

Character Animation Using Reinforcement Learning
and
Imitation Learning Algorithms

by

Tokey Tahmid

17101359

Mohammad Abu Lobabah

17101376

Muntasir Ahsan

17101011

Raisa Zarin

17301072

Sabah Shahnoor Anis

18301288

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
June 2021

© 2021. Brac University
All rights reserved.

Declaration

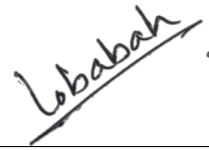
It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:



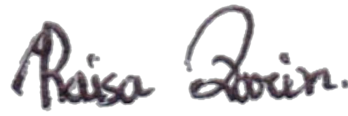
Tokey Tahmid
17101359



Mohammad Abu Lobabah
17101376



Muntasir Ahsan
17101011



Raisa Zarin
17301072



Sabah Shahnoor Anis
18301288

Approval

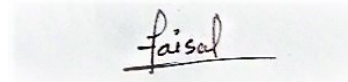
The thesis/project titled “Character Animation Using Reinforcement Learning and Imitation Learning Algorithms” submitted by

1. Tokey Tahmid (17101359)
2. Mohammad Abu Lobabah (17101376)
3. Muntasir Ahsan (17101011)
4. Raisa Zarin (17301072)
5. Sabah Shahnoor Anis (18301288)

Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on June, 2021.

Examining Committee:

Supervisor:
(Member)



Faisal Bin Ashraf
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)



Sadia Hamid Kazi
Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

Real-time character animation for gaming and film industries is challenging and achieving production-ready quality is the hardest part. Managing time and resources also plays a vital role here. Animation through marker-based motion capture is quite a tiresome process that requires costly motion-capture suits, multiple cameras, and a large amount of storage space to store all the animation. In order to make advancements in the field of animation, AI can help us manage our time and resources as well as achieve high-quality animation. In this paper, we propose a model that aims to generate real-time character animation for biped locomotion in Unity ML agents using Reinforcement learning and Imitation learning algorithms. We first evaluate the training with solely the state-of-the-art RL algorithm, PPO. Then we analyze the combination of Imitation learning algorithms BC and GAIL in conjunction with PPO. We further discuss the comparison between the two training datasets and show that our model is able to generate animations in real-time avoiding all the tedious work and large databases. We demonstrate that this approach will result in a good amount of data compression making it effortless while maintaining the quality.

Keywords: Animation; AI; Reinforcement Learning; Imitation Learning; PPO; BC; GAIL; Unity ML agents

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.

Secondly, to our supervisor Mr. Faisal Bin Ashraf sir for his kind support and advice in our work. He helped us whenever we needed help.

And finally to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

Declaration	i
Approval	ii
Abstract	iii
Acknowledgment	iv
Table of Contents	v
List of Figures	vii
List of Tables	viii
Nomenclature	ix
1 Introduction	1
1.1 Field of Character Animation	1
1.2 Aims and Objectives	2
1.3 Thesis Overview	2
2 Problem Statement	3
2.1 Scholarly Literature and Limitations	3
2.2 Our Proposal	3
2.3 Research Objectives	4
3 Literature Review	5
3.1 Character Animation and Motion Capture	5
3.2 Reinforcement Learning for Character Animation	6
3.3 Reinforcement Learning and Imitation Learning	8
4 Methodology	9
4.1 Overview	9
4.2 Algorithms	9
4.2.1 Reinforcement Learning	9
4.2.2 Proximal Policy Optimization (PPO)	11
4.2.3 Imitation Learning	12
4.2.4 Behavioral Cloning (BC)	12
4.2.5 Generative Adversarial Imitation Learning (GAIL)	12
4.3 ML Agents Toolkit	13

4.4	Reward	15
4.4.1	Extrinsic Reward Signal	15
4.4.2	Intrinsic Reward Signal	15
4.5	Training	15
4.6	Analysis	22
5	Results	24
6	Discussion	34
7	Conclusion	35
	Bibliography	37

List of Figures

4.1	Proposed System	10
4.2	Block Diagram of ML Agent Toolkit [19]	14
4.3	Extrinsic Reward Signal(I)	16
4.4	Extrinsic Reward Signal(II)	17
4.5	Intrinsic Reward Signal	18
4.6	ML Agent Installation(I)	19
4.7	ML Agent Installation(II)	19
4.8	Walker Example Environment	20
4.9	Character Model	21
4.10	Hyperparameters for Training Session(I)	21
4.11	Demonstration Data	22
4.12	Hyperparameters for Training Session(II)	23
5.1	Character Model Performing Task	24
5.2	Environment Cumulative Reward	25
5.3	Environment Episode Length	25
5.4	Losses GAIL Loss	26
5.5	Losses Policy Loss	26
5.6	Losses Pretraining Loss	27
5.7	Losses Value Loss	27
5.8	Policy Beta	28
5.9	Policy Entropy	28
5.10	Policy Epsilon	29
5.11	Policy Extrinsic Reward	29
5.12	Policy Extrinsic Value Estimate	30
5.13	Policy GAIL Expert Estimate	30
5.14	Policy GAIL Grad Mag Loss	31
5.15	Policy GAIL Policy Estimate	31
5.16	Policy GAIL Reward	32
5.17	Policy GAIL Value Estimate	32
5.18	Policy Learning Rate	33

List of Tables

5.1	Training Session(I) Results Data	24
5.2	Training Session(II) Results Data	25

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

AI Artificial Intelligence

API Application Programming Interface

BC Behavioral Cloning

CC Creative Cloud

DRL Deep Reinforcement Learning

GAIL Generative Adversarial Imitation Learning

ICM Intrinsic Curiosity Module

IL Imitation Learning

LSTM Long Short-Term Memory

MDP Markov Decision Process

ML Machine Learning

MS Microsoft

PCA Principle Component Analysis

PPO Proximal Policy Optimization

RL Reinforcement Learning

RSI Reference State Initialization

SDK Software Development Kit

TD Temporal Difference

Chapter 1

Introduction

In this ever so neoteric era of Information and Communication Technologies, where AI and ML are being used every day starting from personalized assistants to medical diagnosis, cyber security, digital devices and etc. the only progressive and sagacious move would be to adapt to and get benefited by using this approach. The proposed paper - “Character Animation Using Reinforcement Learning and Imitation Learning Algorithms” uses deep reinforcement learning and imitation learning, in here targeted towards increasing the efficiency of the ever demanding, animation industry that can be clearly comprehended from the size of animation market worldwide from 2017-2020 [16].

1.1 Field of Character Animation

Character Animation in particular is replacing other techniques such as Animate CC used for designing and publishing ads and games and also seen being used by the special effects supervisor during film-making. Animating 3D Characters using reinforcement learning will not only enhance the character quality providing a more visually aesthetic appearance but also save a lot of time and energy spent in processing as it can be performed in real-time resulting in overall efficiency of the system. With the use of neural networks, it was possible to get new and unique motion data. The neural network approaches can be divided into two learning approaches. They are supervised learning and reinforcement learning. Many researchers applied the supervised learning approach into various character animation and control projects such as locomotion, kung fu motion and sports activities as well as for motion re-targeting and interpolating keyframe postures[13]. There is a risk that these poses might converge into average poses because of ambiguity. Later on, Holden et al. introduced a special architecture. It was called the Phase-Functioned Neural Networks[13] for producing sharp movements. In recent times, there has been a surge in work using deep RL (DRL) to train models for simulated character animation[12]. We further explore in this direction for character animation using deep RL (DRL) and IL.

1.2 Aims and Objectives

Our aim is to generate real time character animation using ML agent's toolkit with Reinforcement learning and Imitation learning algorithms. Our system will be cost efficient, easy to implement, and it does not require huge datasets. The importance of this research is both in its practical application i.e. generating an animated character using ML Agent tool kit, as well as, in advancing scholarly understanding of the topic through experimental results of programming using the Deep RL(PPO) and IL(BC GAIL) algorithms. As opposed to previously used marker-based motion capture techniques, this paper uses the Unity Machine Learning Agent Toolkit cutting down time, resources and cost spent in the project. This paper proposes to eliminate all of the tiresome processes used in the field of Character Animation using this system. The idea here is to minimize the processing time using robust algorithms in addition to making it affordable for widespread use by implementing it in ML-Agent toolkits.

1.3 Thesis Overview

The thesis paper is presented as follows:

Chapter 1 is the Introduction of our work where an overview of the problem and our aims and objectives are given

Chapter 2 states the Problem Statement and illustrates the Research Objectives

Chapter 3 narrates the Literature Review

Chapter 4 describes the Methodology

Chapter 5 shows the Results

Chapter 6 the analysis of the method and results are discussed

Chapter 7 lastly, concludes the paper followed by the References

Chapter 2

Problem Statement

In major animation studios and gaming industries, character animation is done using a marker-based motion capture technique that requires a performer to wear a motion-capture suit. Multiple cameras are required in the room to capture the motion of the performer. After that, the motion capture data is applied to the 3D virtual character through the rigging process. This process of character animation requires a huge amount of time and resources.

2.1 Scholarly Literature and Limitations

Ashish Shingade and Archana Ghotkar[4] proposed a markerless motion capture method using the Microsoft Kinect for character animation. This method, however, faced many problems. By using a single depth camera on the Microsoft Kinect, they saw the accuracy dropping for recognition which limits the usability of applications when interaction with external objects is needed. Hubert P. H. Shum along with Edmond S. L. Ho and two others[3] proposed a new method. This new method could reconstruct valid movement from incomplete and noisy postures. Their work improves the quality of posture recognition. However, all of these methods of animation require a lot of resources, cost and time. This is where AI can help us to ease up the process and mitigate the need for huge resources, cost and time.

2.2 Our Proposal

We believe that the combination of RL and IL is the key to solving the character animation problem. In reinforcement learning, the model learns the animation through trial and error experience. This experience is utilized to maximize the rewards which encourage high-level behaviors making character animation easier to generate. On the other hand, with imitation learning the model learns from expert demonstrations which encourage behaviors to generate human-like animations. Going into this project, our aim is to provide a system that combines RL algorithm and IL algorithms that generates real-time character animation. We intend to make use of the Unity ML Agents Toolkit and its easy to implement features.

2.3 Research Objectives

In order to reach the aim of this project, we intend to proceed through the following steps:

1.Environment Setup: First, we need to set up our training environment in Unity ML agents.

2.Developing Model: Then we need to import and configure our 3D character model in the environment.

3.Training: We will train our model in two different training sessions to compare the performance of our system.

4.Generating Results: Lastly, we will generate results and analyze the comparison between the two result datas.

Chapter 3

Literature Review

3.1 Character Animation and Motion Capture

In order to generate character animation, motion capture data is needed. Since marker-based motion capture has many drawbacks like wearing a suit and having multiple cameras in the room, we searched for studies that worked with markerless motion capture data. International Journal of Computer Graphics & Animation (IJCGA) published a paper on Animation of 3D Human Model Using Markerless Motion Capture Applied To Sports[4].

The paper discusses a markerless motion capture method for 3D animation which uses a Microsoft Kinect to get the motion capture data. They used the MS Kinect for skeleton recognition and tracking which is essential for rigging the 3D human model. Their dataset consists of some human gestures like sprinting, jumping, waving which is captured through the MS Kinect. They used MakeHuman and Autodesk Maya to create the 3D human model. Then the character is rigged using skeleton recognition and tracking in order to apply the motion data. There was a mismatch of the size of the skeleton of motion capture data and the rigged 3D human model. In order to avoid that they needed to extract the rotations of the skeleton tracking phase. In order to do that, motion data transformation's concept was used.

They have found a good method for this using the MS Kinect. They showed how the MS Kinect has depth cameras that make detecting skeleton joints and tracking easier. Their method also takes less development and processing time. According to the authors, if the method is further developed in the future, it can be widely applied for game development and the film industry.

