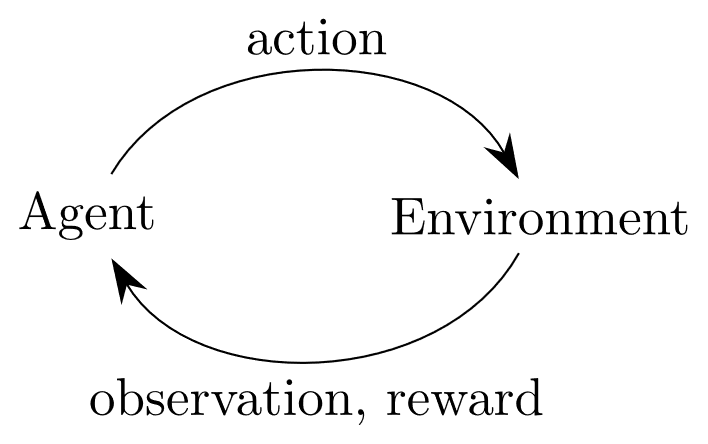
“We’ve created OpenAI LP, a new ‘capped-profit’ company that allows us to rapidly increase our investments in compute and talent while including checks and balances to actualize our mission.” [16]

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.

# 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*, encamps 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.



[17] [18]

## Environments

Gym boasts a considerable number of premade environments.

Their website lists them in the following categories:

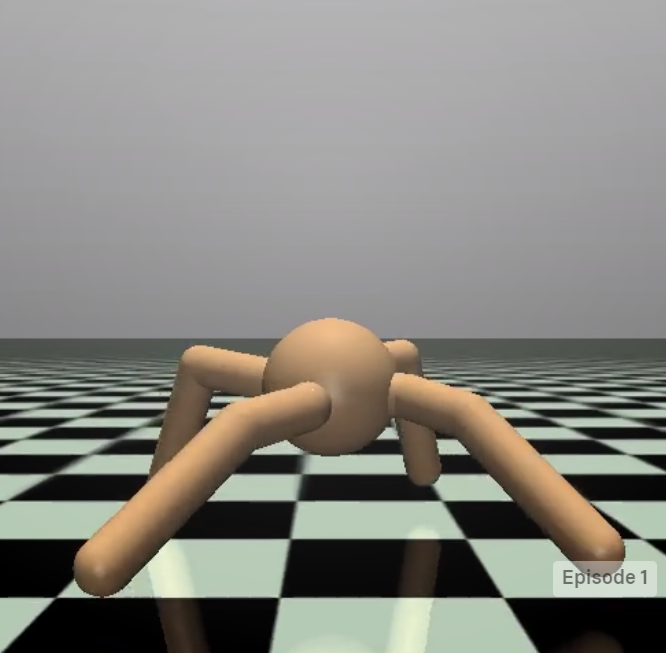
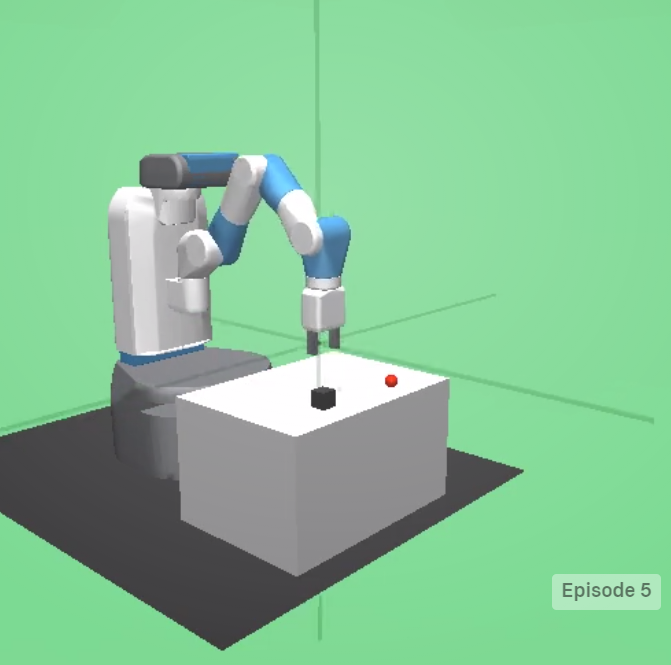
* Algorithms – Simple computations or logic actions such as addition or reversing of symbols provided by the environment. While such a task would be trivial for a computer, the goal is to have the agent learn by examples, very much like a person would.
* Box2D – 2-dimensional simulations for both discrete and continuous control tasks.
* Classic control – 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 – an integration of the Arcade Learning Environment(<https://github.com/mgbellemare/Arcade-Learning-Environment>), numerous Atari game emulations can be found in this package with most if not all offering the on screen image as the environments observation. An agent performing in such an environment would be no different to a person playing the very same game on the console- having visual observation as an input and pressing appropriate buttons as actions.
* MuJoCo – 3d simulations prepared in MuJoCo physics engine. The name stands for Multi-Joint dynamics with Contact.
* Robotics -
* Toy text

