BEHAVIORAL DIFFERENCES IN INTELLIGENT VIRTUAL AGENTS WITH AND WITHOUT ACTIVE VISION SYSTEMS

Submitted to:

Department of Engineering and Applied Sciences
in partial fulfilment of the requirement for the degree of
Master of Science in Artificial Intelligence

by

PUNIT SINGH

Supervised by: DR. ULYSSES BERNARDET

September 2021

ABSTRACT

With research on Virtual Reality technologies gaining traction and focus on virtual agents situated in 3D environments, it is important to know and understand what information the agents have access to and how the information is provided to the agents when expected to perform certain tasks. This dissertation thesis tries to understand how possessing an active vision system or a passive vision system by an agent affects the performance of the agent in a task that involves locating and moving towards a flower object and drinking nectar from the flowers placed in the environment. This experiment tests four agents with different perception systems, starting with an agent with a passive perception system that gets symbolic information about the location and position of the flowers and the distance to the nearest flower and also uses a RayCase sensor to sense the environment, and gradually reduce the number of observations that each agent is getting form the environment with the fourth agent having active perception depending only on a camera feed. The experiment is in part a comparison of agents using symbolic information and non symbolic information about the environment and trying to find out the behavioural differences in agents when they have access to different levels of information about the environment.

The experiment was planned such that the agents are completely identical in the sense that they even have the same architecture of the neural network, but with the different amount of data and graphical requirements being used to derive inferences for completing the task of drinking nectar form the flowers, after pre experiment runs, it was hard to determine which hyperparameters are best for training all of the agents. The final experiment was then run using different network architectures and hyperparameters for the different agents.

ACKNOWLEDGEMENTS

I would like to thank everyone who played a part in my academics. First of all my parents and my friends who supported me through everything.
Secondly, I would like to thank my supervisor – Dr. Ulysses for guiding me through this project and allowing access to the lab for this rather graphics heavy experiment. This dissertation project would not have been possible without his support.
I would also like to thank my sister Ishveen for keeping me motivated throughout this project and also while writing this thesis.
Thank you all for your support.

DECLARATION

I, Punit singh declare that this Master's Thesis titled **"BEHAVIORAL DIFFERENCES IN INTELLIGENT VIRTUAL AGENTS WITH AND WITHOUT ACTIVE VISION SYSTEMS"**, is my original work, and has not been submitted for an award of a degree in any other university or college. I would also like to declare that the base environment used for the experiment in this dissertation is provided by Immersive limit tutorial on Unity learn platform and that it has been properly cited and given due credit in the appropriate section and the bibliography.

Birmingham, UK, September 2021

Punit Singh

CONTENTS

I FORMATION OF THE IDEA 1

1 INTRODUCTION 3
- 1.1 Inception . . . 3
- 1.2 Introduction . . . 4
- 1.3 Problem Statement . . . 4

2 LITERATURE ANALYSIS 7
- 2.1 Passive Vision . . . 7
- 2.2 Active Vision . . . 7

3 DESIGN 13
- 3.1 Requirements . . . 13
  - 3.1.1 3D Environment Simulator . . . 13
- 3.2 The experiment . . . 14
  - 3.2.1 Reward Penalty System . . . 17
  - 3.2.2 Agent Design . . . 19

4 PROJECT MANAGEMENT 21

II IMPLEMENTATION 25

5 PREREQUISITES 27
- 5.1 Anaconda . . . 27
- 5.2 Setting up the scene . . . 28
- 5.3 Agent Setup . . . 29

6 THE ENVIRONMENT AND METHODS 31
- 6.1 FlowerArea.cs script: . . . 31
- 6.2 Flower.cs script . . . 31
- 6.3 Hummingbird Agent scripts . . . 33
  - 6.3.1 HummingBirdSymbolic.cs - passive perception system . . . 35
  - 6.3.2 HummingBirdHybrid1.cs - Hybrid perception system 1 . . . 36
  - 6.3.3 HummingBirdHybrid2.cs - Hybrid perception system 2 . . . 37
  - 6.3.4 HummingBirdActive.cs - Active perception system . . . 37
- 6.4 Camera Sensor . . . 37

7 TRAINING 43
- 7.1 Training using ml-agents . . . 43
- 7.2 TensorBoard . . . 46

III EVALUATION AND RESULTS 47

8 HYPOTHESIS 49
- 8.1 Hypothesis 0 . . . 49

9 THE RESULTS 51
9.1 Analysis on the results . . . 52
9.2 Summary . . . 53
10 CONCLUSION 57

IV APPENDIX 59
A SOURCE CODE 61
A.1 Flower Area Script . . . 61
A.1.1 The Reset Flowers method . . . 61
A.1.2 Finding child flowers on the island . . . 61
A.2 Script for each flower . . . 62
A.3 The Hummingbird agent scripts . . . 63
A.3.1 Common methods: . . . 63
A.3.2 CollectObservations() methods for each hummingbird . . . 67

LIST OF FIGURES

Figure 1 Representation of the Inverse Augmented Reality (Zhang et al., 2018) 8
Figure 2 A representation of how active perception looks in effect. (Terzopoulos and Rabie, 1995) 9
Figure 3 Binocular vision applied to the DI-Guy (Rabie and Terzopoulos, 2000) 11
Figure 4 Binocular vision in animat fish. (Rabie and Terzopoulos, 2001) 11
Figure 5 Floating Island 16
Figure 6 Bush Obstacles 16
Figure 7 Rock Obstacles 16
Figure 8 Island scene with agents 18
Figure 9 Hummingbird Agents on the Island 18
Figure 10 Flower objects on the Island 18
Figure 11 Initial plan for the project 21
Figure 12 Changed plan after running into errors 22
Figure 13 Final project plan with 15 days off for coursework 23
Figure 14 Flowers when full 32
Figure 15 Empty flowers 32
Figure 16 The Nectar Collider inside flowers 33
Figure 17 The beak tip of the hummingbird agents . . . 33
Figure 18 single RayCast sensor mounted on a hummingbird agent 36
Figure 19 Top view of the RayCast sensor in the environment. 36
Figure 20 The distorted view seen in the camera at wider field of views 39
Figure 21 The mounting of the camera sensor on the agents 39
Figure 22 The mounting of the camera sensor on the agents - side view 39
Figure 23 View from the camera 40
Figure 24 The actual 300x150 view of the environment as seen by the agents 40
Figure 25 Clipping distance for the cameras 40
Figure 26 Culled image for the neural network 41
Figure 27 All agents training simultaneously 44
Figure 28 The active agent in training 45
Figure 29 The passive agent in training 45
Figure 30 Agent's perspective when training 46
Figure 31 Cumulative reward of all four agents 51
Figure 34 Rewards for passive perception system 52

Figure 32 Average reward of all four agents 52
Figure 33 Entropy for all agents 52
Figure 35 Rewards for Hybrid 1 perceptionsystem 53
Figure 36 Rewards for Hybrid2 perception system 53
Figure 37 Rewards for Active Perception system 53
Figure 38 Average rewards for each agent 54
Figure 39 Cumulative rewards for agents 54
Figure 40 Entropy for each agent. 55

LIST OF TABLES

Table 1 Information available to the agents 19
Table 2 Observations collected by agents 19
Table 3 Hyperparameters for each agent 43
Table 4 Legend for Figure 31 51

LISTINGS

Listing 1 Create conda environment 27
Listing 2 List conda environments 27
Listing 3 Activate conda environment 28
Listing 4 Install mlagents 28
Listing 5 .yaml file contents 44
Listing 6 Training Command ml-agents 45
Listing 7 Run tensorboard 46
Listing 8 resetFlowers() method 61
Listing 9 findChildFlowers() method 61
Listing 10 Feed() method 62
Listing 11 OnEpisodeBegin() method 63
Listing 12 OnActionReceived() method 64
Listing 13 MoveToSafeRandomPosition 65
Listing 14 TriggerEnterOrStay() method 66
Listing 15 Symbolic and hybrid 1 agents collected observations 68
Listing 16 Hybrid 2 agents collected observations 68

Part I

FORMATION OF THE IDEA

In this part of this dissertation thesis, I explain the basic concepts behind this experiment and define the problem statement on both a higher level and a lower level. I then go through the literature and previous research related to this project along with the thought process that led up to the formation and design of this experiment.

1 INTRODUCTION

1.1 INCEPTION

Recently Samsung Sam - A virtual human introduced as Samsung's new virtual agent- went viral on the internet. Although Sam may not have been acknowledged by Samsung officially, Sam going viral does show how interested people are in virtual humans they can interact with. We also have other mainstream virtual agents like Apple's Siri and Amazon's Alexa that can be designed into virtual humans.

GlobalData.com, in their September 2021 thematic report (GlobalData, 2021) mention that though the market for Augmented Reality was worth $7 billion in 2020, over the next 10 years it will increase at a compounded annual growth rate of 36% reaching a total revenue of $160 billion, of which around $45 billion will be in the consumer market. This report also implies that although AR technology is still dependent on smartphones for processing capabilities, it intends to replace our smartphones and act as standalone devices in the next decade or so. Once AR is mainstream, there will also be a lot of virtual agents who live on our AR devices like the agents that live in our devices now (Siri, Alexa etc.), the only difference being that to thrive in a mixed reality world (Billinghurst and Kato, 1999), the agents will need to have a visual form that humans can see and interact with.

The researchers at AI Group have been developing "Max" (Weitnauer et al., 2008), a virtual agent and studying the in-situ interactions between Max and real humans in the virtual world of the Second Life online virtual world. The article states a few prerequisites that a virtual agent must have if it is to act in a virtual environment. The prerequisites being a) the ability to perceive, b) the ability to act, c) the ability to reason and d) the ability to have and display social skills.

When reading a blog post about one of these agents, a question I asked myself was how do these agents see? It was obvious that they do not, they have perfect world knowledge or atleast access to all knowledge about the environment they exist in, and do not need to "see". But what about in dynamic situations, where these agents have to co-exist with humans, or other non deterministic actors, in these situations, it gets really hard to keep track of all the actors or even have perfect world knowledge leading to the environment itself having imperfect world knowledge and inability to predict what the

attributes of an actor in the environment are going to be next. What if we give these virtual agents the ability to see, perceive visually and observe the environment just like us humans have. **The question itself is justified in the larger scheme of Artificial Intelligence research by another question that how can we expect to build truly intelligent agents that try to imitate human cognition without even providing them with one of our most prominent senses, "Vision"?**

1.2 INTRODUCTION

This dissertation thesis focuses on the perception of intelligent virtual agents with emphasis on visual perception and how possessing either passive vision or active vision change the ways in which the agent performs certain tasks in a virtual environment. The idea being that once the agents are put in a real world environment, like in an Mixed Reality environment, the agents can only be supplied with a limited number of observations in symbolic form and those too can be localised, like the orientation and direction of the agent and very few non-local observations like the nearest interactable object or metadata about other agents. Although this information is extremely important for agents to act independently in an environment, it is only practically implementable in environments with a small number of agents and a world that is limited in scope. For example, in Second Life, as of mid June 2021, the average daily usage of the platform was 200,000 users (Voyager, 2021). If we were to put 200,000 Intelligent Virtual Agents in a metaverse as large as Second Life, the amount of data required as symbolic input for all the agents will be too large to manage. Another thing to consider is that when put in mixed reality environments, we do not always have access to 100% information about the world, therefore depending completely on symbolic data for agent behaviour might not be entirely possible leading to some missing values or noise in the symbolic data leading to a noise in the action and performance of the agents. This dissertation project tries to find a way to significantly reduce the amount of symbolic data required by the agents for acting in the environment by equipping them with a way to visually perceive the environment and collect relevant data on their own without being fed any, or at most very limited information by the environment.