However, the single depth camera of Microsoft Kinect drops the accuracy of recognition which limits the usability significantly. To solve this problem, IEEE TRANSACTIONS ON CYBERNETICS published a paper on Real-Time Posture Reconstruction for Microsoft Kinect[3].

Kinect is Microsoft's motion capture system. The kinect uses an infrared sensor for obtaining the depth information. Thus it is faced with a problem that is similar to that of an optical motion capture. One of these problems are occlusions and the other one is mixing up tracked body parts. In this paper, a new method is introduced. This new method is used to measure the reliability of the tracked body parts. There are significant advantages to this new method. One of the advantages is that it depends on behavioral information. This makes it possible to apply the method in various tracking frameworks. Even with the ones that have different types

of source data. Instead of a boolean value, a degree of reliability is returned with the new method. So the parts that are neither completely right nor wrong can be made use of. Utilizing the dependability estimation, another strategy to address broken postures with a movement information base is presented. It initially consolidates the determined unwavering quality incentive as loads into a movement information base inquiry to remove a bunch of comparable postures that are kinematically substantial. At that point, utilizing these postures, a local idle space called the neural posture space with nearby Principle Component Analysis (PCA) is built. In light of the high comparability of the source information, the built regular posture space is low dimensional and smooth. At that point a space for a posture that fits well with the followed parts while fulfilling kinematic constraints is streamlined. At long last, the outcome is passed to a simulated character for additional upholding kinematic highlights, for example, segment length and the stability of movement.

The proposed framework can reproduce movement from fragmented and noisy motion information gained from the Microsoft Kinect. It is productive and accomplishes real-time execution, helping the dense client to client interaction. The constraints of the system are the computational costs, postures that are unavailable in the database and have to be recreated from the natural postures may not be a hundred percent correct, reconstruction of fast movements are also difficult causing motion blur. Also, the system can hardly handle 45 degrees of rotation which results in visibility restriction. The writers accept their structure is general and can be applied in various motion capture frameworks. Another path is to apply it for improving the conventional optical motion capture system. Thus in conclusion this framework may improvise the already existing Kinect system by some noticeable marks but not entirely.

3.2 Reinforcement Learning for Character Animation

With the advancement of AI, more specifically with neural networks, it is possible to make progress in the quality of animation. Many studies have shown that Reinforcement Learning algorithms are able to generate better realistic movements. That is why we looked into the study of Reinforcement Learning.

Reinforcement Learning: A Survey[1] was published in a Journal of Artificial Intelligence. The paper discusses the core issues of Reinforcement learning. Right off the bat, they talk about exploitation versus exploration which is characterized as a solitary state case. Here a significant distinction between supervised learning and reinforcement learning is that a reinforcement learner should expressly investigate the environment. The surroundings should be more efficiently explored because to define a problem perfectly, exploration is mandatory for the best possible outcome. One of the least difficult reinforcement learning problems is known as the k-armed bandit which has a tremendous report in the field of measurements and applied science. In this problem, we get to know about the fundamental difference between exploitation and exploration. This paper also gives us an idea about delayed reward. It portrays that the agent should have the option to realize which of the actions are better dependent on the reward which can happen later on. Problems identified with delayed reinforcement learning are all displayed as "Markov decision

processes". Adding to that we get to know about the generalization case. It is needed when the state is large and it is expected that a similar state should have similar values and similar actions. So for this, it is needed to have a large discrete state space. Through generalization, we can find out about huge spaces and have the option to have the capacity of learned data and transfer of information between "comparative" states and action. Last they express that another procedure of managing huge state spaces is to regard them as a hierarchy of learning problems. The hierarchical solution presents a minor sub-optimality in execution. However, in the case of execution time, learning time, and space it is very much efficient. In the case of reinforcement learning techniques, there are lots of variety but all these varieties work effectively for smaller problems. Among these very few of the techniques works for large-scale problems. This is because it is very difficult in general cases to solve arbitrary problems. To solve problems that are highly complex the tabula rasa learning technique should not be used rather bias should be for getting help in the learning process. From this paper, we can say that there is still a long way to go. Some fascinating inquiries stay for learning strategies and techniques for approximating, disintegrating, and joining bias into problems. In this way, with precise biases upheld by human developers or teachers, difficult reinforcement learning problems will eventually get addressed and can be compelling later on also. In another study, Algorithms for Reinforcement Learning [2], we get a better understanding of the Reinforcement Learning algorithms. The book gives an overview of the Reinforcement Learning algorithms. The algorithms are expanded on the basis of dynamic programming. The contents in the book are categorized into 3 sections namely Markov Decision Process, Value Prediction Problems, and Control. In the first section, they give a short overview on the Markov Decision Process. Then the basics of dynamic programming algorithms is described. They present two examples of MDPs and describe that MDPs are not finite one-dimensional state and action spaces. Next, they talk about how they calculate optimal value function. Then with the optimal value function, they determine optimal behavior for the state. After that, they describe dynamic programming algorithms which are value and policy iteration algorithms. In section 2 of the book, the problem of learning to predict values associated with states is discussed. They clarify the essential thoughts for the tabular case. $TD()$ is the principal algorithm they clarified. This can be seen as the learning process to value iteration and dynamic programming. At that point they portray the new gradient based techniques (GTD2 and TDC), which can be seen as improved forms of $TD()$. At last, the third area is added to algorithms that are created for control learning. In the rest of section 3, direct methods for estimating the optimal action-values and actor-critic methods are discussed.

The book aimed to provide a self-contained and relatively short overview of the Reinforcement Learning algorithms with updated contents. They discussed various problems and their solutions. They showed improved and better iterations of the methods as they progressed throughout the discussions. As the book was a short but precise overview of the algorithms, the authors had to make a few compromises. Due to space constraints, the book inevitably missed a large portion of the reinforcement learning literature. However, they concluded with links to various resources of the field for further exploration.

Now that we have gathered a bit of knowledge on Reinforcement Learning and the algorithms, we looked into some of the applications of this field for character

animation. An interesting work caught our attention among many, DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills [10]. The paper was published on ACM Transactions on Graphic Journal.

The authors showed that deep RL methods can be used to make the agent imitate a good number of reference motion clips. Their method can be applied for extremely dynamic actions as well. Keyframed motions can also be used in their method. They combined the imitation of motion objectives along with a task objective. Their method maintains the quality of motion capture methods while making it more convenient to use. Their future exploration denotes using a number of clips in the training process.

3.3 Reinforcement Learning and Imitation Learning

This study shows the use of imitation learning in conjunction with reinforcement learning. The paper Self-Imitation Learning of Locomotion Movements through Termination Curriculum [11], is the closest research to our proposed system. In their work, the authors propose a self-imitation learning system that enables the stable locomotion controllers for rapid learning. Their approach mainly combines the two recent works of continuous control which are FDI-MCTS and DeepMimic. However, their work aims to overcome the limitations of these two methods. These limitations include high run-time cost of the FDI-MCTS method and data dependency and sample complexity of the DeepMimic method. In their proposed system, they first generate reference motion using an online tree search method which allows them to use reinforcement learning with RSI to find a neural network controller in order to imitate the reference motion. The following description will help us to understand the research of imitation learning. In today's modern era Imitation learning approach aims to imitate human interaction for a certain work. Imitation learning uses observations and actions from a human demonstration to learn a policy. About Imitation learning we can say that the idea to teach through it has been throughout the world for a long time. However, the field is gaining more popularity in recent times because of its advances in computing and sensing adding to that the more upgrowing demand for intellect implementation. The pattern for seeking through imitation learning is achieving more acceptance due to it becoming easier to learn complex tasks without having that much explicit knowledge about the tasks [6]. In "Imitation learning" an agent learns from demonstrations of specific actions, which are usually done by a human (Expert) and then proceeds to imitate the actions. Imitation learning aims to master a policy that results in a desired trajectory. Here the desired trajectory are basically an expert's demonstrations [9].

Chapter 4

Methodology

4.1 Overview

Our system uses the Unity ML Agents Toolkit for training an agent in a simulated environment. We were able to train our model using reinforcement learning and imitation learning with the help of a simple-to-use Python API. The ML agents workflow can be divided into three steps. Creating the environment, training the neural network and then embedding the neural network in a different environment[8]. At first, we created the environment and the character model with the help of the walker example environment provided by the Unity ML agents. In ML agents, the environment consists of an academy, brain and agents. The academy is what lets us use other training networks using Unity’s python API[8]. The training was done in two different sessions. The first one was done with the state of the art reinforcement learning algorithm, PPO. In the second training session we used imitation learning algorithms(GAIL and BC) in conjunction with PPO. In order to better visualize the results in graphs, we used Tensorboard. After that, we analyzed the results of the two training runs. Our proposed system is shown in Figure 4.1.

4.2 Algorithms

With the use of ML agents toolkit we have access to two state of the art Reinforcement Learning algorithms and two Imitation Learning algorithms. After much research we have decided to use the Reinforcement Learning algorithm, PPO in conjunction with the two Imitation Learning algorithms, GAIL and BC. In the next sections, we further explain our reasoning for choosing these algorithms and how they work.

4.2.1 Reinforcement Learning

Reinforcement Learning is a subdivision of Machine Learning. Through “Reinforcement Learning” an agent faces different challenges and in the process learns a variety of behavior by interacting with a vigorous environment. The main difference between Reinforcement Learning and Supervised Learning is that it gives the agent slight assessment throughout the agent’s learning process and this assessment also has an indissoluble impact afterwards. Thus through more time the agent learns to perform better. With Reinforcement learning more coherent algorithms can be