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())

### Mujoco

Gym lists MuJoCo as their environment of choice for continuous control tasks. The environment take advantage of MuJoCo’s fast physics engine to simulate agents’ limbs as they try to move with the highest velocity. The MuJoCo engine is also used in the environments dedicated to robot control.   
   
While the 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 it is learnt 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>

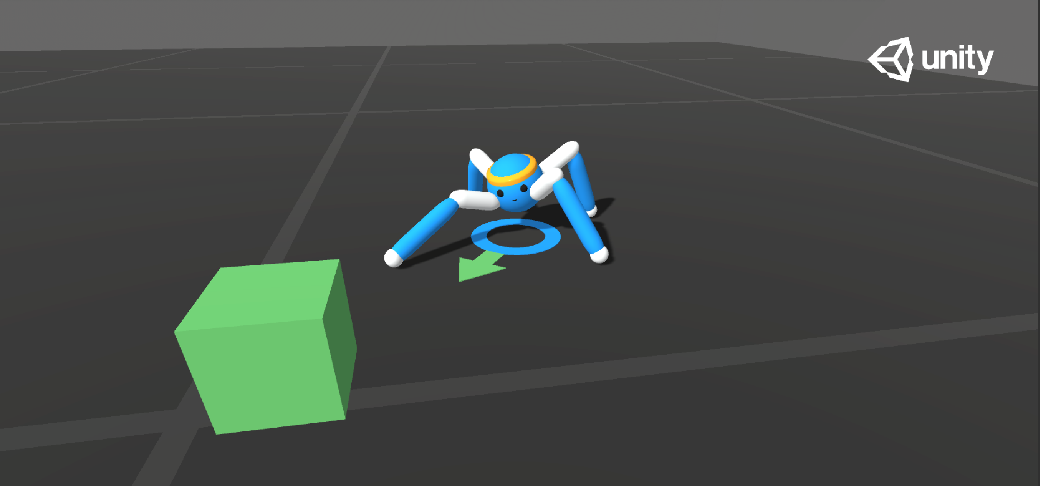
https://blogs.unity3d.com/2020/05/12/announcing-ml-agents-unity-package-v1-0/

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:

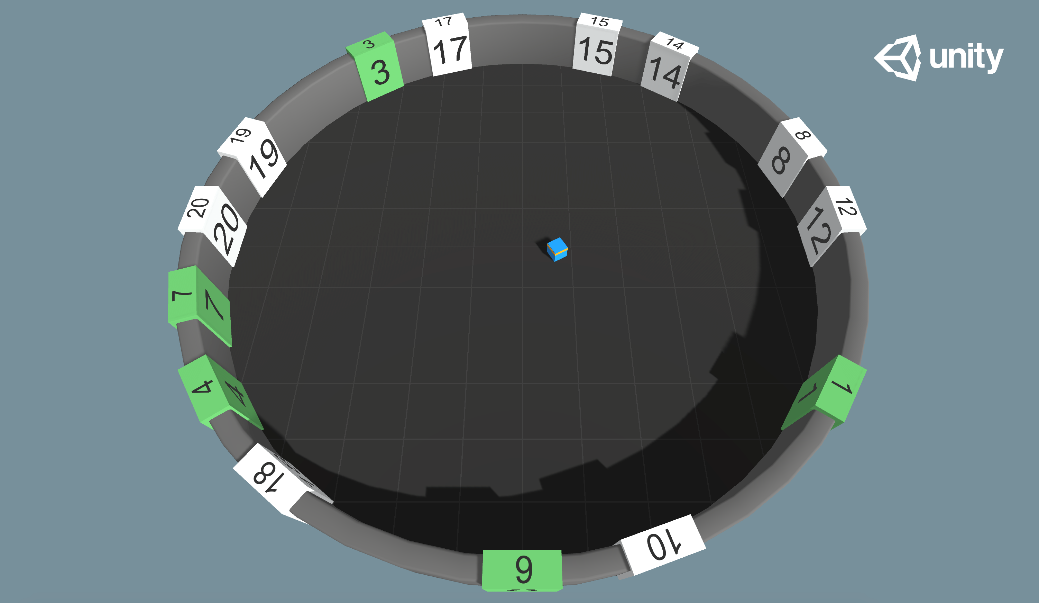
### Crawler

<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.

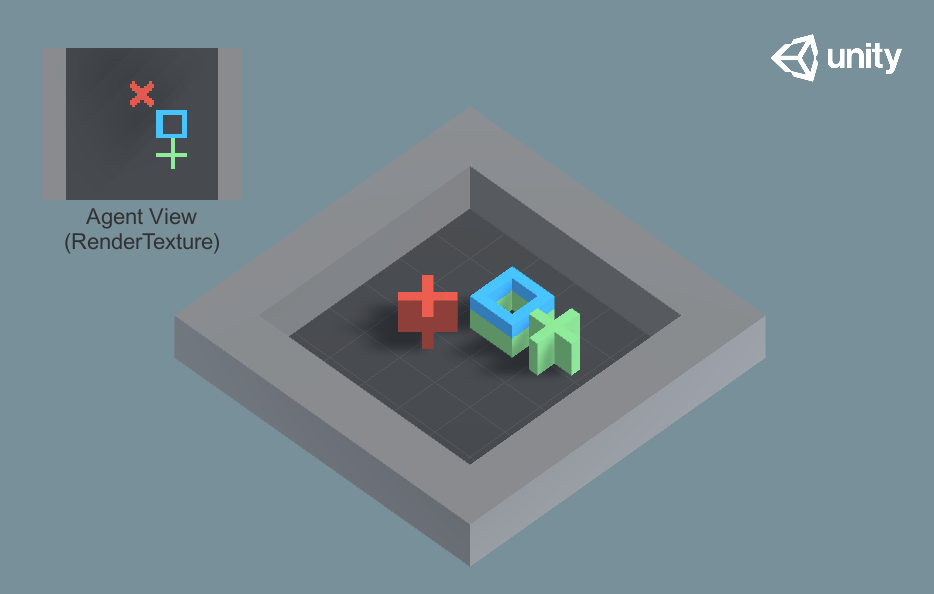
### Sorter

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.