1.3 PROBLEM STATEMENT

When talking on a larger scale, agents that seem to exist and live in virtual worlds or metaverses, if aiming to be truly "alive", should have independent access to information about their environments in order to perform the various tasks they are designed for without in-

terference or having to depend on any other system designed to feed information to the agents. At a higher level, the problem statement would be that if we give the virtual agents a sense of vision instead of providing it with data in the form of pre-calculated numbers, will the agent be able to make the same or even better sense out of the visual feed as it made when it did with symbolic data? Will the agent completely fail to perform the same tasks it could perform quite easily when provided with just numbers, or will it perform even better than its symbolic counterpart finding better and efficient solutions for the task that it is given or will there be an equilibrium where the dependency on symbolic data and visual data is balanced and the best solution will be attained.
It is a fair point to say that using visual perception is completely contextual and depends on the task to be done, we cannot use visual perception with all agents and for all problems. Yes, completely agreed. But this project focuses on a niche task identifying flowers in an area, moving close to the flowers and drinking nectar present in the flowers. It is a task, which, if we ask a human to solve, the human would rely completely on their visual senses solve. *For example, if I am given a task of collecting nectar from the flowers scattered around an area, I will not go asking around for the exact location of the flowers, I will prefer to look around the area myself and locate the flowers that are visible from my current location and gather nectar from all of them and then look around for more flowers.* The environment design itself is such that this problem can be solved using visual senses. The flowers that have nectar in them are red in colour present in a green dominated environment (green grass, bushes, grey rocks etc.) and once all the nectar from a flower has been removed, the colour of the flower changes to purple. This should make it easier for a visual agent to differentiate between a) the flowers and the environment because according to the traditional RGB colour model, red and green colours are complementary, and b) flowers that have nectar and those that are empty because red and lavender are also quite distant from each other on the traditional RGB colour model, so locating the red flowers on a green background and differentiating between the empty purple and red full flowers should be fairly easy.

On a lower level, the problem is about finding a balance between the amount of symbolic data that can be provided to a virtual agent and the amount of information the agent can gather using an active vision system in a way that is neither heavy on resources like processing power nor causes the agent to be highly dependent on information that is provided to it. This problem focuses on the approach suggested in the book Active Vision (book Active Vision) that in the case of a visual environment, an entity hoping to be intelligent might benefit more from being able to observe the environment when the

sensor is located inside the environment rather than observing the whole environment externally or all at once.

2

LITERATURE ANALYSIS

Being a rather important discipline since computers became mainstream, computer vision to a large extent has been concerned with passive inversion of the image formation process, to either produce 3D reconstructions of the world or, at a higher level, to identify objects and their location orientations.
In the sixties, it was discovered that simulating human vision with computers was a much harder path to attain than imagined. Although our own vision seems effortless, doing the same for computers is a massive task. So, although other harder problems seemed to have been solved or progressing at a much higher pace, computer vision has made slow progress largely because of the unavailability of hardware that can process graphic heavy processes. However, simplification of scenes to a world of blocks with uniformly coloured faces did show a substantial amount of progress, which paved the way for two landmark projects :

1. Copy demo at MIT (Shah, 2002), and
2. Robot Shakey at Stanford Research Institute (Nilsson et al., 1984).

2.1 PASSIVE VISION

Among hardware systems that add visual perception to computers or virtual agents like cameras, LiDAR sensors, depth sensors or proximity sensors, there exists a perceptual oracle or oracle vision system that does not require any hardware. In perceptual oracle systems, agents in a virtual environment access information about the environment and the agents' locality by sending queries to the environment itself and get access to highly processed and privileged information which is not available to animats otherwise like if an animat is in a swarm and possesses perceptive oracle system, it can access the number of animats in a swarm or the absolute location of the animat with respect to the environment along with its relative position. This can cause a hive mind like situation where each animat is fed the same information and the whole herd is led to a unanimous goal rather than individualistic actions and reflexes.

2.2 ACTIVE VISION

According to (Marr and Vaina, 1982), Vision is knowing what is where by looking. (Shirai, 1972), (Jarvis, 1983) and (Besl and Jain, 1982) de-

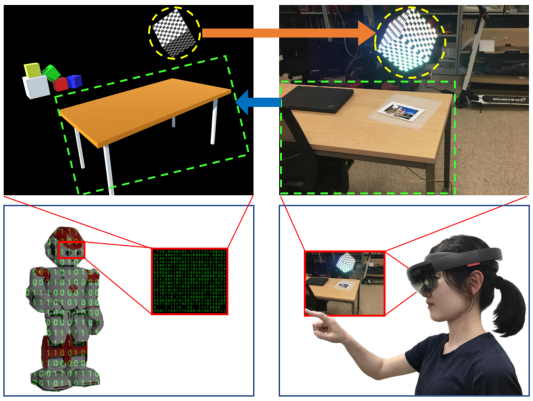

Figure 1: Representation of the Inverse Augmented Reality (Zhang et al., 2018)

fine active vision as a vision system that uses sensors that do not emit of radiation on their own. Rather they depend on sensors that can only take input from the environment. (Bajcsy, 1988) says that, in a way, active vision systems can be achieved by only using passive sensors.

One of the reasons that the Active Vision has become so important is that it is a matter of technological opportunity as well as of the traditional approaches to computer vision running aground (Owen, 1993). The mid-eighties involved the use of computing power just in the experimentation in Computer Vision for analysing the static images. However, in the last five years, it's become relatively easier to analyse images owing to relatively powerful general purpose processors becoming relatively cheap.

According to (Marr and Vaina, 1982), Vision is knowing what is where by looking. (Shirai, 1972), (Jarvis, 1983) and (Besl and Jain, 1982) define active vision as a vision system that uses sensors that do not emit of radiation on their own. Rather they depend on sensors that can only take input from the environment. (Bajcsy, 1988) says that, in a way, active vision systems can be achieved by only using passive sensors.

(Bajcsy, 1988) also defined active perception as a data acquisition problem in an intelligent way where systems proactively take steps to gather more information about their environment. The paper explores both, a top down approach based on a task or a query from a database and a bottom-up approach where the task is initialised without any database query and with the aim of exploring the environment. In the first step they gather either a geometric skeleton of the scene or a set of differentiating factors to be searched for followed by a search operation in the database enabled by a decision making algorithm which is a primary goal of the active vision systems. In Figure 2, we can see the field of view of the fish where because of the

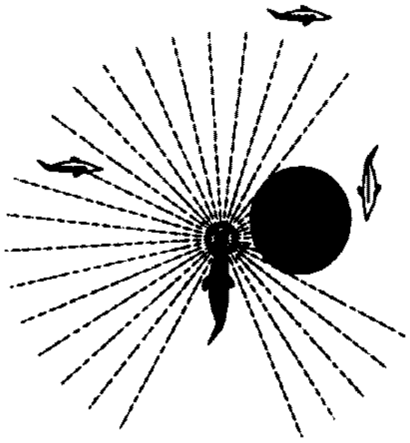

Figure 2: A representation of how active perception looks in effect. (Terzopoulos and Rabie, 1995)

object occlusion, the fish can only see the fish on the left side while the fish on the right is hidden behind the spherical object. The center of the perspective is the eye position of the fish and it cannot see behind itself. Also, the fish cannot see the fish right in the front because the range of its vision does not reach that length.

Active Vision is not concerned with the methods used to observe the environment but with the strategies that are used for collecting information about the environment. According to (Spetsakis and Aloimonos, 1987), though there is continuous interaction between the two, in active perception, the observer is the active entity and not the sensor. This also involves analysing available visual sensory data to answer questions generated by the observer. The observer also adjusts its vantage point, in order to allow the sensor to help uncover a specific piece of information or problem that demands immediate attention.

Active observer basically, has an opportunity to plan sensor-actions in real-time based on cumulatively acquired information incrementally. The main aim is to maximise the information acquired during each step in order to obtain plausible results consistent with the intent of the goal task. Information theoretic measures of relevant information have been used in machine learning (Quinlan, 1986). When it concerns the sensory planning problems involving simple sensors, rigorously bayesian measures have been put to use.

The real test of Active Vision Paradigm has been its efficacy in assisting to build seeing systems. These systems are seen generally operating at real time rates performing navigation, recognition and surface analysis. Of course these systems are not as advanced as put forth by the scientists around 10 to 15 years ago, but being somewhat cosseted and protected they are effectively able to perform well de-

fined tasks in insulated environments.
Active Vision viewpoint is expected to evolve and change considerably. It suggests a paradigm shift (Kuhn, 1962) a radical change of emphasis on what is considered important in vision. Such transitions require the development and availability of new tools and initially can pose more problems than they solve. As stated by ((Owen, 1993)), there are more problems to be solved concerned with designing simulated elastic mechanisms, algorithms for future search and efficient control systems. The progress of the active visions systems from performing simple tasks to gradually shifting to more difficult ones has been very promising.

(Burton, 1993) puts forward, that true knowledge can not exist without perceiving the environment around us independently and the same applies to Artificial Intelligence. The Molyneux problem (Davis, 1960) is a classic analogy to be given here; assume a visually impaired person, who has learned to distinguish by touch, different shapes, say, a cube and a sphere is suddenly provided with an ability to see, will the person be able to distinguish between the same sphere and the cube only by seeing both of them placed on a table in front of them and not touching them. The answer to this problem was given by (Held, 2011) when their study showed that the newly sighted subjects (previously blind people who had their vision restored by medical procedures) were not able to differentiate between different shapes based only on visual input. In the case of AI, it doesn't matter how better AI get at performing tasks, if it gets better only at performing tasks, we cannot call it truly intelligent. (Burton, 1993) gives a good example that a chess playing AI does not know that it is playing chess, for that matter, it does not know what chess is, just like a computer does not know that it is a computer. You have to program a computer for it to function properly and you have to train an AI for it to behave intelligently.
Human intelligence and artificial intelligence differ in the sense that humans do not receive input from the environment passively, we actively seek new sensory inputs form the environment and that is how we learn (Kozma, 2007). Artificial Intelligence on the other hand is more goal oriented in the sense that if the goals that the AI is defined to accomplish, are being reached , the AI will not try to learn anything new or will not even explore the environment it is placed in and that might have something to do with the absence of perceptive "organs".

Animat fish were used by (Terzopoulos and Rabie, 1995) in an attempt to give them active vision without the use of any specialised hardware that might aid the fish in gathering information of their environment. Computer vision algorithms were applied on top of their virtual retinas. Some fish in the environment employed perception or-

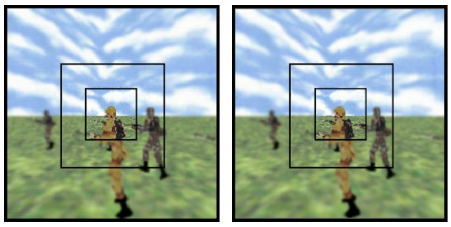

Figure 3: Binocular vision applied to the DI-Guy (Rabie and Terzopoulos, 2000)

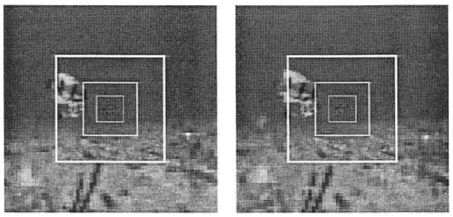

Figure 4: Binocular vision in animat fish. (Rabie and Terzopoulos, 2001)

acle as their vision system where they got the local, geometric, chromatic and every other form of information available to the rendering engine by directly integrating with the environment in a meta way.
They gave binocular vision to the animat fish such that the eyes were controlled by two "gaze angles," one for the horizontal rotation and one for the vertical rotation (θ, ϕ). The system was setup such that when the eyes are looking forward, the gaze angles are $0°$, $(\theta = \phi = 0°)$. The field of view also plays an important role when designing vision systems for agents. Figure 4 shows the binocular perspective of the artificial fish.