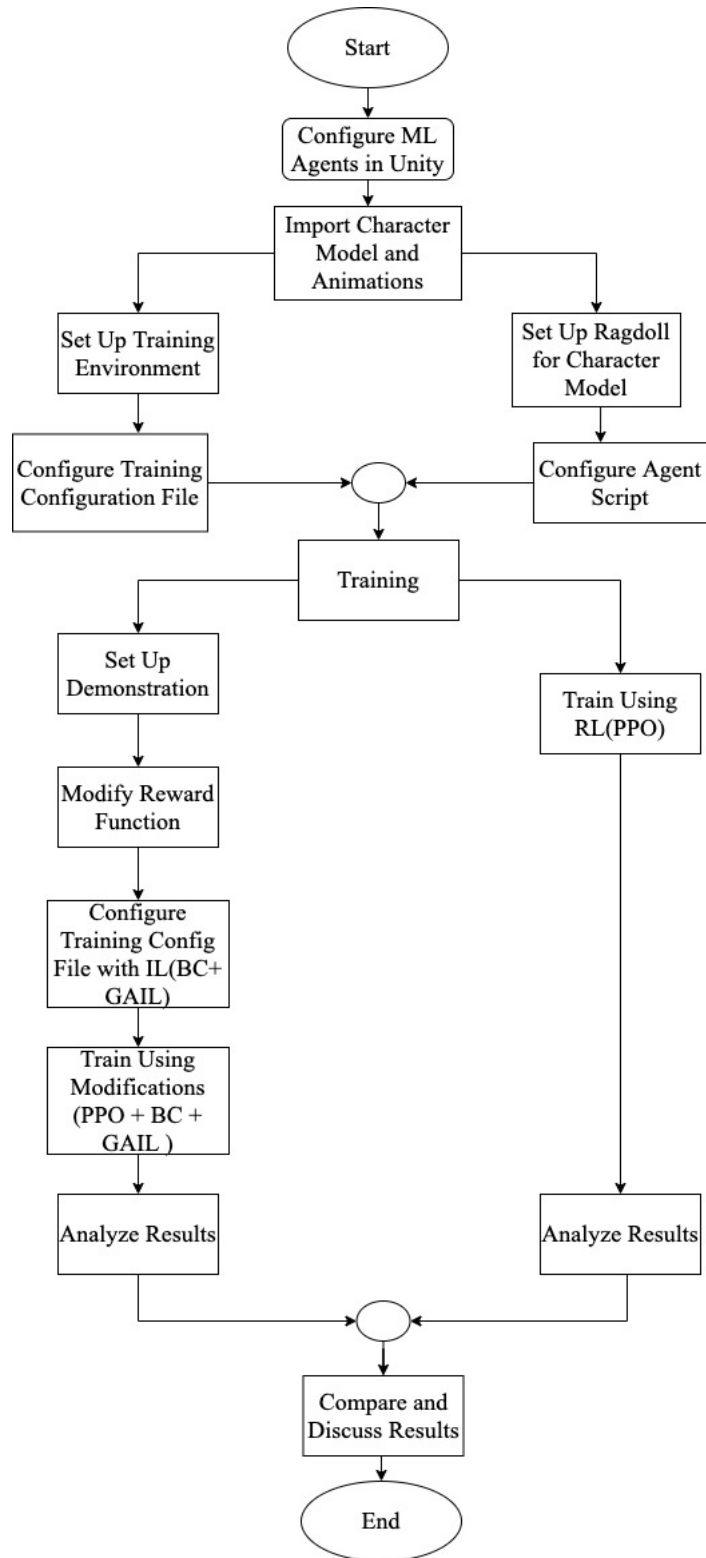


Figure 4.1: Proposed System

built which can be applied to various important fields. In a practical RL setting an agent in a system receives the systems state and reward that is related to its latest change in the systems environment. After that the agent computes an activity and sends it back to the system. Lastly the system makes changes in its state and the process is repeated again. In order to increase total rewards a better way to control the system is needed. In the framework of Markovian Decision Processes (MDPs) it uses dynamic programming to find an efficient value function instead of finding an efficient controller. RL algorithms are realistic versions of dynamic programming methods, they use robust function approximation methods to present value functions. This helps working with huge magnitude systems. So, in reinforcement learning, the agent discovers a policy that maximizes the reward. Among numerous RL algorithms, Proximal Policy Optimization(PPO) has proven to be more stable and more general purpose. Hence, we decided to use PPO as our RL algorithm.

4.2.2 Proximal Policy Optimization (PPO)

According to OpenAI, Proximal Policy Optimization (PPO) performs comparatively better than other state of the art algorithms[14]. Aside from this fact, one of the other reasons why we chose this algorithm is that it is much simpler to implement and tune making it ease-of use while maintaining good performance. PPO is what made it possible for us to train the agent in a challenging environment where the agent tries to move to the destination (target object). In this process it learns to walk and run. Even turn, and face toward the target. In other RL algorithms, a substantial amount of effort is put into tuning the hyperparameters to get a good result. This is where PPO makes it easy to implement. Tuning the hyperparameters couldn't be any easier than in PPO[14].

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (4.1)$$

Here, in this algorithm [7], ϵ is a hyperparameter where $\epsilon=0.2$, θ is the policy parameter, \hat{E}_t is the empirical expectation over timesteps, r_t is the ratio of the probability under the new and old policies, \hat{A}_t is the estimated advantage at time t. In order to do a Trust Region update, the objective finds a way to implement it. It is compatible with Stochastic Gradient Descent. Furthermore, by removing the KL penalty, the algorithm is simplified[14]. In various tests, the algorithm has performed quite well.

Algorithm 1 PPO, Actor-Critic Style

```

1: for iteration = 1, 2, ... do
2:   for actor = 1, 2, ..., N do
3:     Run policy  $\pi_{\theta_{old}}$  in environment for T time steps
4:     Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
5:   end for
6:   Optimize surrogate L wrt.  $\theta$ , with K epochs and minibatch size  $M \leq NT$ 
7:    $\theta_{old} \leftarrow \theta$ 
8: end for

```

It is proved that, with this PPO algorithm [7], the agent develops flexible movement policies. This helps them to make turns while it moves towards the destination[14]. This is why PPO was the perfect choice for our implementation.

4.2.3 Imitation Learning

Training an agent merely with reinforcement learning to make it learn via trial-and-error could result in non-human-like behavior. Hence, it is more intuitive to show the agent how to perform the task in a human-like-behavior. With the help of ML agents toolkit, imitation learning can be used solely or in conjunction with reinforcement learning [19]. We have researched and found out that the performance of the agent could increase drastically when imitation learning is used in conjunction with reinforcement learning. So, we have decided to use the two imitation learning algorithms Behavioral Cloning(BC) and Generative Adversarial Imitation Learning(GAIL) in conjunction with the reinforcement learning algorithm, PPO.

4.2.4 Behavioral Cloning (BC)

In imitation learning it is presumed that a set of expert dataset is obtainable. The dataset contains a set of trajectories and factors. By implementing these datasets a policy can be learned by direct mapping of the factor to trajectory. This process to form a policy can be put together by using an algorithm called BC (Behavioral Cloning) [9]. Behavioral Cloning is the most straightforward and easiest form of imitation learning. In this algorithm it gathers a set of demonstrations done by an expert, which is D . This is presented as a set of trajectory. Thus for application a policy presentation is needed. To compare the resemblance between the demonstrated action and the agent's own policy - objective function L has to be selected. Lastly the policy parameters has to be returned [9]. This algorithm [9] is considered efficient as it is capable of imitating the expert without having direct contact with the environment. It also takes a short amount of time.

Algorithm 2 Abstract of behavioral cloning

- Collect a set of trajectories demonstratd by the expert D
 - 2: Select a policy representation π_θ
Select an objective function L
 - 4: Optimize L w.r.t. the policy parameter θ using D
return optimized policy parameters θ
-

4.2.5 Generative Adversarial Imitation Learning (GAIL)

Generative Adversarial Imitation Learning algorithm (GAIL) is a fast and direct approach to drawing an analogy between generative adversarial networks imitation learning (IL) targeted towards boosting performance over imitating complex behaviors in large, high-dimensional environments [5]. In the GAIL framework, a secondary neural network called the discriminator is used which distinguishes the observations/actions between the demonstration and the agent. The discriminator rewards the agent if the observations/actions of the agent get closer to the demonstration. With each training step, the agent tries to maximize the reward and the discriminator gets better at distinguishing between agent and demonstration [19]. In this way, the agent learns a policy that performs states and actions similarly to the demonstration resulting in human-like behavior. In this regard, the IL algorithm

[5]:

$$\text{minimize}_{\pi} \psi_{GA}^*(\rho_{\pi} - \rho_{\pi E}) - \lambda H(\pi) = D_{JS}(\rho_{\pi}, \rho_{\pi E}) - \lambda H(\pi) \quad (4.2)$$

Here, this equation builds a bridge between IL generative adversarial networks resulting in training a generative model G in the presence of a discriminative classifier D. Now, the above equation’s practical version, which in fact is the GAIL algorithm [5], is:

$$E_{\pi}[\log(D(s, a))] + E_{\pi E}[\log(1 - D(s, a))] - \lambda H(\pi) \quad (4.3)$$

Algorithm 3 Generative adversarial imitation learning

- 1:
- 2: **for** *iteration* = 1, 2, ... **do**
- 3: Sample trajectories t_i π_{θ}
- 4: Update the discriminator parameters from ω_i to ω_{i+1} with the gradient

$$\hat{E}_{t_i}[\nabla_{\omega} \log(D_{\omega}(s, a))] + \hat{E}_{t_E}[\nabla_{\omega} \log(1 - D_{\omega}(s, a))] \quad (4.4)$$

- 6: Take a policy step from θ_i to θ_{i+1} , using **TRPO** rule with the cost function $\log(D_{\omega_{i+1}}(s, a))$. Specially, take a KL-constrained natural gradient step with
- 7:

$$\hat{E}_{\tau_i}[\nabla_{\theta} \log \pi_{\theta}(a|s)Q(s, a)] - \lambda \nabla_{\theta} H(\pi_{\theta})$$

(4.5)

- where $Q(\tilde{s}, \tilde{a}) = \hat{E}_{t_i}[\log(D_{\omega_{i+1}}(s, a))|s_0 = \tilde{s}, a_0 = \tilde{a}]$
- 8: **end for**
-

4.3 ML Agents Toolkit

A new module for the game engine, which is Unity ML-Agents, permits us to establish or utilise pre-made conditions to train the agents. The toolkit is totally free to use for the researchers and developers. While building the simulated environments, researchers and developers make the best use of the Unity Editor with the help of Python API. The toolkit has ML Agents SDK. It has all usefulness imperative to portray conditions inside the Unity Editor close by the middle C scripts to build a learning pipeline [8]. A set of example environments is one of the main features of the toolkit. Several algorithms are also included in the features of the toolkit. For instance, RL and IL algorithms, PPO, BC, ICM, LSTM[8]. There are three key components in the ML-Agents SDK. They are: the Brains, Agents and an Academy. The Agent component is basically the Characters that are used in a scene. An agent has several features. It can collect observations, take actions and receive rewards. The agent will collect observations with required sensors. A policy with a behavior name is a component of an agent[8].

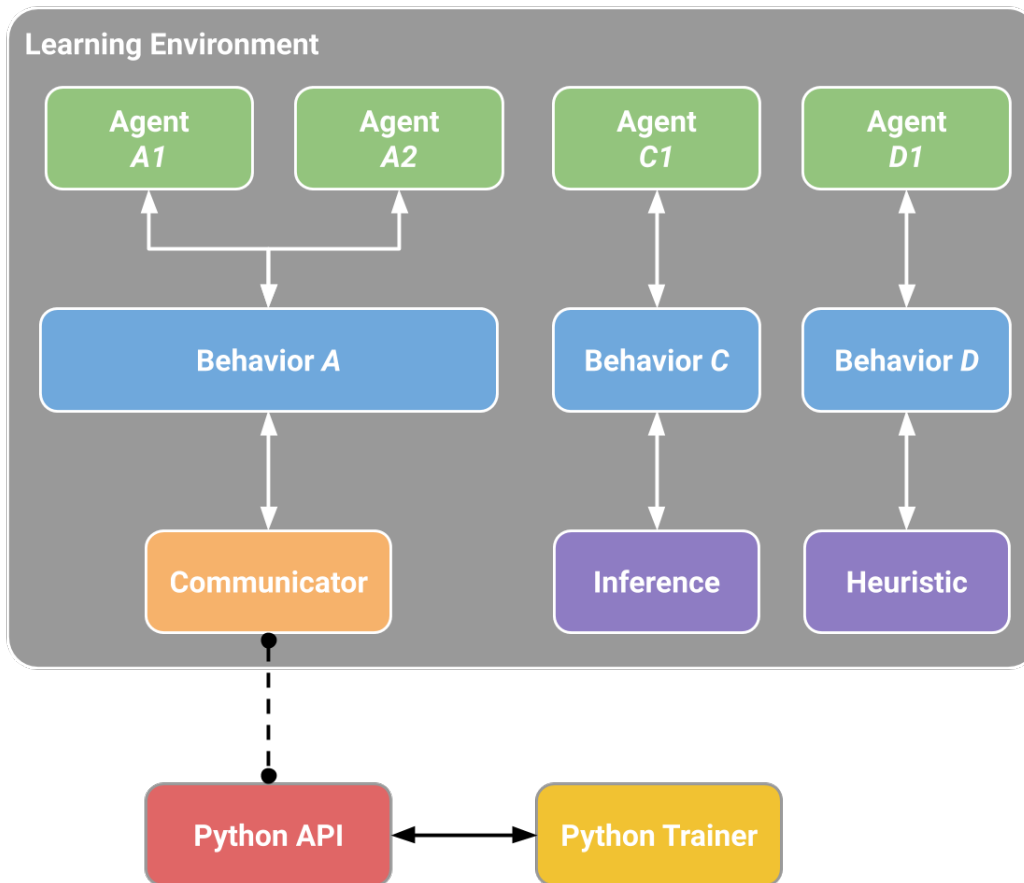


Figure 4.2: Block Diagram of ML Agent Toolkit [19]

The agents with the same behavior name will execute the same policy and share experience data during training. Not only there can be multiple agents in the same scenario, but also there can be multiple behavior names within a scene with multiple agents or single agents executing many different behavior types[8]. Block diagram of ML Agent Toolkit is shown in 4.2

The reward function is used to provide a learning signal to the agent. It can be defined or modified at any time during the simulation using the Unity scripting system. The Academy is used to keep track of the steps of the simulation and manage the agents. Using the Academy, environment parameters can be defined. That will help to change the configuration of the environment at runtime[8]. Currently, real-time character animation is done using a marker-based motion capture technique. It requires countless preparations which kills a lot of time and money. A performer needs to wear a motion-capture suit, multiple cameras in the room, special hardware and special software are required to obtain and process all the data for the real-time character animation which turns out to be a big hassle. On the other hand, using the Unity ML-Agent Toolkit and with the help of deep reinforcement learning and imitation learning algorithms, we can easily generate character motions without the hassle of motion-capture. It reduces a lot of expenses. The environment can be redefined using the ML-Agent Toolkit. It is much simpler to implement and test[8]. The Unity ML-Agents Toolkit does not only have the core functionalities. It contains a number of example environments too. These environments contain examples of single and multi-agent scenarios, with agents using either vector or

visual observations, taking either discrete or continuous actions and receiving either dense or sparse rewards[8]. For building our model, we have followed the Walker example environment. It is a physics-based biped character. The character can freely move 26 parts of it's joints. The objective of the agent is to move towards the target without falling over[8].