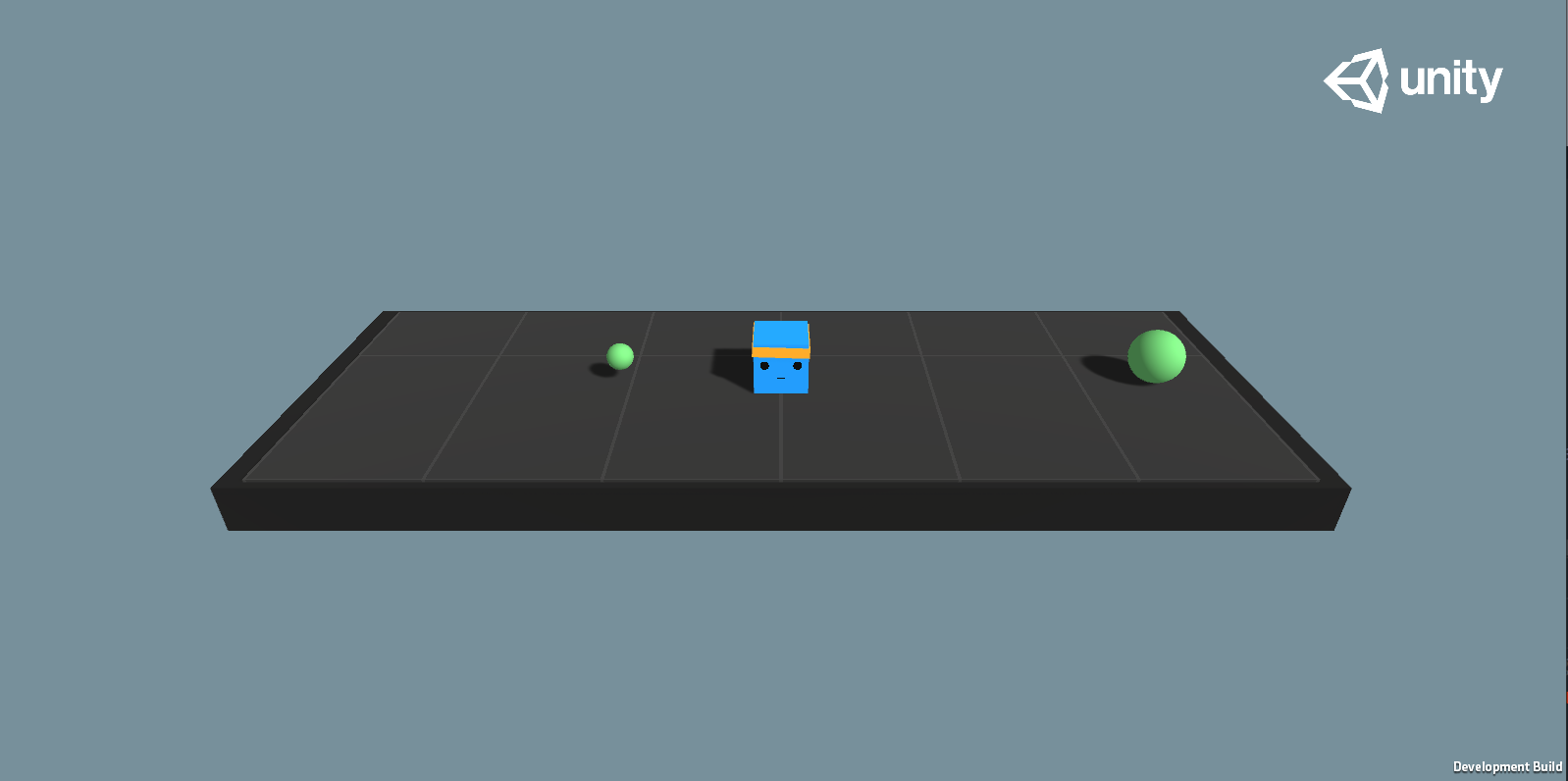
<https://github.com/Unity-Technologies/articulations-robot-demo/tree/mlagents>