A majority of the research on active vision systems has been done in the early 1990's with researchers trying to implement active vision systems on animats (Rabie and Terzopoulos, 2000), (Tamer F. Rabie, 2001), and virtual humans like DI-Guy (Rabie and Terzopoulos, 2000). They used different techniques for implementing active vision and visual perception as the sole input to the agents, some of them being colour histograms or even developing virtual eyes that could move independently and send feedback to the motor control mechanism so that the agent can move based on the feedback. Now that simulating softwares and frameworks like Unity and Unreal etc. are easily available, this project uses Hummingbird agents in a 3D environment to simulate the effects that possessing active or passive perception systems have on the performance of the agents.
The environment consists of an island suspended in 3D space. On the island, there are different objects: a tree, some rocks spread on the surface, a few bushes and some flowers placed on collections of flower plants. Each flower contains nectar and the goal of the hummingbird agent is to feed on as much nectar as it can in an episode. At the start of the episode, each flower is red in colour and stays so until

it has any amount of nectar present inside it, however once the hummingbird agent drinks all the nectar from the flower, the colour of the flower changes to purple. This project uses reinforcement learning to train the agent that the goal is to feed on nectar, so a reward/penalty system has been put in place. The reward system is discussed in detail later in this thesis.

3

DESIGN

For this experiment to be a fair comparison and easily executable and reproducible, we need a way to simulate a 3D environment that can host multiple virtual agents each having different capabilities for sensing the environment and ability to navigate the environment freely performing tasks and gathering rewards / penalties. When put like this, it sounds a little bit complicated, however we can break down the requirements into smaller parts.

3.1 REQUIREMENTS

3.1.1 *3D Environment Simulator*

There are multiple softwares available that can simulate and design 3D environments like OpenSimulator (OpenSimulator, 2021), Gazebo (Gazebo, 2021), VRep (Vrep, 2021), WeBots (WeBots, 2021), Unity3D (Unity, 2021) and Unreal Engine (Unreal, 2021). Each of these can be used to simulate 3D environments and can host virtual agents but each of these has their own strength and use case. We need to choose one that can be used to train the virtual agents with ease and in a way that can be upscaled and used with other environments and with agents that have different tasks to perform. For this project, we will be using Unity3D for simulating the experiment because of the ml-agents framework (Unity-Technologies, 2021) available for it which can train agents using deep learning at a much faster rate than Unreal's MindMaker AI. We choose ml-agents over MindMaker also because the ml-agents framework is fairly straight forward to use and training can be started only using one command in the terminal.
Unity is an open ended and modular video game engine that is primarily used to make and design video games. Modularity of the Unity engine means the capability of the software to use external packages and add more functionality to the games. Since recently games have been trying to incorporate artificial intelligence in their non-playing-characters (NPC's) and also in opponents, packages like ml-agents have been developed to aid the training and creation of intelligent virtual characters for games, and luckily for AI researchers, this means that they have a large platform to run and simulate experiments for their experiments with Intelligent agents that can be comparatively easily deployed in a 3d environment and studied rather than having to develop a specialised framework that carters to only one or similar experiments. Unity3D can be downloaded form the of-

ficial unity website (Unity, 2021).

3.2 THE EXPERIMENT

The experiment is designed such that there are 4 hummingbird agents situated on a floating island suspended in an infinite 3d space [5]. The island is populated with obstacles like rock clusters, a tree in the center of the island, clusters of bushes shown in images [6 - 7]. The environment is also populated with 4 hummingbird Agents each having its own perception system; one that is completely symbolic and uses 10 symbolic observations fed into it from the environment, the second one being a completely camera-based perception system where the neural network of the agent only gets fed the raw RGB pixels from the camera, the third one is a hybrid system where the neural network is provided with all of the symbolic data along with the raw RGB camera feed and in the last one, which is also a hybrid system, a limited number of symbolic observations along with the raw RGB camera feed is sent as input into the neural network. This experiment would be a fair comparison of these perception systems only if the underlying decision system of all the agents is the same, that is if they all use the same architecture for the neural network. **??** shows the general architecture and design of this experiment. For the agents that have used the raw RGB camera feed, we need to use some convolutional layers for feature extraction but after the convolutional layers, the neural network architecture of all the four agents is the same and even for the agents where convolutional layers are used, the same network has been used for all three of them so that the comparison of the results is as fair as it can be. All four agents have been placed in the same environment (on the same island, for this experiment) and the agent that can get the maximum reward per episode, has the better perception system among these four because it is only using their perception system the agents are able to locate the flowers on the relatively large island and among the obstacles. Obstacles play an important role in the experiment because of the reward and penalty system.

In order to test the visual capabilities of an agent, one of the methods is to give the agent the task of finding an object, a totem located in the environment, and associate finding the object with getting a reward. For this project, an agent capable of navigating an environment populated with objects - collectables and non-collectables - would be the ideal choice. The agent should have a few basic capabilities which are identified as a) locomotion, the ability to move around in the environment, b) the ability to rotate about the three axes in its current position, c) the ability to perceive the environment in which the agent is situated. Making a bi-ped agent would have been the first choice but

then the agent would have to learn how to walk in the environment and avoid obstacles which is a different problem altogether. We need an agent capable of moving in a straight line without having to worry too much about the obstacles or the geography of the environment and also the problem with bi-ped agents is that they can only rotate about two axes and not the third one. In order to simulate the gaze angles for a bi-ped agent, the camera system of the agent will have to be given a separate gaze controller and again the agent will have to learn how to control the gaze angle so that it is looking directly at the objects and not too high up or too low. This experiment also needed a way to compare the differences between the performance of agents with symbolic data input and active vision raw pixel input without having to compromise on any of the other conditions. In essence, the agents with the symbolic perception system and agent with the active perception system should be able to co-exist in the same environment and have the exact same characteristics and "brain" with the exception being that one agent is fed symbolic data and the other is fed a raw camera feed. After following along a tutorial for training a virtual agent in unity with MLagents, I decided to use the tutorial environment itself as the base of this experiment in which a bird agent that can fly around in the environment without having to worry about the altitude of the ground or any obstacles in its path to the flowers. Yes, there still are some obstacles for the agent like the tree and the bushes that the agent can run into but it will have to learn to avoid those.

Having confirmed that we will be using unity to simulate the experiment, Detailed tutorials that explain how ml-agents is used to train virtual agents in a virtual environment are needed. This project uses the ml-agents tutorial (Limit, 2021) from the unity's learning page for the assets and for the basic scene setup. The tutorial link provides the base scene for this project and all the meshes that have to populate the floating island on which the experiment takes place. The tutorial provides a single zip file that contains the .unitypackage file that can be imported into a new scene and set up. It is also possible to design a new scene for this project from scratch but that would include 3D modelling the meshes from scratch using a separate software like Blender3D or Autodesk Maya and would also need textures. That is too time expensive, so I went with the assets provided with this tutorial.

Figure 5: Floating Island

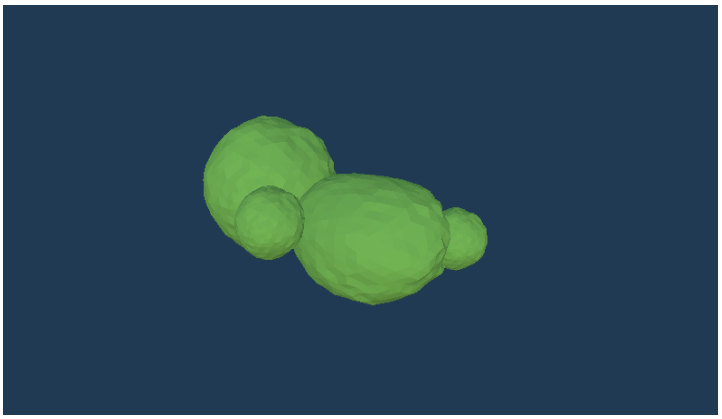

Figure 6: Bush Obstacles

Figure 7: Rock Obstacles

3.2.1 *Reward Penalty System*

The reward and penalty system of this experiment is such that the hummingbirds get a major reward only if they satisfy one condition that is, they are successful in drinking nectar from any of the flowers, but if the hummingbirds collide into obstacles, then the agents get small penalties, the amount of penalty applied depends on the offence committed by the agents and also on how far off from the goal the agents are. If an agent collides with the stem of a flower plant, it means that it is really close to the flower, it then gets a small penalty. However, if the agent is completely off course and nowhere close to completing the task like the agent tries to fly off the edge of the island or is trying to dig into the floor of the island, the agent gets a heavy penalty applied to it. All other offences like colliding with a bush, colliding with a tree have smaller penalties because they just mean that the agent is trying to fly around and figuring out the environment and the task, and not just trying to avoid getting penalties by flying around but also not completing the tasks. Figures 8 - 10 show how the floating island looks when ready for the experiment, the four agents ready to start training and the flowers and obstacles on the island respectively.

Theoretically, the maximum award that can be received by the agent is 1500 units because there are 300 flowers placed on one island, and each flower can hold a maximum of 1 unit of nectar. The rule is that for each time step that the agent's beak tip is inside the nectar collider of the flower the agent will receive a reward of 0.1 units multiplied by the negative dot product of the direction of the beak tip and the nectar collider, which means that if the beak tip is going into the flower completely straight, then the reward will be multiplied by 1 hence maximum. The time step is updated 50 times each second.
The reward per time step is calculated by the equation:

$$reward = 0.1 \times -(X_{beak}, Y_{beak}, Z_{beak}) \bullet (X_{nectar}, Y_{nectar}, Z_{nectar})$$

Assuming we have a perfect alignment of the beak tip and the nectar collider of the flowers at all times, the dot product becomes 1 and the reward will be 0.1.

$$\implies reward_{max} = 300 \times 0.1 \times 50$$
$$\implies reward_{max} = 1500$$

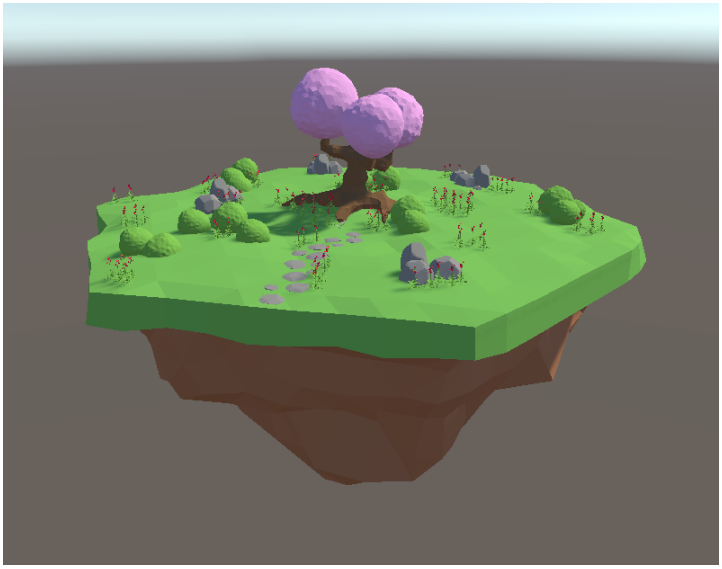

Figure 8: Island scene with agents

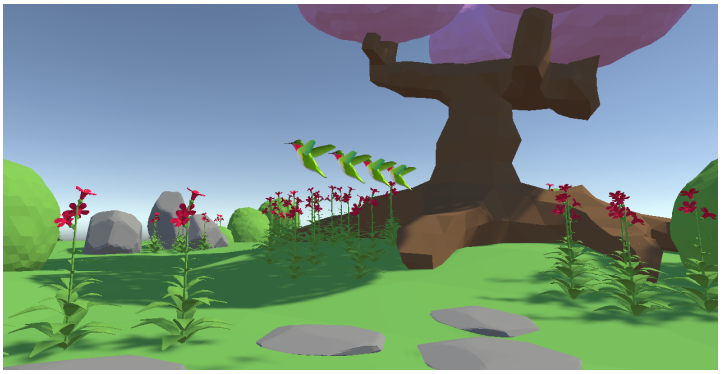

Figure 9: Hummingbird Agents on the Island

Figure 10: Flower objects on the Island

Agent name	Symbolic Obs.	No. Sensors	Perception System
HummingbirdSymbolic	10	3	Passive
HummingbirdHybrid1	10	1	Hybrid
HummingbirdHybrid2	6	1	Hybrid
HummingbirdActive	0	1	Active

Table 1: Information available to the agents

3.2.2 *Agent Design*

There are four agents being used in this experiment, each with their own set of perception system and different amount and forms of information available to them. I designed these agents in a way that emphasises an agents access to information about the environment where the agent at the lowest level has the highest access to symbolic information about the environment and as we go up the list of agents, the amount of information that an agent can access reduces gradually until we reach the top most agent that has no access to the symbolic information about the environment and has to rely solely on the active vision system available to it. The tables 1 - 4 highlight the differences between the agents and the perception systems used by each of the agent.
Although the agents are being treated as separate entities and have different perception mechanisms, it is important to note that for this experiment to be a fair comparison of only the perception systems, the agents have to be completely similar. All the agents used in this experiment are derived from the same agent design and use the same mechanics for movement around the environment and the same the scripts as controllers. The difference in the scripts is only the *CollectObservations()* method that collects symbolic information from the environment and sends it into the neural network as input.