4.4 Reward

ML agents toolkit defines the reward signals in a modular way. It provides two types of reward signals. The extrinsic reward type represents the rewards that are defined by the environment and corresponds to reaching a goal. On the other hand, intrinsic reward signals are defined outside of the environment to encourage agent behavior in certain ways and help the learning of the extrinsic reward[19].

4.4.1 Extrinsic Reward Signal

The extrinsic reward represents the task objective reward for the agent. Here the agent receives positive rewards when it matches the target speed and looks towards the goal direction or touches the goal object and receives negative rewards for touching the ground. The reward signal is expressed by the following flow chart in 4.3 and 4.4.

4.4.2 Intrinsic Reward Signal

The intrinsic reward represents the imitation objective reward. For the intrinsic reward, we have used GAIL intrinsic reward. The GAIL reward introduces a survivor bias to the learning process with the method of Discriminator-Actor-Critic [19]. Now let us show the flow chart for the discriminator of the GAIL reward signal. Now let us show the flow chart for the discriminator of the GAIL reward signal in 4.5.

4.5 Training

In order to train our model in Unity ML agents, we first needed to set up ML agents and all its components. First, we installed ML agents in Unity. Next, we installed python version 3.7(recommended 3.6 or 3.7). We used the ml agents release 8 for our project. ML agent installaion is shown in 4.6.

4.7 shows the importing process. We followed the unity documentation[25] provided in the reference. After that, we needed to set up a virtual environment and install pytorch. To set up virtual environment following commands were used in the command line[15]:

```
'md python-envs'  
'python -m venv python-envs-env'  
'python-envs-env'  
'pip install --upgrade pip'
```

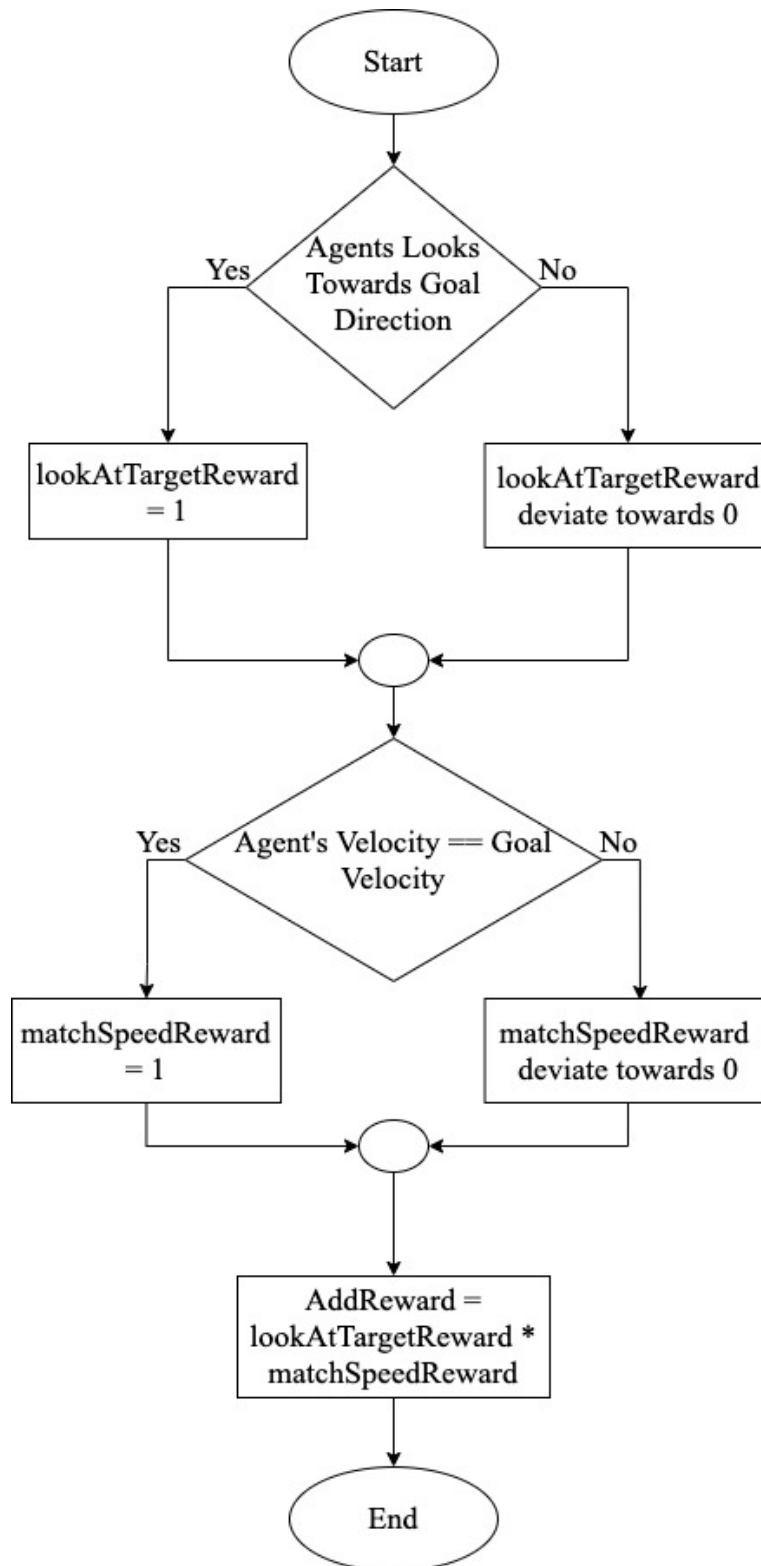


Figure 4.3: Extrinsic Reward Signal(I)

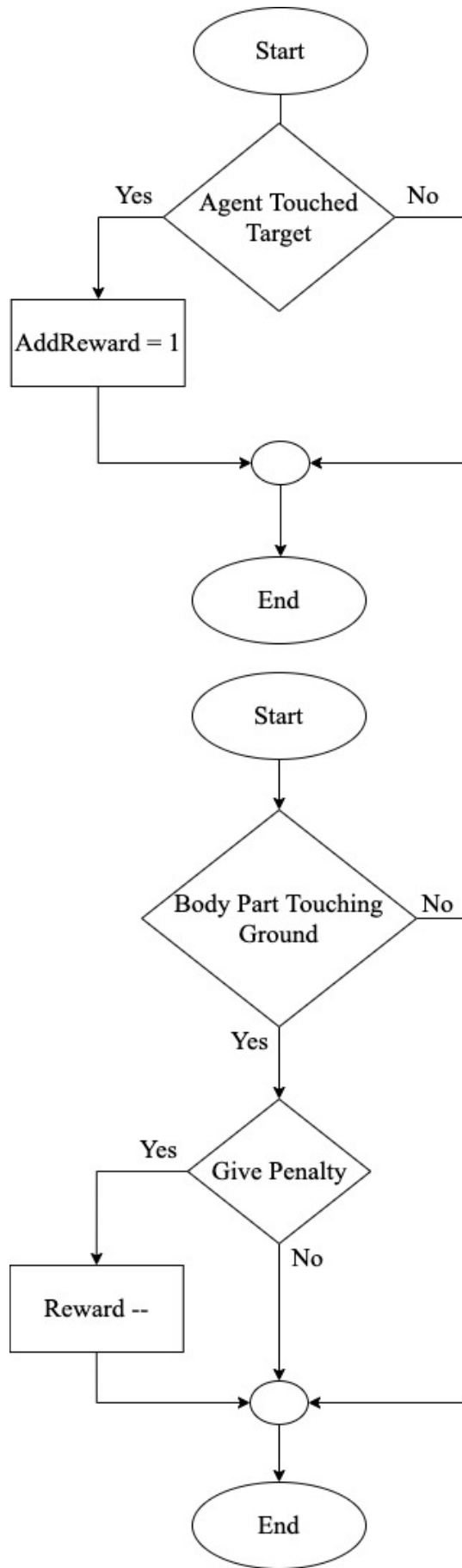


Figure 4.4: Extrinsic Reward Signal(II)