# 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.



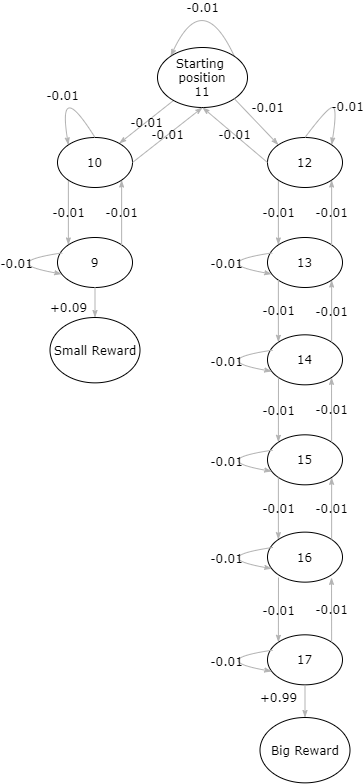
1 "Basic" environment in its initial state

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. [19]

Markov decision process representation of the Basic environment

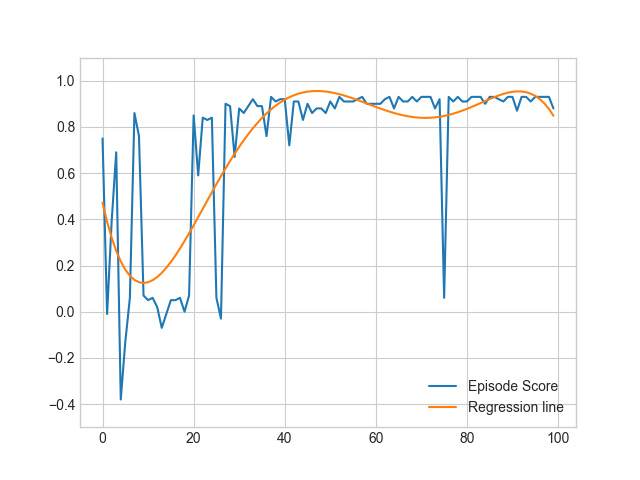


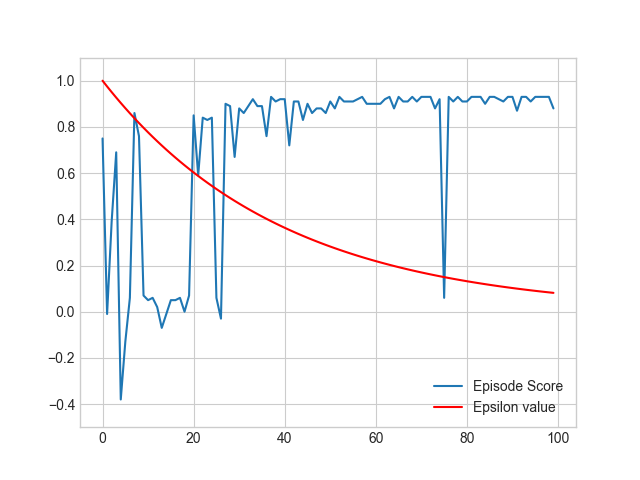
# Experiments

The original run had the following arguments:

* Learning rate = 0.99
* Discount Factor = 0.99
* Epsilon decay = 0.975

And resulted in this:





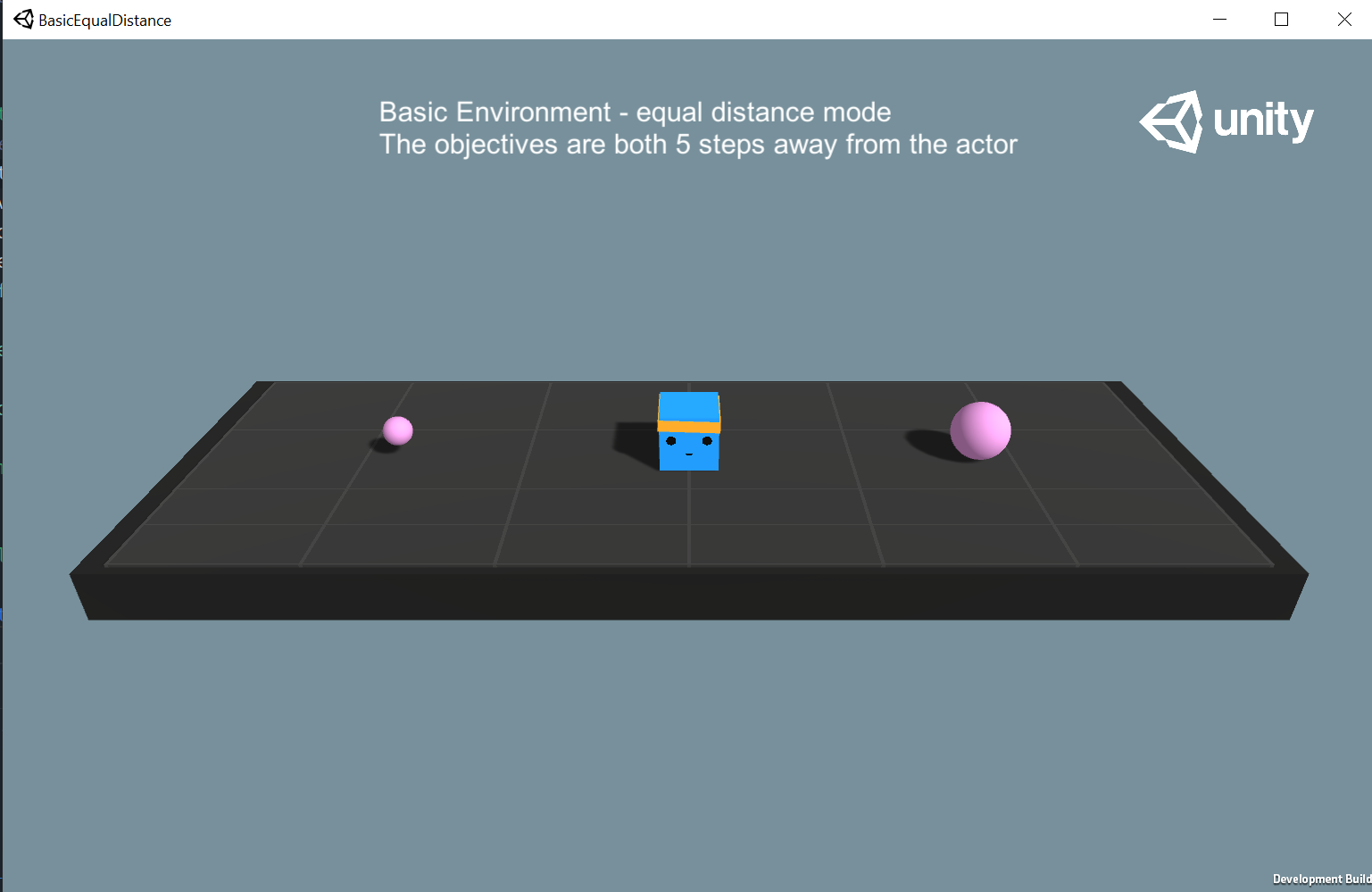
|  |  |  |
| --- | --- | --- |
| Do nothing | Move left | Move right |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 0.0791 | 0.09 | 0.068309 |
| 0.83626195 | 0.0791 | 0.85481005 |
| 0.85481005 | 0.83626195 | 0.87354551 |
| 0.87354551 | 0.85481005 | 0.89247021 |
| 0.89247021 | 0.87354551 | 0.91158607 |
| 0.91158607 | 0.89247021 | 0.93089502 |
| 0.93089502 | 0.91158607 | 0.95039901 |
| 0.95039901 | 0.93089502 | 0.97010001 |
| 0.97010001 | 0.95039901 | 0.99000001 |
| 0. | 0. | 0. |
| 0. | 0. | 0. |
| 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.