Obs	No. Obs	Symbolic	Hybrid 1	Hybrid 2	Active
Local Rotation	4	Yes	Yes	Yes	No
Nearest Flower	3	Yes	Yes	No	No
Distance to Flower	1	Yes	Yes	No	No
BeakTip near Flower	1	Yes	Yes	Yes	No
Pointing to Flower	1	Yes	Yes	Yes	No
Sensor Used		RayCast	RGB Camera	RGB Camera	RGB Camera

Table 2: Observations collected by agents

The observations that has been provided to each of the non-symbolic agents in not based on speculation but is particularly selected based on the fact that these observations are local to the agents. That is, the knowledge that a real hummingbird would have about its environment, like a real hummingbird would know its own local rotation (orientation), it would know that if it is pointing towards a flower or not, it would know whether the tip of its beak is pointing towards the inside of the flower. A real hummingbird would not know which flower from among a bunch of flowers is the closest one to its location, and does not have access to a vector pointing towards the nearest flower and it definitely does not know the distance form its own location to the flower. These are all symbolic observations that are only available in passive vision systems just like the perception oracle in (Terzopoulos and Rabie, 1995) and provided to the agents by the environment.
Keeping this thing in mind, the information available to the Hybrid2 agent in Table 4 has been made available.

4 PROJECT MANAGEMENT

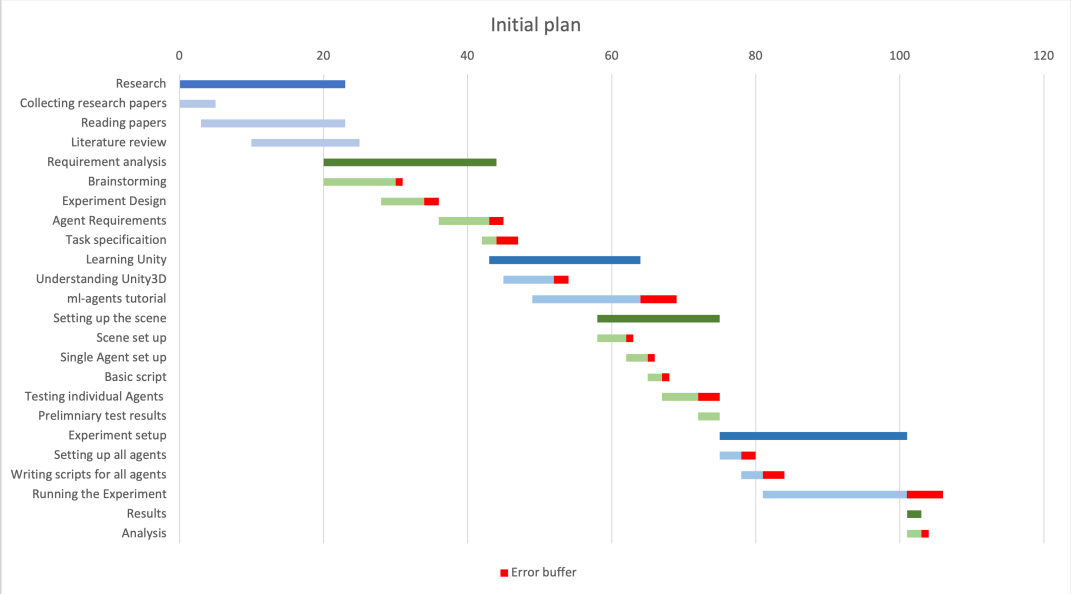

Figure 11: Initial plan for the project

To manage the project, the project was divided in to 6 sections and each sections had a few tasks to complete. The sections for the project are:

- **Research:** Reading the literature and understanding the basics of the differences between passive vision and active vision.
- **Requirement analysis:** Designing the experiment and analysing the tools currently available to run the experiment.
- **Learning Unity:** Learning to use and navigate through the unity interface.
- **Setting up the scene:** Setting up the scene and following the ml-agents tutorial
- **Experiment set up:** Setting up the different agents in the experiment and the different perception systems.
- **Running the experiment**

Figure 11 shows the initial plan that was to be followed, since I was going to have to learn how to use unity and the ml-agents library, I was expecting to run into some errors. I figured out the different

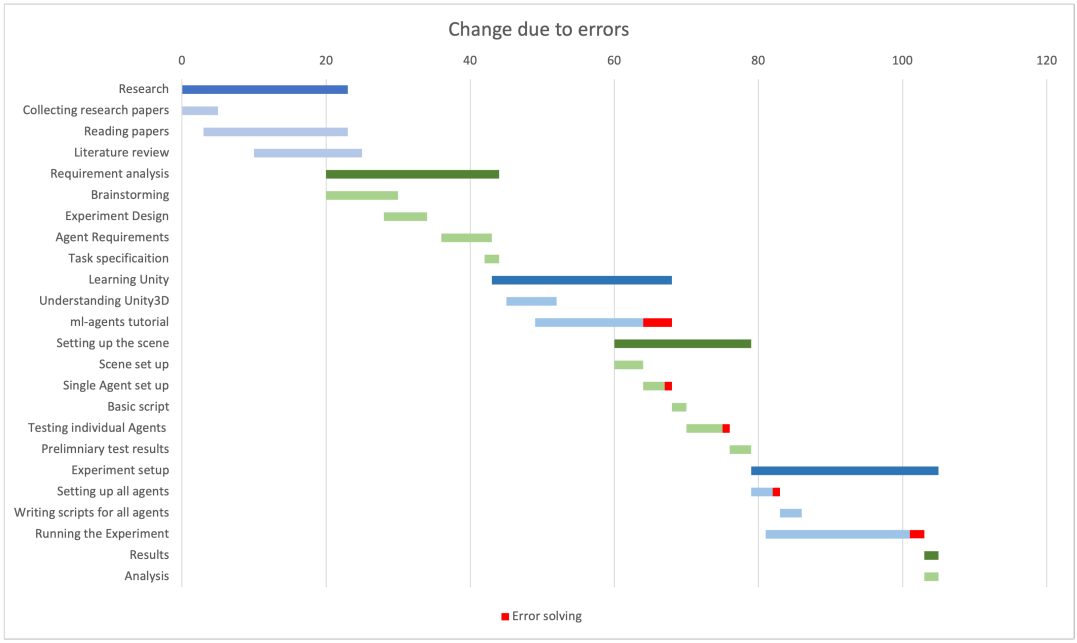

Figure 12: Changed plan after running into errors

stages where I thought that I will get stuck, and left a few days as error solving days. In the image, the error buffer days are highlighted in red. However after going through three phases – Research, Requirement Analysis and Learning Unity and ml-agents – I had used 3 days in error solving and getting the compatible versions of ml-agents C# and ml-agents python set up properly [Figure 12] and I had then kept only a few days for any more errors.
In the middle of August, I had to submit some coursework so I had to stop working for 2 weeks so the plan had to be changed again to account for the missing two weeks. It was because of these two weeks that I had to ask for an extension to submit the dissertation thesis. The final plan of the project is shown in Figure 13.

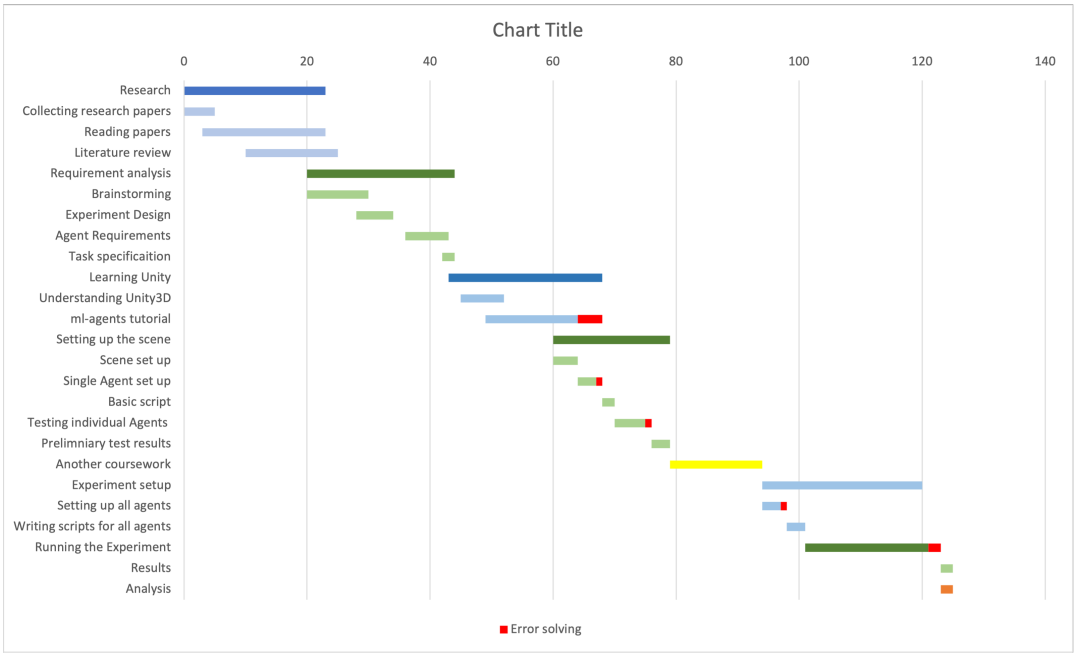

Figure 13: Final project plan with 15 days off for coursework

Part II

IMPLEMENTATION

Based on the detailed concept and design of what the experiment is going to look like, in this part of this dissertation thesis, I walk through the implementation of the experiment in Unity3D using ml-agents and explain how the agent scripts are programmed, trained and deployed in detail.

5

PREREQUISITES

In this part of the dissertation thesis, I go into details of the implementation of this project and how true to design the implementation is. I will start with setting up the Anaconda environment for training the agents to setting up ml-agents and lead into designing and implementing the scripts for the individual agents.

5.1 ANACONDA

Anaconda (Anaconda, 2021) is a free for personal use platform that is used to streamline data science and machine learning applications using python and R languages. It comes with the necessary packages pre-installed and about 7,500 more packages can also be installed if and when needed. This project uses the anaconda platform to run TensorFlow python and ml-agents libraries and also to connect to the unity3D software via port 5004.