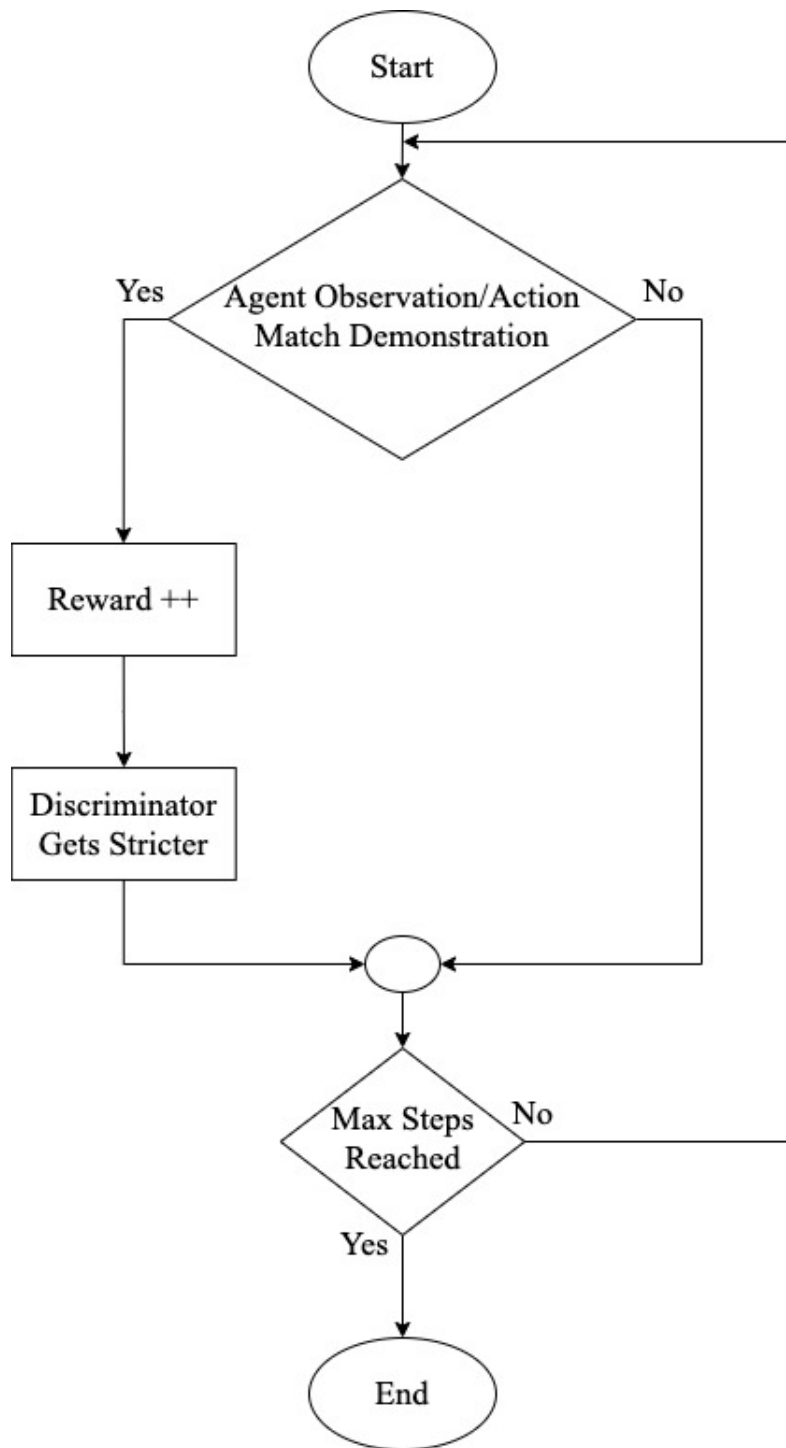


Figure 4.5: Intrinsic Reward Signal

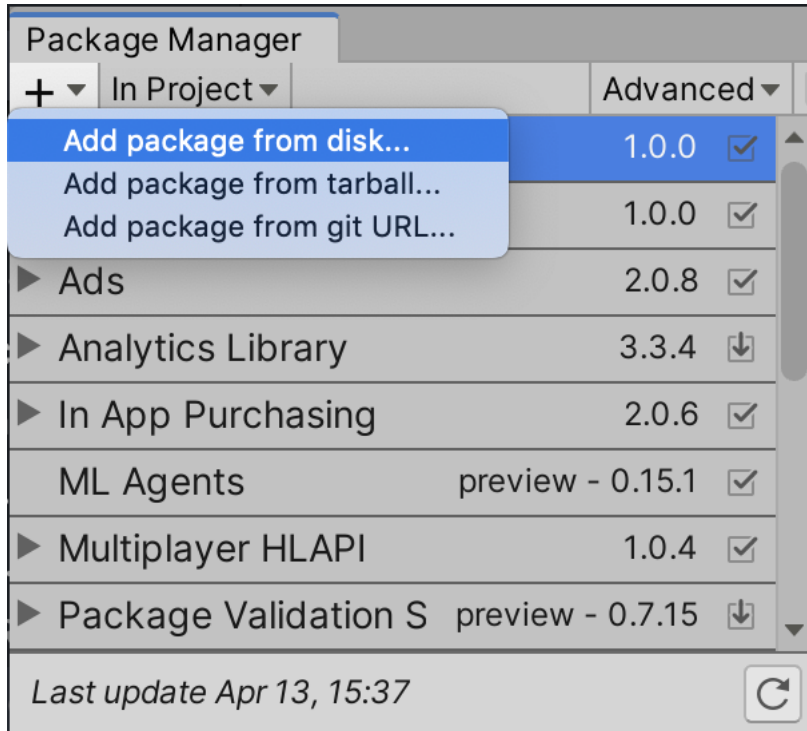


Figure 4.6: ML Agent Installation(I)

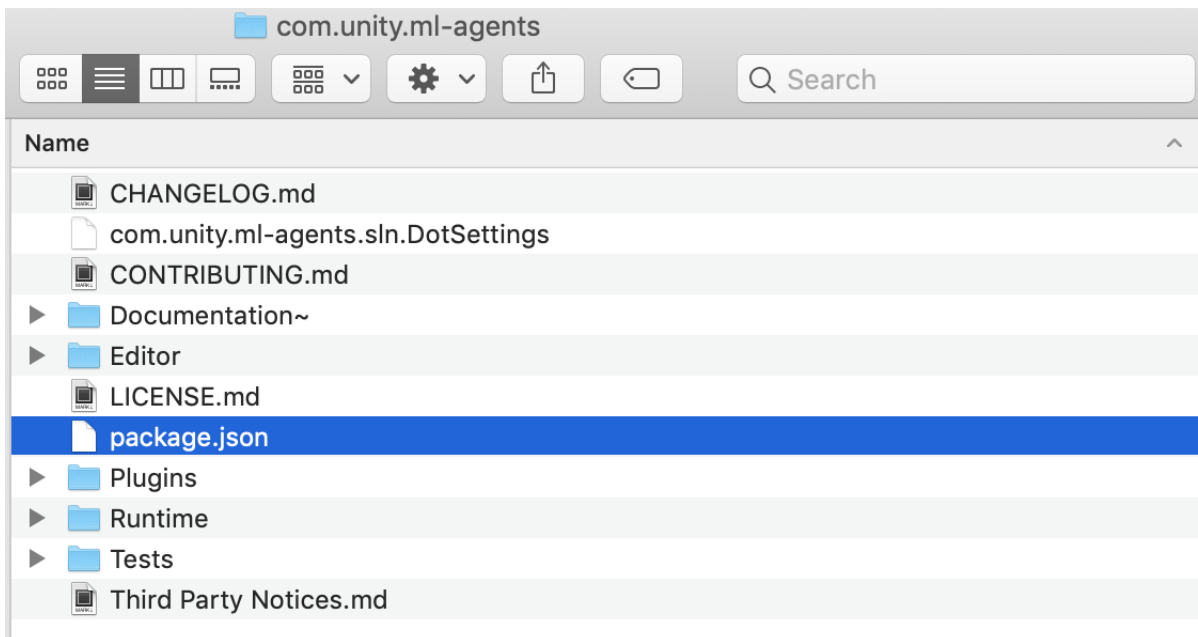


Figure 4.7: ML Agent Installation(II)

Next we installed pytorch using the following command line[17]:

```
‘pip3 install torch==1.7.0 -f’
```

```
‘https://download.pytorch.org/whl/torch_table.html’
```

And lastly we install mlagents python package to finish the installation process[17].

```
‘pip3 install mlagents’
```

After that, we tested the training process using the example environment: walker which is shown in 4.8. For this, the following command line was used in the command prompt within the virtual environment[24]:

```
‘mlagents-learn config/ppo/WalkerDynamic.yaml --run-id=firstWalkerRun’
```

In order to resume training the following command line was used[24]:

```
‘mlagents-learn config/ppo/WalkerDynamic.yaml --run-id=firstWalkerRun --resume’
```

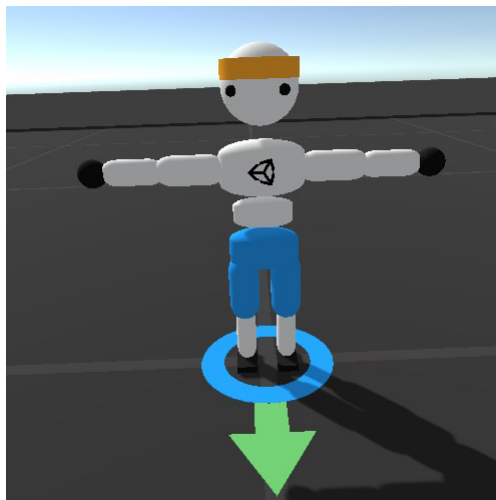


Figure 4.8: Walker Example Environment

After we finished the test run, we could now start training our own model which is shown in 4.9. Our character model was imported from Mixamo[20]. We used the Ybot and we could use any of the models as well.

With the model being imported now we needed to configure the ragdoll. The ragdoll was set up similarly to the walker example model. Then we set up the agent script and used the following training configuration file for our first training session shown in 4.10.

As we can see from the hyperparameters, only PPO(RL) was used for this training session. We trained the model up to the max steps. After we generated the results we moved on to session 2 which was training using both RL and IL. For this we first needed a demonstration for the IL algorithms. So, we set up the animator component for our model with walking animations imported from Mixamo[20]. After that, we set up the heuristic function in order to control the character movements. Then with Unity’s demonstration recorder[19] we recorded our demonstration shown in 4.11.

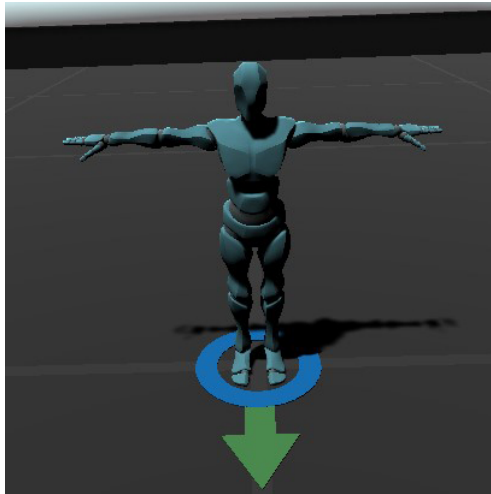


Figure 4.9: Character Model

```
1 behaviors:
2   WalkerDynamicVariableSpeed:
3     trainer_type: ppo
4     hyperparameters:
5       batch_size: 2048
6       buffer_size: 20480
7       learning_rate: 0.0003
8       beta: 0.005
9       epsilon: 0.2
10      lambda: 0.95
11      num_epoch: 3
12      learning_rate_schedule: linear
13     network_settings:
14       normalize: true
15       hidden_units: 512
16       num_layers: 3
17       vis_encode_type: simple
18     reward_signals:
19       extrinsic:
20         gamma: 0.995
21         strength: 1.0
22     keep_checkpoints: 5
23     max_steps: 30000000
24     time_horizon: 1000
25     summary_freq: 30000
26     threaded: true
27
```

Figure 4.10: Hyperparameters for Training Session(I)

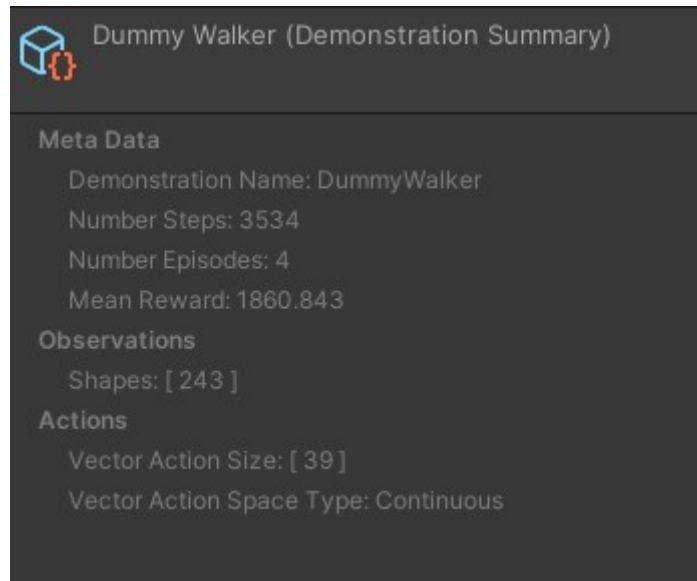


Figure 4.11: Demonstration Data

It shows the data from the demonstration which we could see in the Unity editor. Now with the demonstrations set up, we could start our training session 2 with the following training configurations.

As we can see in 4.12, IL algorithms GAIL and BC have been used for this training session. After max steps were completed we generated the results and that concluded our whole training process.