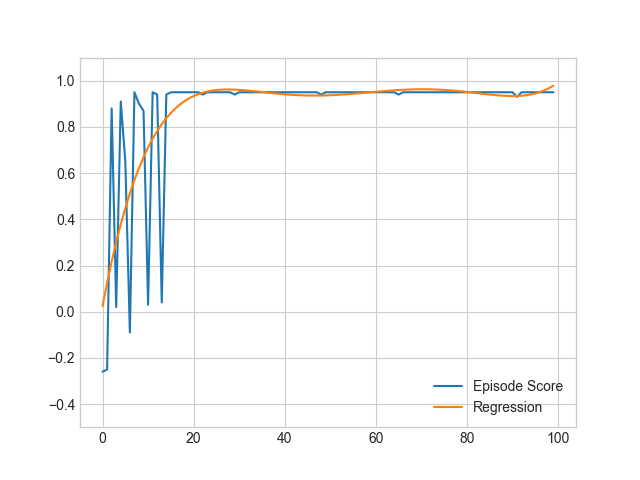
Unity provides us with a way to easily create new environments and alter existing ones. Using the intuitive UI, I was able to quickly alter the position of the rewards, their color, and add a description of this mode.

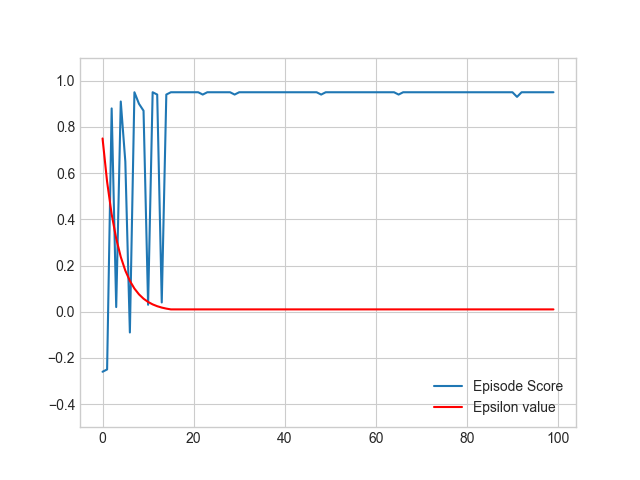


After building the executable I performed the Sarsamax training with the following attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

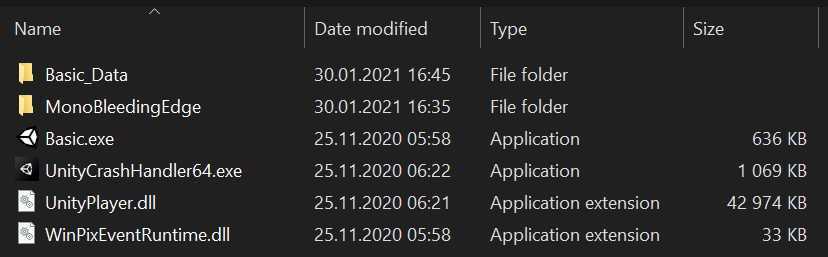
|  |  |
| --- | --- |
| [1] | R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, Massachusetts: The MIT Press, 2014, 2015. |
| [2] | P. Winder, Reinforcement Learning, O'Reilly Media, Inc., 2020. |
| [3] | K. J. Åström, "Optimal Control of Markov Processes with Incomplete 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] | H. v. Hasselt, Reinforcement Learning: State-of-the-Art, Springer Science & Business Media, pp. 207-251. |
| [16] | OpenAI, "OpenAI LP," 11 March 2019. [Online]. Available: https://openai.com/blog/openai-lp/. |
| [17] | "OpenAi Gym Documentation," [Online]. Available: https://gym.openai.com/docs/. [Accessed 24 March 2021]. |
| [18] | "OpenAI Gym whitepaper," [Online]. Available: https://arxiv.org/abs/1606.01540. |
| [19] | S. Levine, "Deep Reinforcement Learning lectures," [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse-fa19/. [Accessed 21 05 2021]. |
| [20] | 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?