The steps to follow to configure anaconda for use with this project are:

- Download and install anaconda individual edition for your operating system from www.anaconda.com,
- Open the anaconda prompt, (in case of macOS or Linux distribution, it is the native terminal)
- Create a new Conda environment using

Listing 1: Create conda environment

```
1        conda create <env_name>
```

- To confirm that the environment has been created, run

Listing 2: List conda environments

```
        conda env list
```

This command lists all the available environments in anaconda. Make sure that the environment you created in the previous step is visible

- Activate the desired environment using the following command:

Listing 3: Activate conda environment

```
conda activate <env_name>
```

- Once the environment is activated, we need to install all the required packages and libraries from the anaconda prompt or anaconda navigator, the following libraries need to be installed:
  - Tensorflow: using anaconda navigator
    1. Open up the navigator window,
    2. Select the environment tab from the left pane on the window,
    3. Click on the environment name from the list
    4. From the drop-down list at the top, select the "all" option.
    5. Search the name of the package you want to install - TensorFlow if you do not have a CUDA enabled GPU or TensorFlow-GPU is you do have a CUDA enabled GPU.
  - Mlagents:
    1. Open up an anaconda prompt window,
    2. Activate the correct environment,
    3. Use the following command:

Listing 4: Install mlagents

```
python -m pip install mlagents==<version>
```

5.2 SETTING UP THE SCENE

This project uses the ml-agents tutorial from the unity-learn platform (Limit, 2021) for the assets and for the basic scene setup. The tutorial link provides the base scene for this project and all the meshes that have to populate the floating island on which the experiment takes place. The tutorial provides a single zip file that contains the .unitypackage file that can be imported into a new scene and set up. It is also possible to design a new scene for this project from scratch but that would include 3D modeling the meshes from scratch using a separate software like Blender3D or Autodesk Maya and would also need texturing. That is too time expensive, so I went with the assets provided with this tutorial. The steps to set up the scene are:

- Download the .zip file from the tutorial home page and extract the contents into a folder. The .zip file should contain a .unitypackage file(link to tut)

- Open Unity Hub on your computer
- Create a new project
- Use the Universal Render Pipeline template and choose a name for the project.
- Select a folder as the location of the unity project you are creating
- When the scene has loaded up in Unity, click and drag the package file extracted earlier into the project panel. We need to do this before deleting the previous scene because, in unity, you cannot delete a scene before adding a new one.
- You can now delete the default scene and in the assets folder, go into Hummingbird → scenes folder and double click on the floating island scene.
- The base scene with the flower island, flower plants, and one hummingbird should be ready to work on.

5.3 AGENT SETUP

Now, this project, as mentioned earlier will have 4 different types of hummingbird agents. One with completely symbolic vision, named hummingbirdAgentSymbolic, two with a mix of symbolic and active vision systems using an RGB camera names hummingbirdAgentHybrid1 and hummingbirdAgentHybrid2, and the only difference between the two hybrid agents being the number of symbolic observations each one receives and finally the hummingbirdAgentActive that does not get any symbolic data given to it and has to navigate the environment and complete the task using only the single RGB camera provided to it. In this section we will set up the agent scripts that materialise the differences between different agents and also make sure that other than the *CollectObservations()* method that decides what information the neural network gets as input, all the other methods and functions and capabilities of the four hummingbirds is exactly the same. But before we set up the agent scripts, we need to create the scripts for the island that defines the location and position rotation of the flowers and the agents and the script for flowers that controls the behaviour of all the individual flowers in the scene.

6

THE ENVIRONMENT AND METHODS

There are 6 scripts being used in this unity project which will are used to control the behaviours of the agents and how the agents interact with the environment and also the reward penalty system. Out of the 6 scripts, 4 scripts are for the four agents described above, one script is used to control the flowerArea, that is the floating island and the other one is the script for the flowers on the floating-island that controls the behaviour of the flowers during each episode.

6.1 FLOWERAREA.CS SCRIPT:

The FlowerArea.cs script is called at the start of every episode to set up the island environment and place the objects and agents. It contains the following methods that are executed in this particular order:

- ***Awake():*** The awake function is called at the start of each episode after the *start()* method and creates an empty arrayList for the flowerPlants, the flowers and for the dictionary of nectar colliders.
- ***resetFlowers():*** At the start of each episode, this method is called after the *Awake()* method to give random rotations to the FlowerPlants on the y-axis and some slight rotations on the lateral axes.
- ***findChildFlowers():*** This method recursively finds and returns a list of all the child flowers present on a particular instance of the FlowerArea and adds them to a flowerPlants GameObject ArrayList. This method also adds the nectar amount in each flower to a nectar collider dictionary along with the nectar collider to keep track of the amount of nectar in each of the flowers.

6.2 FLOWER.CS SCRIPT

The Flower.cs script contains the code to control the behaviour of the flowers once a new episode has started. This script first gets some vector value about the flowers like a) the vector pointing straight out of the flower, b) the vector containing the location of the transform of the flowers and c) a scalar value that is the amount of nectar present in the flower and a Boolean value that means whether the flower even has nectar in it or not. The most important method of the Flower.cs script is the `Feed()` method.

- ***Feed():*** This method is called to update the amount of nectar present in the flower when a hummingbird agent tries to drink the nectar from a flower. This method also keeps track of the amount of nectar in the flower and if the nectar amount drops below zero, the value is clamped.
  The control of the mesh collider [Figure 16] of the nectar and the colour of the flower is also controlled by this method such that if a flower has no amount of nectar left in it, the `Feed()` method disables the mesh collider for that particular flower so that it cannot be detected by the agents and also changes the colour of the flower from a bright red to a purple colour in order to be differentiated from flowers with nectar visually by the active vision agents.

  Figure 14 shows when the flowers are full, they are red in colour and Figure 15 shows when the flowers are empty, they change colour to purple.

Figure 14: Flowers when full

Figure 15: Empty flowers

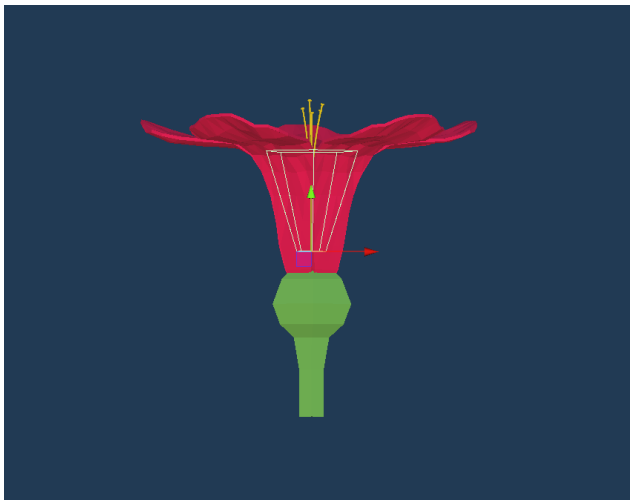

Figure 16: The Nectar Collider inside flowers

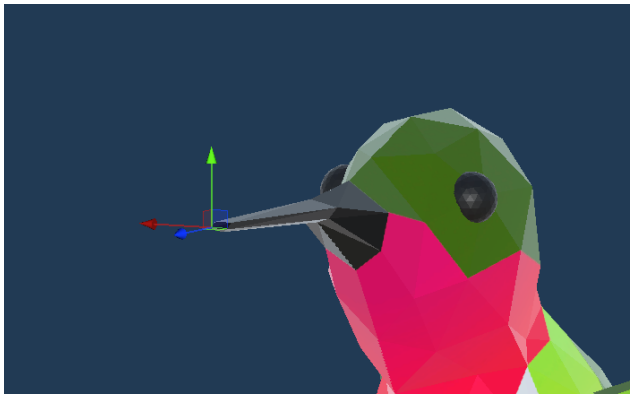

Figure 17: The beak tip of the hummingbird agents

6.3 HUMMINGBIRD AGENT SCRIPTS

There are 4 scripts, one for each of the four agents. These scripts control the perception system of the hummingbird agents and also the reward and penalty system along with the movement of the agents in the environment. Please note that all four scripts have the same skeleton structure, the same reward-penalty system, and even the same methods. The only difference is in the perception system and what data from the environment is available to each agent. The common methods used for all the agents are as follows:

- ***Initialize():*** Built-in function that we are overriding. The initialise method is called when the game starts and is used for the initial setup of the agents and provides the agents with the required child objects from the floatingIsland parent object.
- ***OnEpisodeBegin():*** Built-in function that we are overriding. This method calls the resetFlowers method in the flowerArea class,

sets the nectar that the agent has obtained in the previous episode back to zero, sets the moving and rotation velocity of the agents to zero, that is if the agents were moving when the last episode ended, this method stops the movement of the agent.

- ***OnActionReceived():*** This method determines the action to be taken by the agent when it receives an input form the neural network or a human player. The agent maps 5 inputs into an action vector array of length 5. The five indices of the array represent:
  - Index 0: move vector x (+1 means move to the right, -1 means move to the left)
  - Index 1: move vector y (1 means move up, -1 means move down)
  - Index 2: move vector z (+1 means move forward, -1 means move backwards)
  - Index 3: pitch angle vector (+1 means rotate pitch upwards, -1 means rotate pitch downwards)
  - Index 4: yaw angle vector (+1 means rotate towards the right, -1 means rotate towards the left)

- ***TriggerEnterOrStay():*** This method is called when the beakTip collider of the hummingbird agent collides with a trigger collider, which in this case, the nectar collider of the flower. This method first finds which flower is this from the nectarColliderDictionary so that it knows how much nectar is present in this particular flower, and if the flower has nectar and is not empty, proceeds to drink nectar from this flower using the `Feed()` method. The hummingbird agent drinks nectar from the flower at the rate of 0.1units per time update or 0.2 seconds and updates the amount of nectar taken by the hummingbird and reduces it from the available nectar in the flower.
  If the agent is in training mode and starts drinking nectar from a flower, then this method gives a reward of 0.5 units + a bonus amount to the agent.

- ***OnCollisionEnter():*** In case the agent collides with an object that is not a trigger, for example the agent collides with the tree or a bush or a rock object or even the boundary of the training scene, then the agent receives a penalty of 0.1 units.

The specialised methods made for each of the four hummingbird agents are different only when it comes to the `CollectObservations()` method. Each agent has different set of observations provided to it by this method discussed below.

6.3.1 *HummingBirdSymbolic.cs - passive perception system*

This script is called by the hummingbird agent that takes symbolic input form the environment along with a rayCasting sensor mounted on the agent.

CollectObservations(): In the symbolic agent, the CollectObservations collects 10 observations from the agent and the environment and send them into the neural network to process and give an output.
The observations collected are a mix of local observations like the location and rotation and external observations like position and rotation of flowers. The observations collected by the symbolic agent are:

- The normalized local rotation of the agents' 3 axes. - 4 observations.
- A normalized vector pointing to the nearest flower - 3 observations.
- A normalized vector telling whether the agent's beak is pointing towards the flower or not. - 1 observation.
- A normalized vector telling whether the flower is on its front or at its back. - 1 observation
- And finally a ray casting sensor which acts like a LiDaR sensor which detects whether there is any object in front of the agent up to a certain distance.

Since we are using a RayCasting sensor along with the symbolic observations in this agent, it can be argued that this agent uses a mix of a symbolic perception system and a passive perception system.

1. ***RayCast Sensor***
   The RayCast sensor used with the symbolic agent has 3 layers of rays originating from the same point and ending at an offset of $\pm 45°$ form the central rays. The maximum angle of the rays is set as $60°$ because the field of view of the cameras we have set up is also $60°$. The maximum length of the rays that the sensor emits is 10 units. This means that the RayCast sensor can detect objects located up to 10 unit distance from the origin point. Figure 18 shows a single RayCasting sensor mounted on a hummingbird. The passive agent, in this experiment is using 3 RayCast sensors. The sensors have the same origin point, but the end points are set at an offset of $-45°$, $0°$ and $+45°$.
   The RayCase sensors can detect if an object is placed in front of it, if a ray emitted form the sensor hits the object. The sensor will detect the object only if it is in the range of the sensor —in

this case, 10 units —and any object beyond the range will not be detected. It can also detect that an emitted ray has hit a specific object if we set tag detection to on and specify the number of objects and the tags associated with those objects that it has to detect. Figure 19 shows the top view of the RayCast sensor mounted on the object when the agent is placed in the environment. When a ray hits an object, the colour of that particular ray changes to Red, and if he ray misses any object, the colour remains white.