4.6 Analysis

We analyzed our training results in TensorBoard, a visualization toolkit of tensorflow. It is basically a suite of web applications for inspecting and understanding the training runs and graphs. TensorBoard is designed to run entirely offline on the local machine, without the need for the Internet[23]. TensorBoard provided us with the visualization and tooling needed for our training results. We could track and visualize the model graphs of cumulative reward, episode length, policy loss, value loss and all policy graphs[21]. First we installed tensorboard in the virtual environment with the following command line[22]:

```
'pip install tensorboard'
```

After saving the trained model, we analyzed the result graphs in tensorboard with this command[24]:

```
'tensorboard --logdir results'
```

Let us discuss these results in the following chapter.


```

1 behaviors:
2   WalkerDynamicVariableSpeed:
3     trainer_type: ppo
4     hyperparameters:
5       batch_size: 2048
6       buffer_size: 20480
7       learning_rate: 0.0003
8       beta: 0.005
9       epsilon: 0.2
10      lambda: 0.95
11      num_epoch: 3
12      learning_rate_schedule: linear
13     network_settings:
14       normalize: true
15       hidden_units: 512
16       num_layers: 3
17       vis_encode_type: simple
18     reward_signals:
19       extrinsic:
20         gamma: 0.995
21         strength: 1.0
22       gail:
23         strength: 0.1
24         gamma: 0.99
25         encoding_size: 128
26         demo_path: Demos/DummyWalker3.demo
27         learning_rate: 3.0e-4
28         use_actions: true
29         use_vail: false
30     behavioral_cloning:
31       demo_path: Demos/DummyWalker3.demo
32       strength: 0.1
33     keep_checkpoints: 5
34     max_steps: 50000000
35     time_horizon: 1000
36     summary_freq: 30000
37     threaded: true
38

```

Figure 4.12: Hyperparameters for Training Session(II)

Chapter 5

Results

In the proposed system, Unity ML-Agent toolkit has been used for the training simulation. As discussed before, there were two training sessions. So, we will be looking at two result datas to compare between them. In order to visualize all the results in graphs TensorBoard was used.

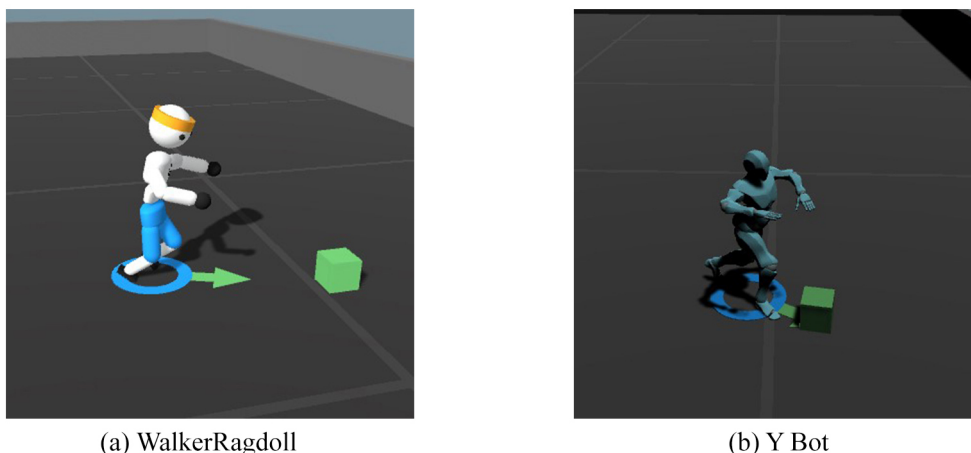


Figure 5.1: Character Model Performing Task

In 5.1, we can see the trained character model performing the task which is walking towards the goal target. The following figures show the results of our first training session where only PPO(RL) is used.

Steps	Cumulative Reward	Episode Length
15000000	395.3	148.5
30000000	488.3	169.4

Table 5.1: Training Session(I) Results Data

Here in 5.1, we can see that for max steps 30000000, the mean cumulative reward is 488.3 and episode length is 169.4. Now let us look at the results of our 2nd training session where GAIL and BC were used in conjunction with PPO.

In 5.2, we can see the training data. After 30000000 max steps the mean cumulative reward is 678.8 and episode length is 235.5. Now let us look at the difference between the two training sessions with the help of the tensorboard graphs.

Steps	Cumulative Reward	Episode Length
15000000	445.7	185.3
30000000	678.8	235.5

Table 5.2: Training Session(II) Results Data

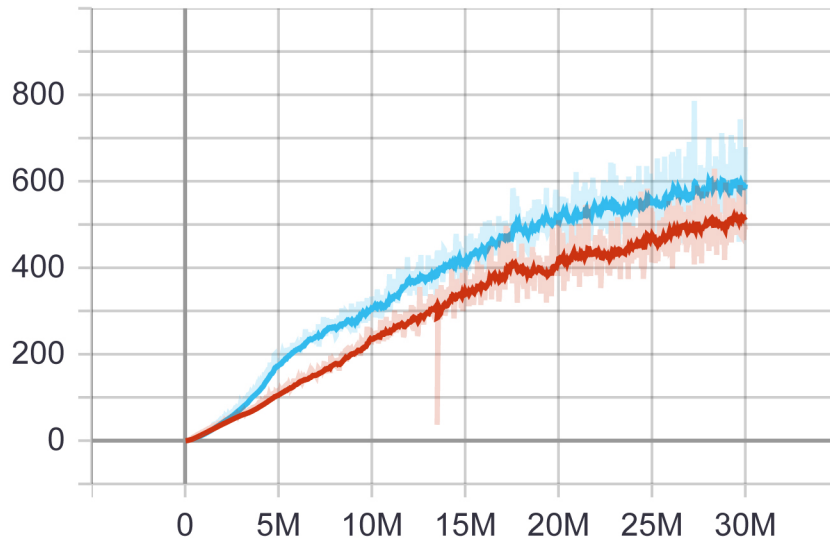


Figure 5.2: Environment Cumulative Reward

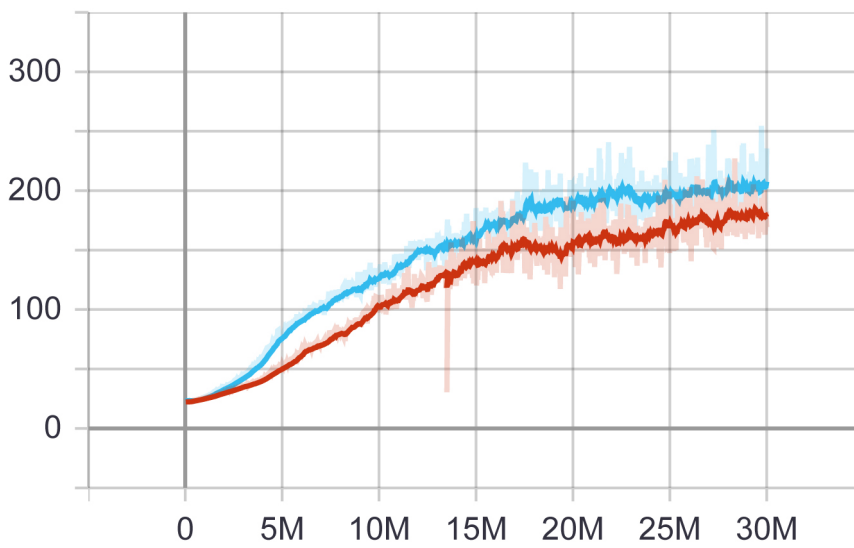


Figure 5.3: Environment Episode Length

5.3 shows the graphical representations of the mean cumulative reward and episode length. Here, the Cumulative Reward graph represents the mean cumulative episode reward of the two training sessions. The first training session(PPO) is color coded with red and the second training session(PPO + GAIL + BC) is color coded with blue. During a successful training session, the graph should be increasing[18]. From the Cumulative reward graph, it can be seen that the second training session generates more mean rewards as it is more increasing. In 5.4, the Episode Length graph represents the mean length of each episode[18]. And here we can see that the episode length is also longer for the second training session.

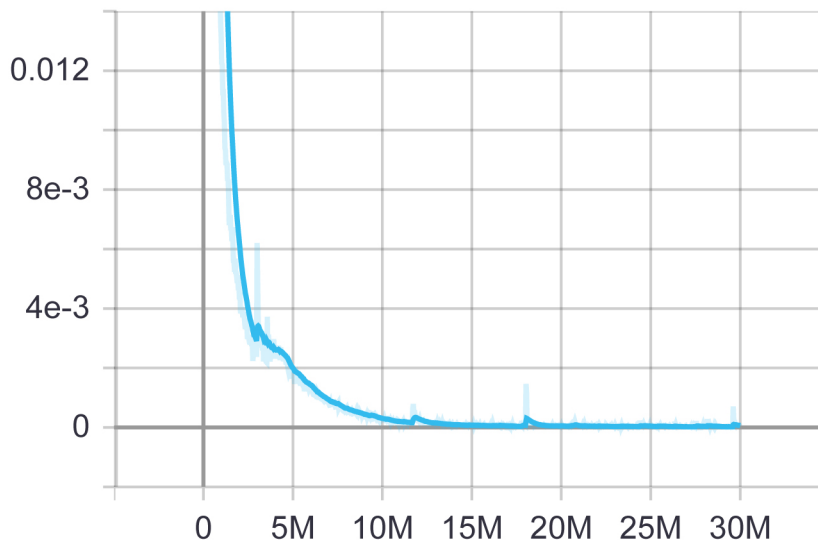


Figure 5.4: Losses GAIL Loss

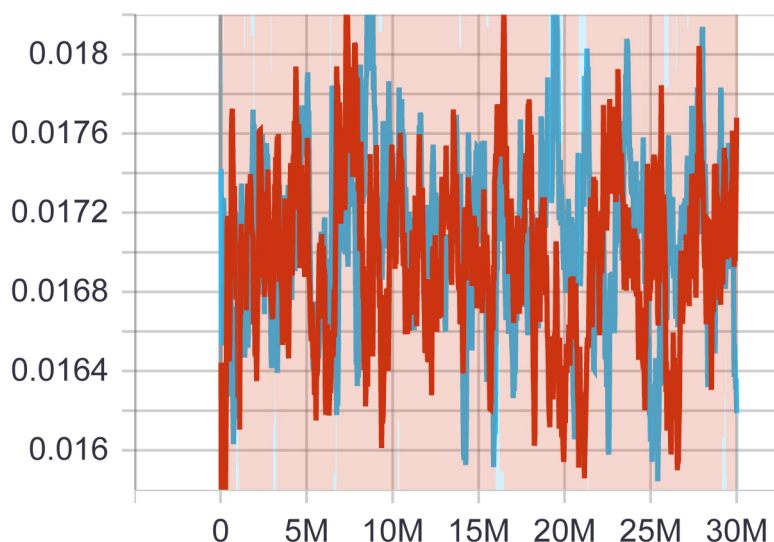


Figure 5.5: Losses Policy Loss

5.4, 5.5, 5.6 and 5.7 illustrates the graphs for loss functions. The GAIL loss graph is generated only for the second training session as it was not used for the first training session. This graph shows the mean value loss for the GAIL discriminator which