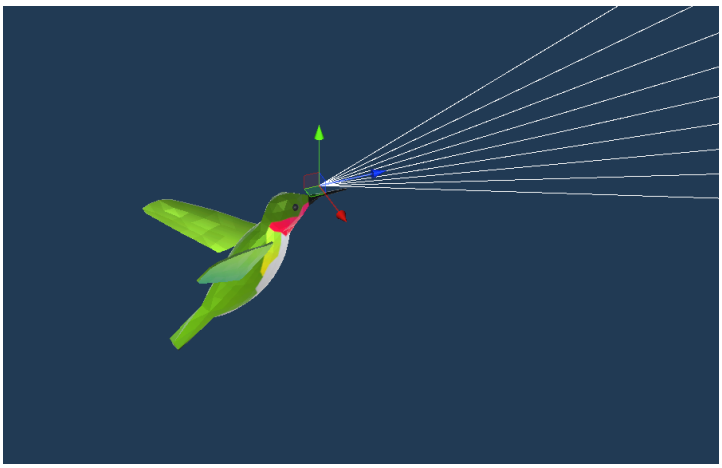

Figure 18: single RayCast sensor mounted on a hummingbird agent

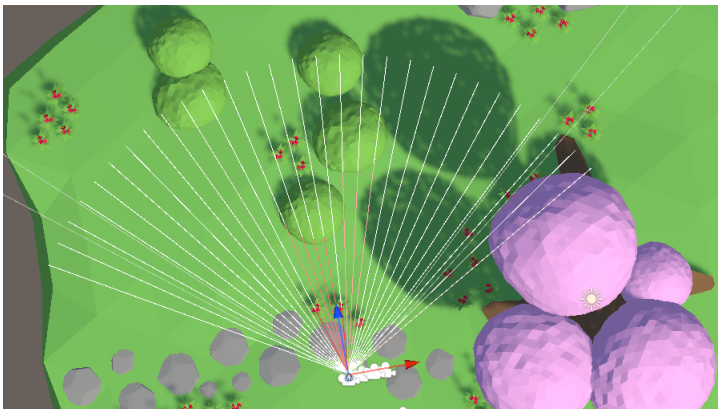

Figure 19: Top view of the RayCast sensor in the environment.

6.3.2 *HummingBirdHybrid1.cs - Hybrid perception system 1*

This script is used by the agent that uses a hybrid perception system that is a hybrid between symbolic perception and active perception.

CollectObservations(): In this agent, this method also collects observations form the environment, and instead of using just a RayCast sensor that emits radiation into the environment, this agent has an

RGB camera mounted on it that acts as it's eye and since the camera does not emit any radiation into the environment, but only receives information in the form of pixels, this perception system can be called a hybrid between symbolic and active perception.

6.3.3 *HummingBirdHybrid2.cs - Hybrid perception system 2*

The agent that calls this script again uses a combination of both the symbolic and active perception systems. This agent does receive some observations from the environment but those observations are strictly limited to the agent itself and are observations that I as a human would have about myself if I were in place of that agent.

CollectObservations(): The `CollectObservations()` method in this script collects 6 observations about the agent. These 6 observations collected have nothing to do with the environment and have been carefully chosen to be focused on the agent. The following observations are provided to this agent:

- Normalised local rotation of the agents 3 axes - 4 observations.
- A normalised vector telling whether the agent's beak is pointing towards the flower or not. - 1 observation.
- A normalised vector telling whether the flower is on its front or at its back. - 1 observation.
- And an RGB camera sensor placed on the hummingbird agent.

This agent also uses an RGB camera in the place of a RayCast sensor that acts as a passive sensor required for active sensing.

6.3.4 *HummingBirdActive.cs - Active perception system*

This script is called by the agent that uses the RGB camera mounted on the agent as the only source of observations. The hummingBirdAgentActive agent in this environment is the only agent with a completely active perception since the only way for it to gather any information about the environment is through the raw pixels it receives from the camera.
The script used by this agent does not have any `CollectObservations()` method in it.

6.4 CAMERA SENSOR

For implementing the active vision part of this project, I have used the inbuilt Unity camera-sensor that is mounted onto the agents as a child sensor of the hummingbird agent prefab. The camera sensor was set up using the following steps:

- Open the prefab of the agent you want to add the camera to,
- Right-click on the left panel and add a new camera
- Change the camera name from the inspector window for the appropriate agent, I used the names cameraSensorHybrid1, cameraSensorHybrid2, and cameraSensorActive for each of the three agents.
- Now, we need to create a new component by clicking on the "add component" button in the inspector window and add a new camera sensor component for each of the agents.
- Drag the camera object from the left pane into the camera field of the cameraSensor component.

The resolution that I have chosen for the cameras in this project is 300x150 pixels. I have chosen a 2 : 1 aspect ratio because it provides a wider field of view to the agents hence covering a larger area of the island. I could have chosen a less wide aspect ratio and increased the field of view in the camera settings, but if we set a larger field of view for the camera, then the image gets distorted Figure 20 also known as perspective distortion. For this reason, a wider aspect ratio is used and the field of view for the cameras is set to 60° We also have to keep in mind the resolution of the cameras because larger images of a complex environment contain more detail, hence a larger network is required to extract information form them. Now since larger networks take a larger time to train, we have to choose the resolution in a way that does not affect the performance of the agents as much while still having enough information to be able to complete the tasks easily.

A camera equipped with a cameraSensor component is mounted on all the agents at the same relative location to maintain the uniformity across all the agents. Figure 21 to 24 show the mounting of the camera and the view seen by the hummingbird agents when in the environment respectively. Figure 25 shows the range up to which the agents can see. The clipping planes for the cameras range from 0.01 to 20 that means that objects between the distance of 0.01 units and 20 units can be seen by the agent.

In order to save time and processing power required, the data fed into the camera sensor was further simplified by using culling masks. The nectar colliders fo the flowers are rendered onto a separate layer along with the beak tip of the agent over a solid background. The camera settings are changed to feed the neural network with only the images rendered on this layer. so although the perspective of the agent is shown in Figure 24, the rendered image that is fed into the neural network is shown in Figure 26. This method of simplifying the output image will help in reducing the training time of the agents by a very large margin.