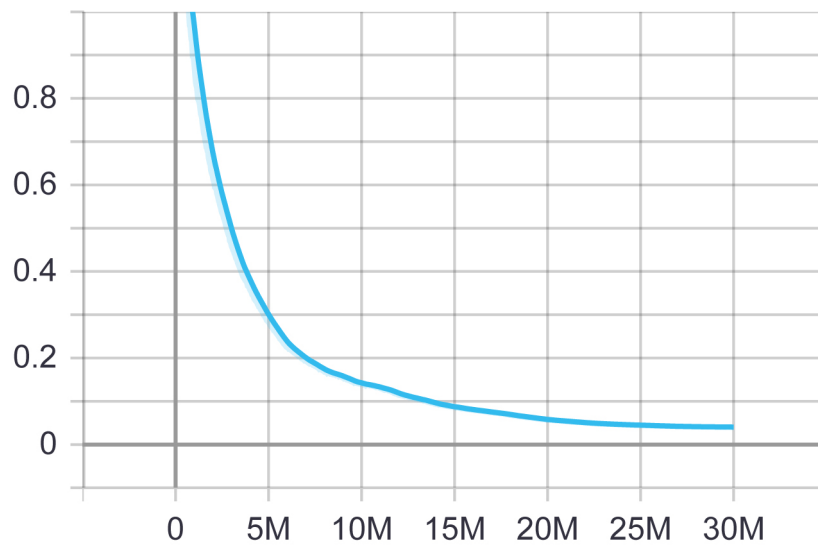


Figure 5.6: Losses Pretraining Loss

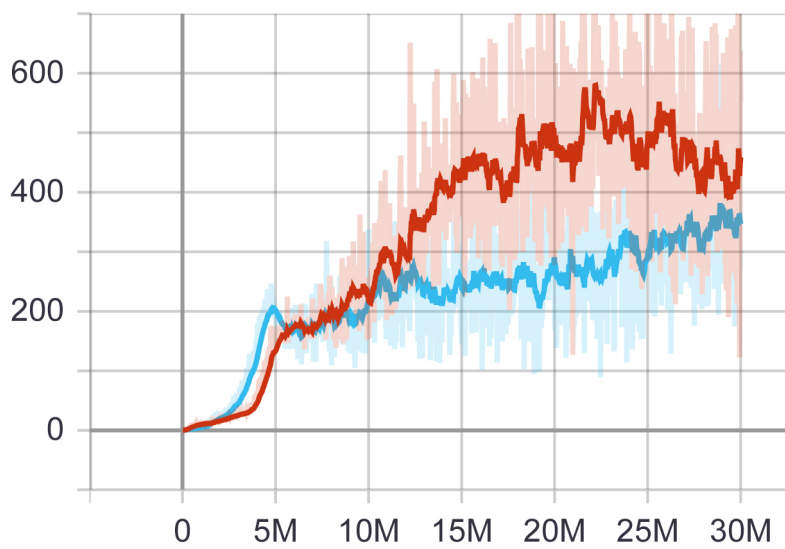


Figure 5.7: Losses Value Loss

tells us the performance of the model in imitating the demonstration data[18]. The mean magnitude of policy loss function is represented by the policy loss graph which shows us how frequently the policy is changing. On a successful training session it should be decreasing at the end and we can see that the 2nd training session is decreasing[18]. The pretraining loss graph is generated for BC which represents the mean magnitude loss for behavioral cloning. And similarly to GAIL loss, this graph shows us the performance of the model in imitating of the demonstration data[18]. Lastly, the value loss graph representing the mean magnitude of the value function update loss, should be increasing during learning and it should decrease when the reward becomes more stabilized. The graph portrays the performance of the model in predicting the value of each state[18].

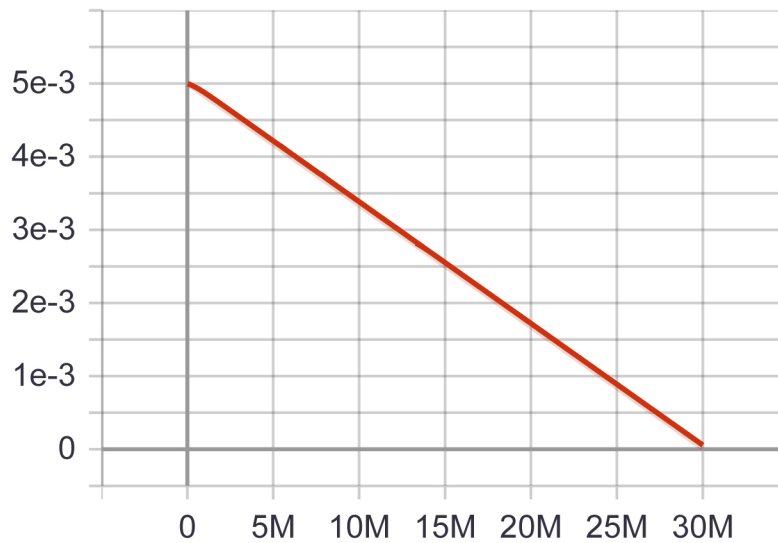


Figure 5.8: Policy Beta

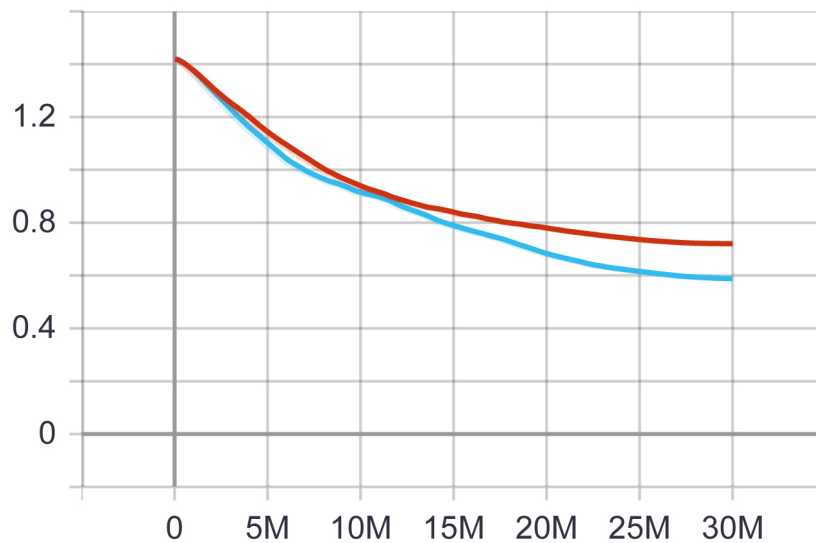


Figure 5.9: Policy Entropy

Now, let us consider the Policy statistics graphs for the two training sessions shown in 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 and 5.18. Entropy represents