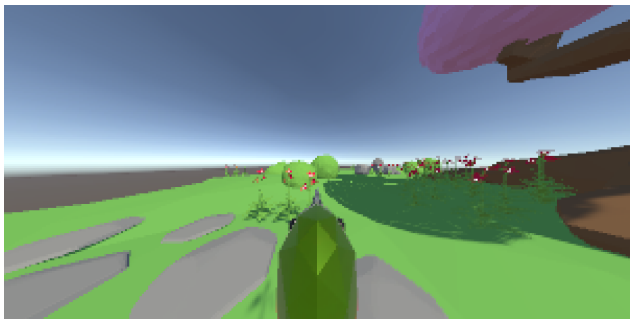

Figure 20: The distorted view seen in the camera at wider field of views

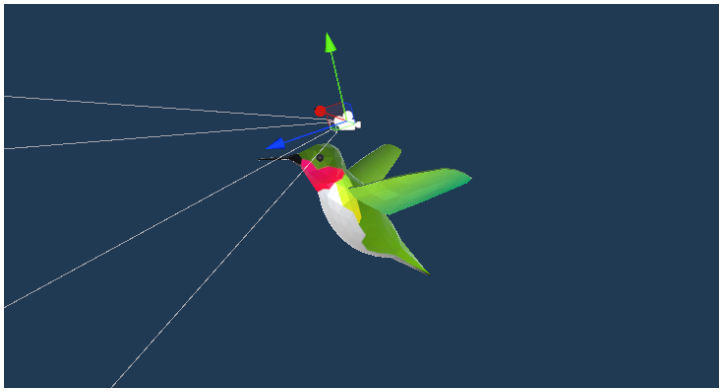

Figure 21: The mounting of the camera sensor on the agents

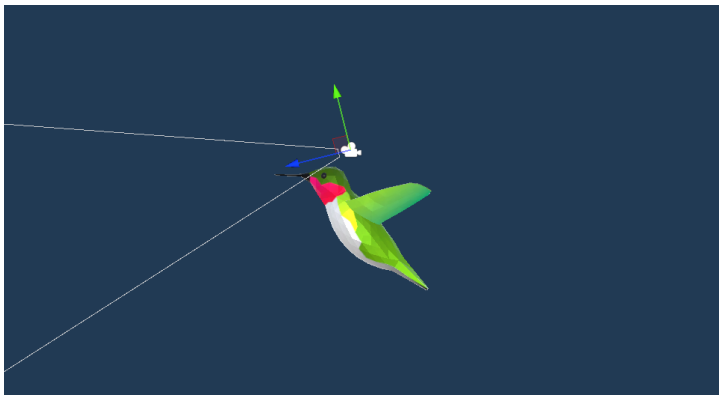

Figure 22: The mounting of the camera sensor on the agents - side view

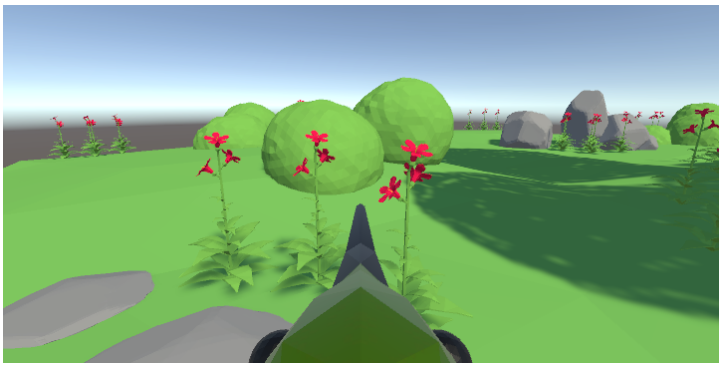

Figure 23: View from the camera

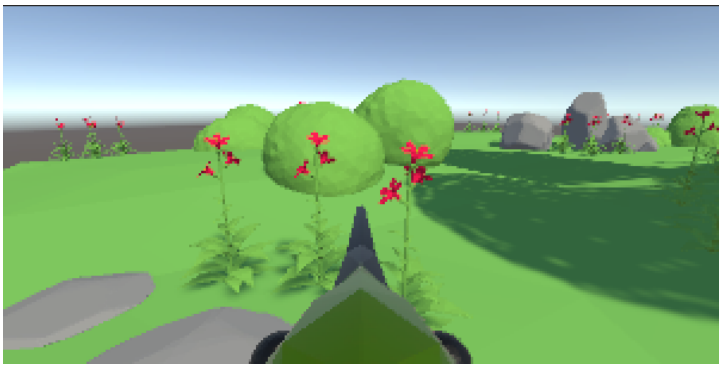

Figure 24: The actual 300x150 view of the environment as seen by the agents

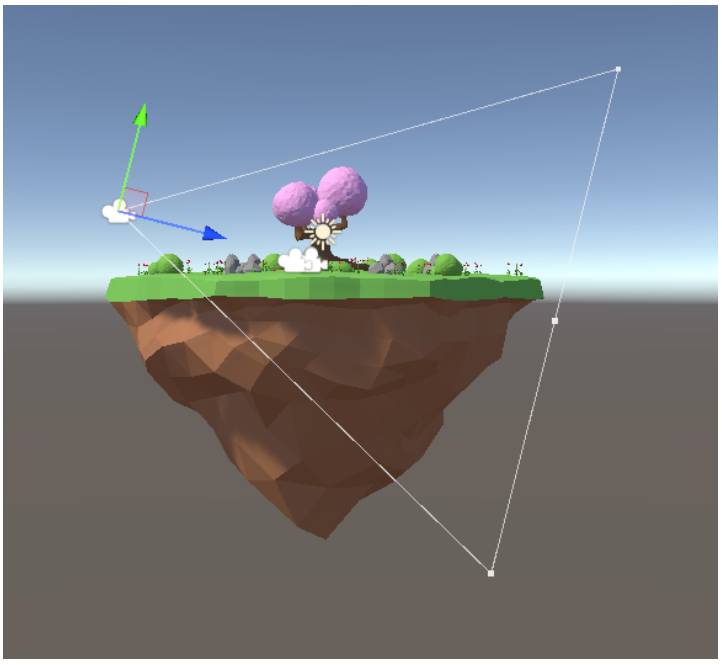

Figure 25: Clipping distance for the cameras

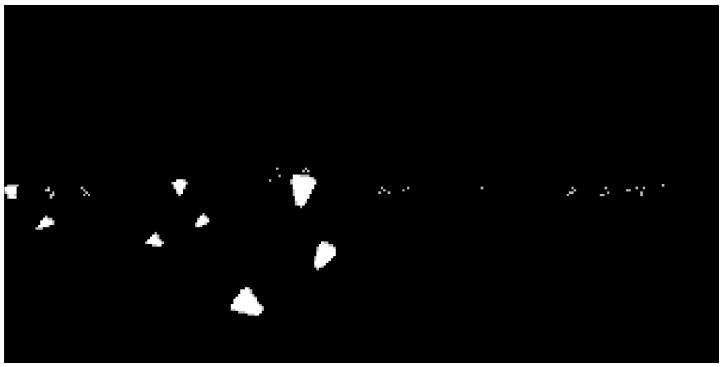

Figure 26: Culled image for the neural network

7 TRAINING

7.1 TRAINING USING ML-AGENTS

Training these agents takes place using the ml-agents framework. We already have set up ml-agents in unity as well as in the conda environment and are ready to start training the agents. To train each agent, we need to use a .yaml file that contains the hyperparameters for the neural network and the environment parameters. .yaml is a data serialisation format that can easily be read by humans. This format is used to store information about the parameters inputs required by a certain program and is independent of the programming language used. The parameters and hyperparameters set in the trainer configuration file for this project contain the information required by the neural network to perform the reinforcement learning process like the number of hidden units, the number of epochs to run when training, the number of hidden units in each hidden layer, the strength of the reward signal and also the discount factor. For this experiment to be a fair comparison, I had intended that all agents use the same architecture of the neural network and the only difference between the symbolic agent and the non-symbolic agents is that the non-symbolic agents use a convolutional network in front of their own neural network, but because the inputs and the dimensions of the data being input into the neural network are completely different between agent, it was extremely hard to find a set of hyperparameters that were optimised for all the four agents simultaneously. In the initial tests, I found that if one agent was performing better, the others were not performing at all until the very end of 10 million steps. I then decided to run the experiment with different network architectures for each agent. I selected sets of hyperparameters for each agent which I knew are tuned for that particular agent and ran the experiment.
Table 3 contains a list of the hyperparameters used by the each agent.

Agent name	No. layers	Hidden units	η	Batch size	epochs
HummingbirdSymbolic	5	256	0.0002	1024	3
HummingbirdHybrid1	5	256	0.0002	500	3
HummingbirdHybrid2	3	512	0.0003	2048	3
HummingbirdActive	1	2048	0.0003	10	3

Table 3: Hyperparameters for each agent

An example of the contents of the training configuration is:

Listing 5: .yaml file contents

```
behaviors:
AgentName:
trainer_type: ppo
hyperparameters:
      batch_size: 2048
      buffer_size: 10000
      learning_rate: 0.0001
      learning_rate_schedule: linear
network_settings:
      normalize: false
      hidden_units: 64
      num_layers: 1
      vis_encode_type: resnet
reward_signals:
      extrinsic:
            gamma: 0.99
            strength: 1.0
keep_checkpoints: 5
checkpoint_interval: 500000
max_steps: 5000000
time_horizon: 128
summary_freq: 20000
threaded: true
```

Figure 27 shows the camera views of all four agents training simultaneously in the same environment.

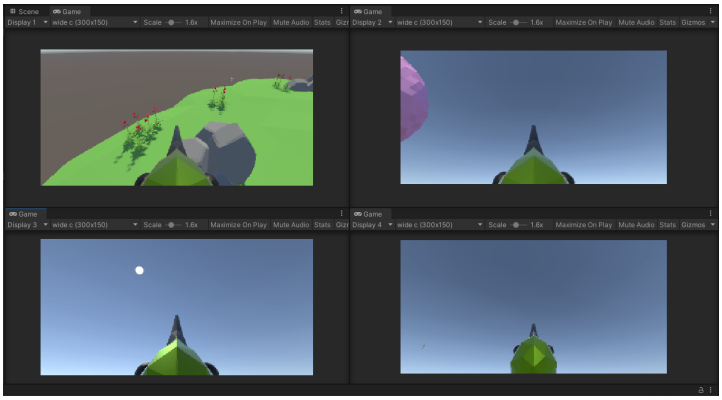

Figure 27: All agents training simultaneously

Use the following steps to start the training:

- Open an anaconda prompt window,
- Activate the environment that you want to use to train the hummingbirds,

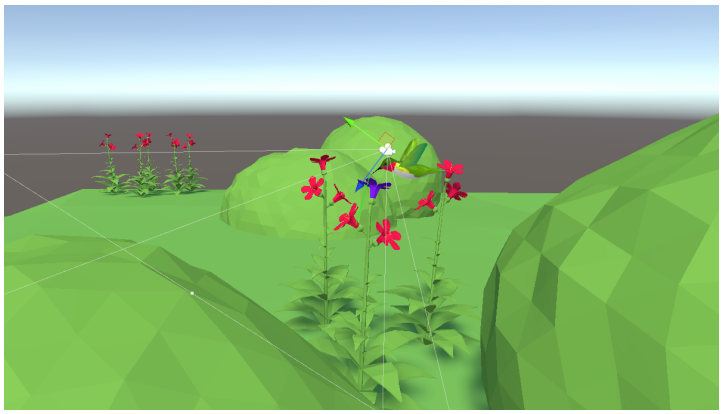

Figure 28: The active agent in training

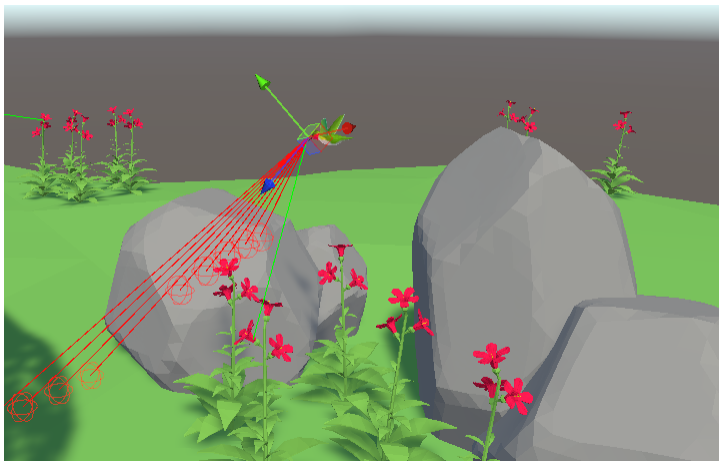

Figure 29: The passive agent in training

- Navigate in the anaconda prompt to the folder you want to save the results in. Ideally the same folder with the training configuration file.
- Call the ml-agents python to train the agents using TensorFlow using the command:

Listing 6: Training Command ml-agents

```
mlagents-learn <training configuration file> --run-id <name
    of run>
```

Figure 28 and Figure 29 show the camera views of the agents when in training and Figure 30 shows the view form the agent camera when the agent is training.

The results of the training are written by ml-agents in a folder named `results` that is located in the same folder as your training configuration file. This folder contains sub-folders with names of all the training runs and each such folder has a separate folder for ev-

Figure 30: Agent's perspective when training

ery agent in that run. We can view the results of the training using TensorBoard.

7.2 TENSORBOARD

According to the configuration set up in the trainer configuration file, the ml-agents will print out a summary of the training results so far after every 20000 steps until it reaches 5 million steps of training for each agent. The summary is also added to a .csv file along with the neural network in .nn file format that can be imported into unity as a trained brain to be used by the agents.
We can see the training in action using TensorBoard. TensorBoard is a toolkit for TensorFlow used in visualisation of the experiments in real-time. TensorBoard can be used to visualise the results using the following steps:

- Open an anaconda prompt window,
- Navigate to the folder with the results folder,
- Run the command:

Listing 7: Run tensorboard

```
tensorboard --logdir results
```

- Open a browser window,
- Go to the url https://localhost:<portnumber> you can see the port used by TensorBoard in the anaconda prompt window.

Part III

EVALUATION AND RESULTS

This part of the dissertation thesis discusses the results of this experiment. I first put forward a test hypothesis based on what I think could be the possible result of the experiment and after that I will discuss the actual results of the experiment followed by an evaluation of the results obtained and where do we go from here.

8

HYPOTHESIS

The results of the preliminary experiments suggest that although all the agents can perform well and learn how to drink nectar from the flowers given each agent has a different network architecture and hyperparameters, all four agents take different number of iterations to learn the winning criteria. I also learned that different agents need different amounts of rewards for learning at a better or a worse rate. One thing that remained fairly consistent was that the agent with hybrid1 perception system, that is, the agent that received all 10 observations form the environment along with the camera input learned to locate the flowers and drink nectar form them the earliest while the agent with the active perception system that only received the camera feed as input took the longest time to learn to drink nectar from the flowers and that too after tweaking the reward system and letting the agent have a slightly more reward when it drinks nectar from a flower, giving it more incentive to learn quickly. The agents with the active vision system and hybrid 2 perception system were in a quite close competition with each other, but the symbolic agent did beat the hybrid 2 agent in the time it took to learn to drink nectar form the flowers. Based on these experiments, I have developed a hypothesis stated in the next section.

8.1 HYPOTHESIS 0

The hypothesis that I propose for this experiment is that under completely identical situation for all the agents, and after selecting network settings with which all the agents have a chance of learning how to complete the task and get consistent substantial rewards in a reasonable time, the decreasing order in which the agents perform best will be:

1. **Hybrid 1 perception system** *10 symbolic observations along with an RGB input from the camera fed into the neural network*
2. **Passive perception system** *10 symbolic observations along with input from the RayCast sensor*
3. **Hybrid 2 perception system** *6 symbolic observations collected from the environment along with the RGB camera feed as input to the neural network*
4. **Active perception system** *No symbolic information available to the agent and only the camera output fed as input to the network.*

9 THE RESULTS

After completing the training run on all the four experiments separately, Figure 31 shows the rewards received by each agent and Figure 32 shows the rewards along with the average reward received by each agent. The individual rewards received by each agent are shown in images 34 through 37. According to these results, the order of the agents who got the maximum amount of rewards to the lowest is:

1. **Passive perception system**
2. **Hybrid 1 Perception system**
3. **Active perception system**
4. **Hybrid 2 perception system**

These first two items in the list conform to the hypothesis put forward initially, however the last two items do not satisfy the hypothesis so we can say that the hypothesis was partially correct. Entropy of an agents' training run means the lack of predictability of an agents actions, in other words, randomness. Therefore to establish that these agents have actually learnt at-least some pattern of actions to complete the task, we will also be looking at the entropy for all the agents. In Figure 33, we can see that the entropy for all the agents is decreasing and is suggesting that the randomness of the agent actions is decreasing and each agent is falling into a pattern of its own.