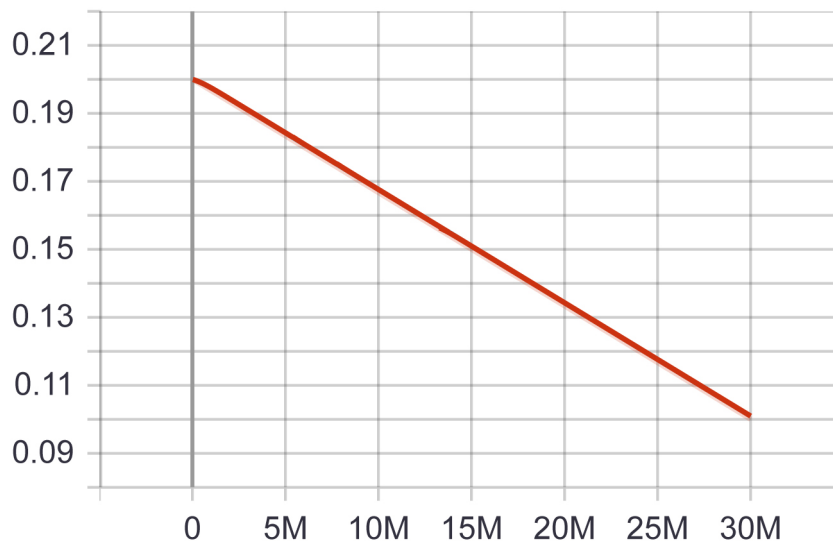


Figure 5.10: Policy Epsilon

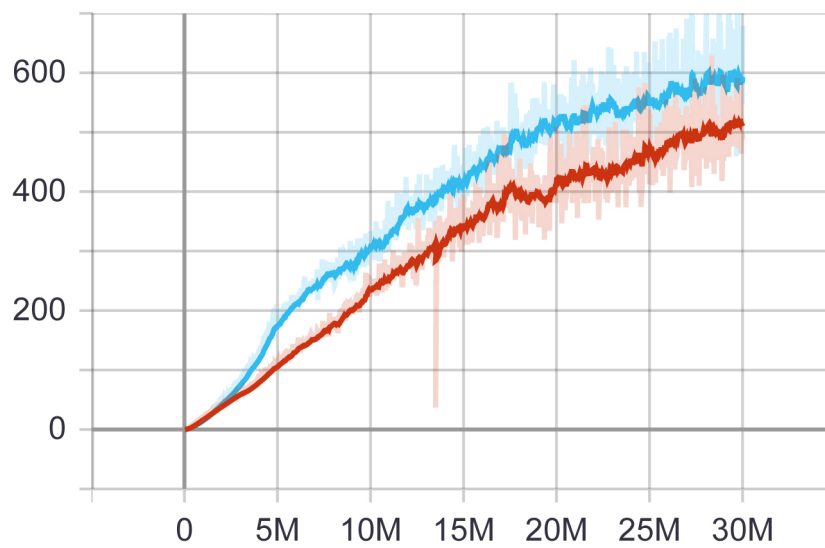


Figure 5.11: Policy Intrinsic Reward

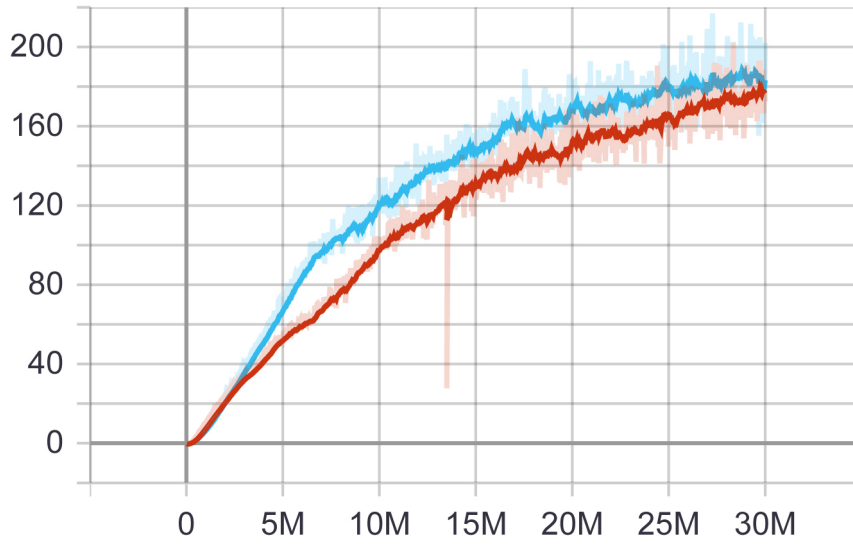


Figure 5.12: Policy Extrinsic Value Estimate

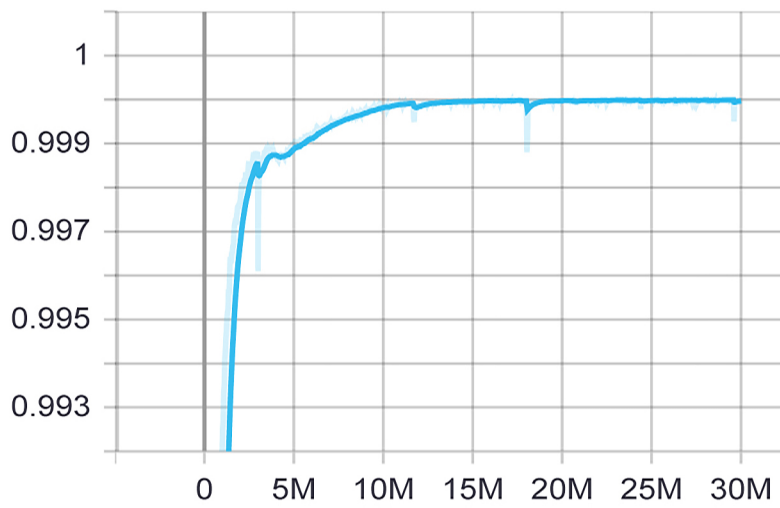


Figure 5.13: Policy GAIL Expert Estimate

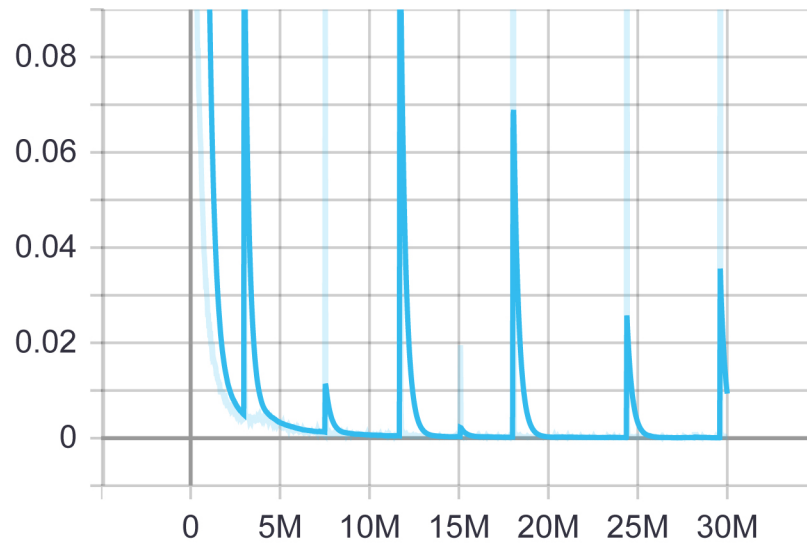


Figure 5.14: Policy GAIL Grad Mag Loss

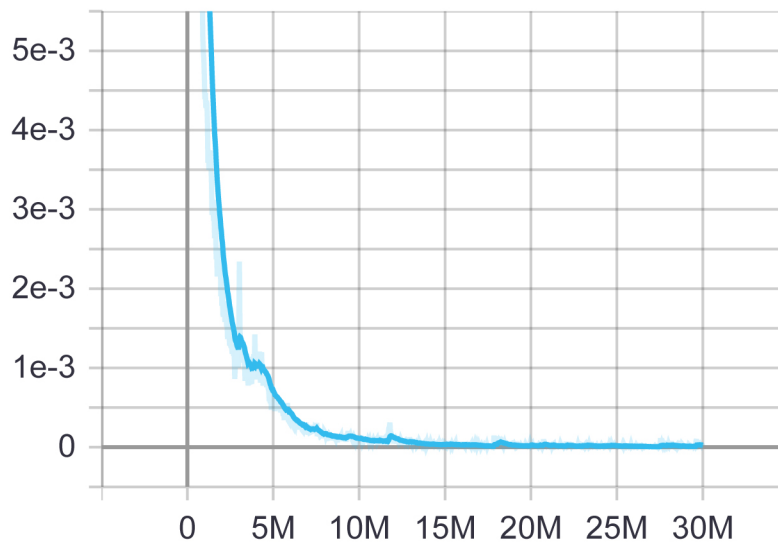


Figure 5.15: Policy GAIL Policy Estimate

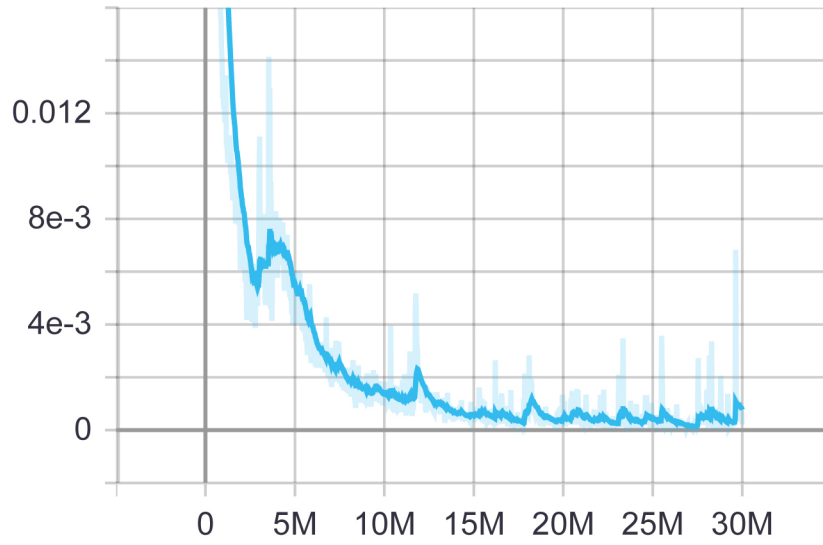


Figure 5.16: Policy GAIL Reward

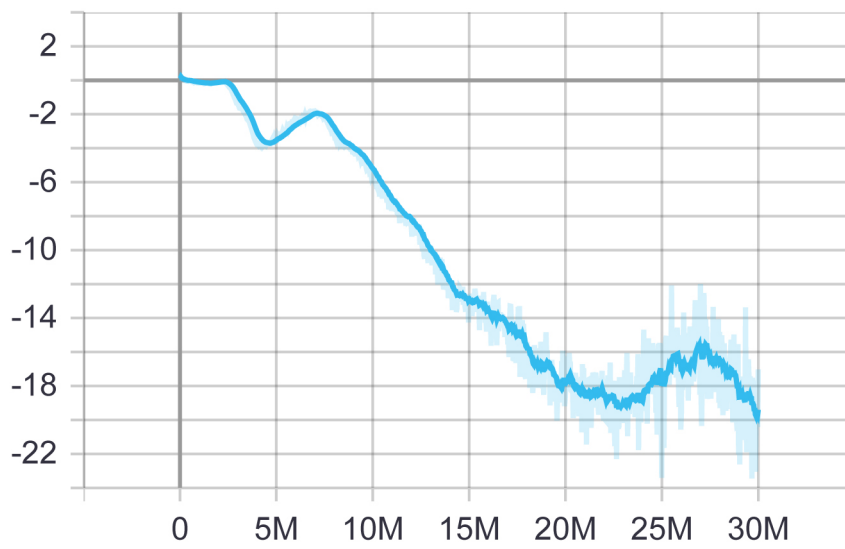


Figure 5.17: Policy GAIL Value Estimate

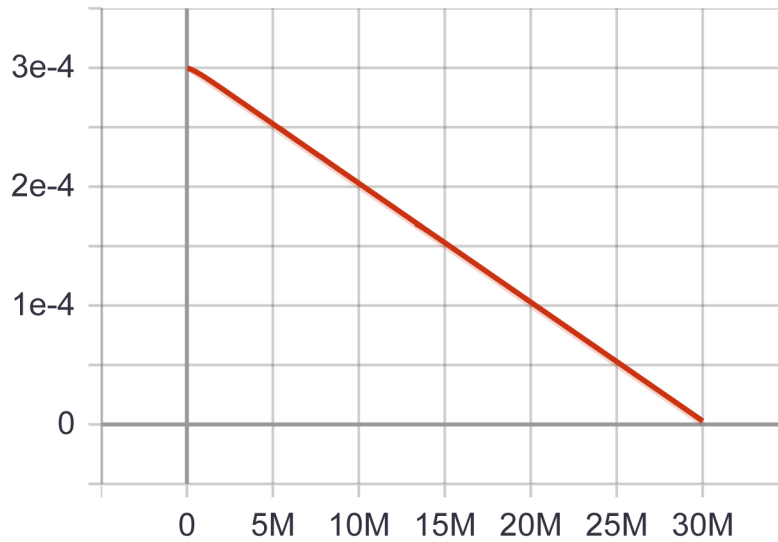


Figure 5.18: Policy Learning Rate

how randomly the model makes the decisions. This graph should decrease slowly for a successful training session[18]. Learning rate corresponds to how large the step will be in search for the optimal policy. Then extrinsic reward estimates how much mean cumulative reward is achieved in each episode. The value estimate graph should be increasing for a successful training session as it represents the mean value estimate for all states[18]. Now the GAIL reward and the GAIL value estimate graphs represent the mean cumulative discriminator-based reward and the GAIL reward value estimate respectively. Lastly, the GAIL policy estimate and expert estimate represents the estimate of the discriminator for policy generated states and actions and the expert demonstrations respectively [18].

Chapter 6

Discussion

We proposed an approach for character animation that combines Reinforcement Learning and Imitation Learning algorithms. First, we have trained our model with Reinforcement learning only. Though this training session was faster the results were not human-like animation. Then we trained the model with the combination of RL and IL algorithms. The results this time were far more human-like even though the training time increased. Our method of character animation proves that it is possible to get human-like character animation without the need for costly motion-cap studios and large databases. However, we faced some challenges when it comes to setting up the ragdoll. Our method seems to work with characters that have similar humanoid bone structure and the ragdoll setup needs to be precisely identical as well. Due to time limitations we also could not fix the arms to move like humans. We believe that with further exploration with this method any sort of animation can be made possible.

Chapter 7

Conclusion

With the proper tuning of the RL algorithm PPO in combination with IL and the use of ML agents toolkit, generating character animation in real-time will be easier than ever reducing production time, cost and resources to a very minimum. So far, we have used the ML agents toolkit to train agents with RL(PPO) and IL(GAIL BC) algorithms. We have generated better training results by tuning the default hyperparameters and modifying intrinsic rewards. For future work, we have a modification for the extrinsic reward that is yet to be tested. We believe with further research and development our method will perform even better animations. Finally, we hope that with future developments our presented model would be comparable or better than the state of the art methods.

Bibliography

- [1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [2] C. Szepesvári, “Algorithms for reinforcement learning,” *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.
- [3] H. P. Shum, E. S. Ho, Y. Jiang, and S. Takagi, “Real-time posture reconstruction for microsoft kinect,” *IEEE transactions on cybernetics*, vol. 43, no. 5, pp. 1357–1369, 2013.
- [4] A. Shingade and A. Ghotkar, “Animation of 3d human model using markerless motion capture applied to sports,” *arXiv preprint arXiv:1402.2363*, 2014.
- [5] J. Ho and S. Ermon, “Generative adversarial imitation learning,” *arXiv preprint arXiv:1606.03476*, 2016.
- [6] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, “Imitation learning: A survey of learning methods,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.
- [7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [8] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, *et al.*, “Unity: A general platform for intelligent agents,” *arXiv preprint arXiv:1809.02627*, 2018.
- [9] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, “An algorithmic perspective on imitation learning,” *arXiv preprint arXiv:1811.06711*, 2018.
- [10] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills,” *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–14, 2018.
- [11] A. Babadi, K. Naderi, and P. Hämäläinen, “Self-imitation learning of locomotion movements through termination curriculum,” in *Motion, Interaction and Games*, 2019, pp. 1–7.
- [12] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes, “Drecon: Data-driven responsive control of physics-based characters,” *ACM Transactions On Graphics (TOG)*, vol. 38, no. 6, pp. 1–11, 2019.
- [13] S. Starke, H. Zhang, T. Komura, and J. Saito, “Neural state machine for character-scene interactions.,” *ACM Trans. Graph.*, vol. 38, no. 6, pp. 209–1, 2019.

- [14] J. Schulman, *Proximal policy optimization*, Sep. 2020. [Online]. Available: <https://openai.com/blog/openai-baselines-ppo/>.
- [15] UnityTechnologies, *Unity-technologies/ml-agents*, Dec. 2020. [Online]. Available: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Virtual-Environment.md>.
- [16] J. Stoll, *Global animation market value 2017-2020*, Jan. 2021. [Online]. Available: <https://www.statista.com/statistics/817601/worldwide-animation-market-size/>.
- [17] Unity-Technologies, *Unity-technologies/ml-agents*, Apr. 2021. [Online]. Available: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Installation.md>.
- [18] *Unity-technologies/ml-agents*, Apr. 2021. [Online]. Available: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md>.
- [19] UnityTechnologies, *Unity-technologies/ml-agents*, Apr. 2021. [Online]. Available: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents-Overview.md#imitation-learning>.
- [20] [Online]. Available: <https://www.mixamo.com/#/?page=1&query=Ybot&type=Character>.
- [21] *Tensorboard : Tensorflow*. [Online]. Available: <https://www.tensorflow.org/tensorboard>.
- [22] *Tensorboard.dev preview easily host, track, and share your ml experiments for free*. [Online]. Available: <https://tensorboard.dev/>.
- [23] Tensorflow, *Tensorflow/tensorboard*. [Online]. Available: <https://github.com/tensorflow/tensorboard>.
- [24] Unity-Technologies, *Unity-technologies/ml-agents*. [Online]. Available: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Getting-Started.md>.
- [25] UnityTechnologies, *Unity-technologies/ml-agents*. [Online]. Available: <https://github.com/Unity-Technologies/ml-agents/tree/main/docs#unity-ml-agents-toolkit-documentation>.