Colour	Perception system	Marker
Red	Passive	–
Green	Hybrid 1	–
Orange	Hybrid 2	–
Blue	Active	–

Table 4: Legend for Figure 31

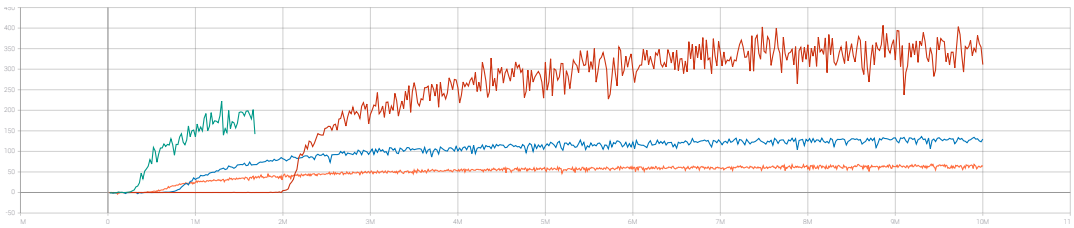

Figure 31: Cumulative reward of all four agents

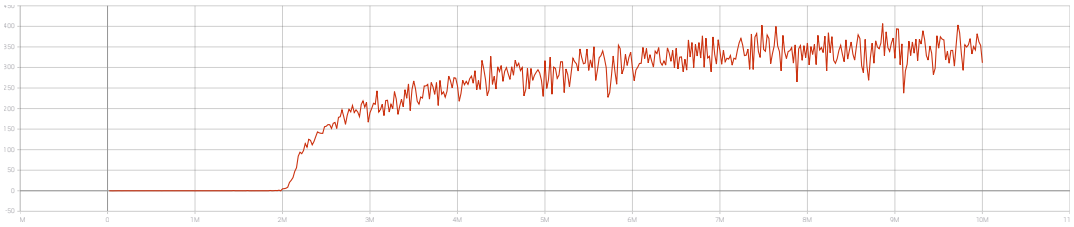

Figure 34: Rewards for passive perception system

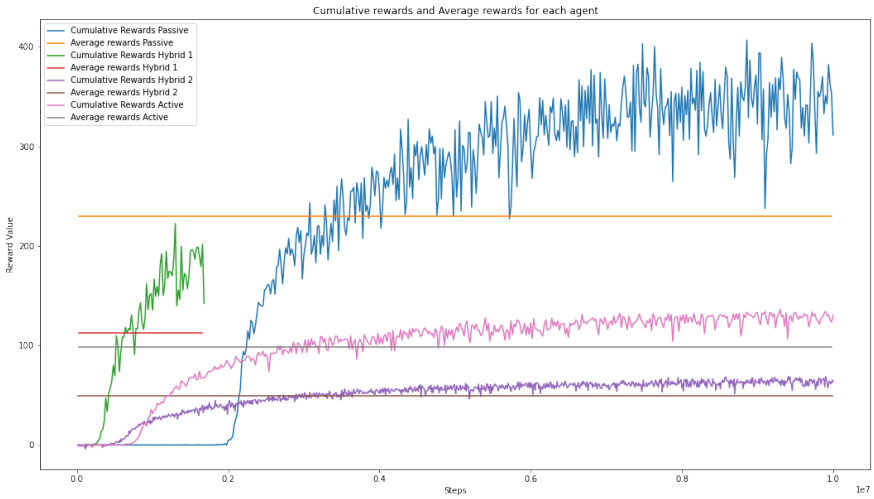

Figure 32: Average reward of all four agents

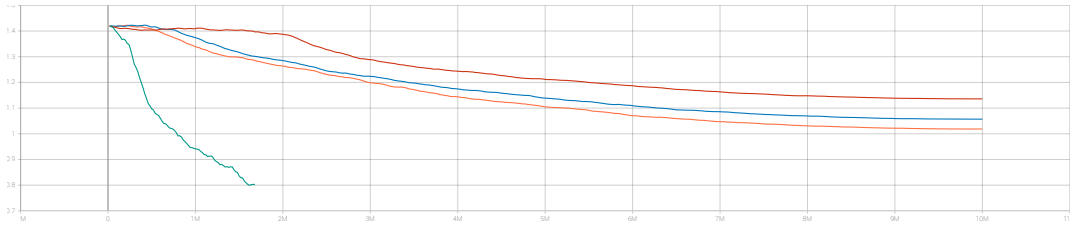

Figure 33: Entropy for all agents

In Figure 33, we can see that the entropy for all the agents is decreasing and is suggesting that the randomness of the agent actions is decreasing and each agent is falling into a pattern of its own.

9.1 ANALYSIS ON THE RESULTS

Now that we have established the results of this experiment, I analysed the result data and pass the data through some data pipelines to see if there are any hidden patterns in the result file. To develop a data pipeline for the result file, we first have to look at the data and

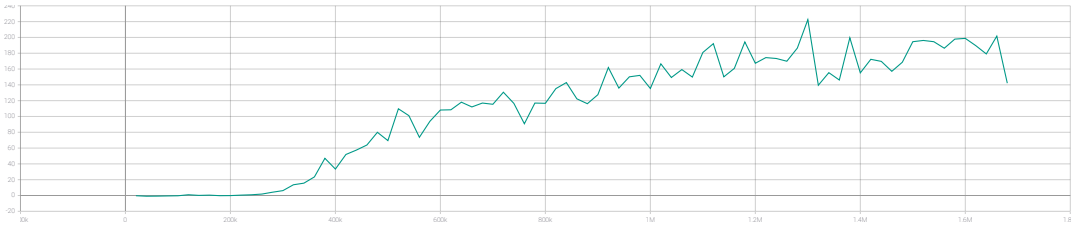

Figure 35: Rewards for Hybrid 1 perceptionsystem

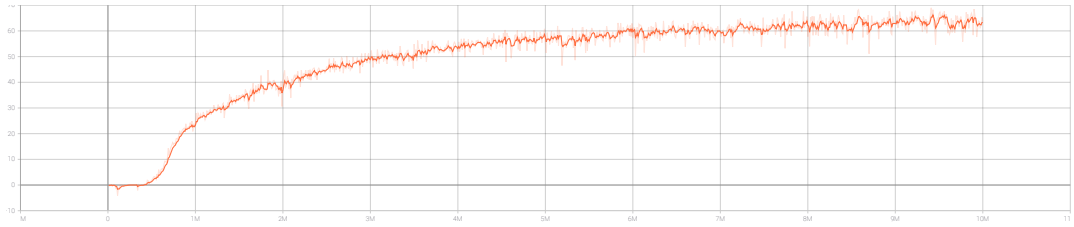

Figure 36: Rewards for Hybrid2 perception system

understand what it contains. The results are exported by ml-agents in a .csv file and can be read by Tensorboard to create the result visualisations.

9.2 SUMMARY

Figure 38 shows the average rewards gained by each agent. The maximum rewards were received by the Agent with passive perception system with the average being over 250 units. there was a significant difference in the rewards received by the other agents. The second highest amount of rewards was received by the hybrid 1 perception system at 120 units closely followed by the Active perception system at 100 units. The least amount of rewards were received by the agent with the hybrid 2 perception system at 50 units.

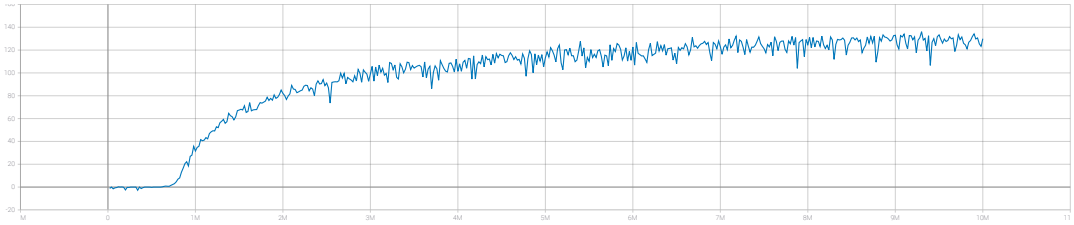

Figure 37: Rewards for Active Perception system

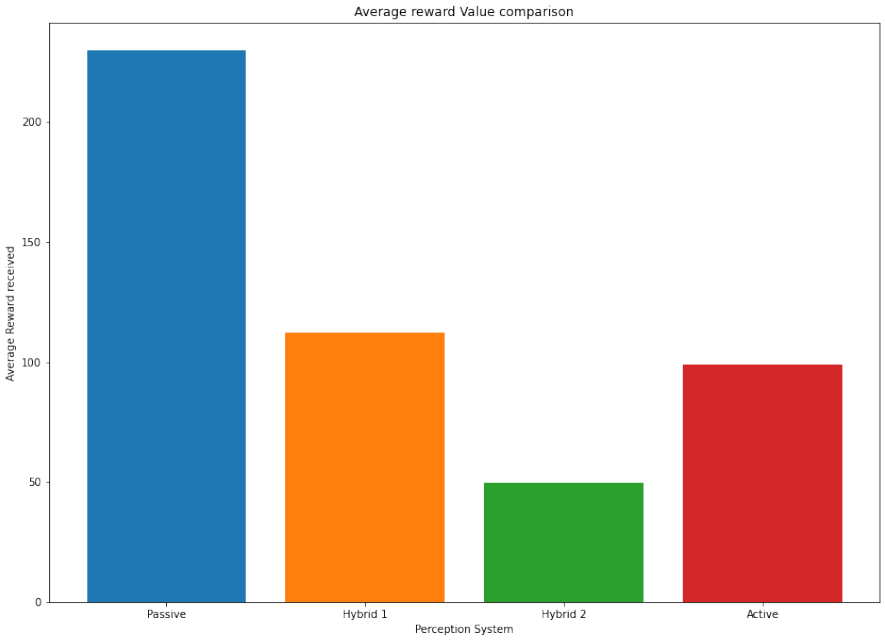

Figure 38: Average rewards for each agent

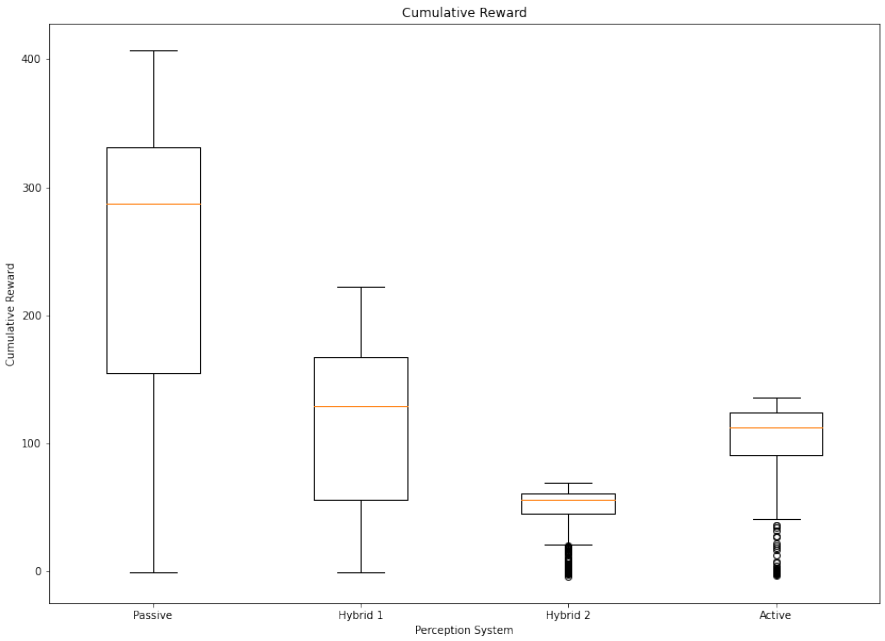

Figure 39: Cumulative rewards for agents

Although the Hybrid 1 perception system was the second best performing perception system, in Figure 40 and Figure 33 we can see that the Hybrid 1 system has the lowest median entropy among all the four systems and also has takes the lowest number of steps to reach the point where it receives constant positive rewards.

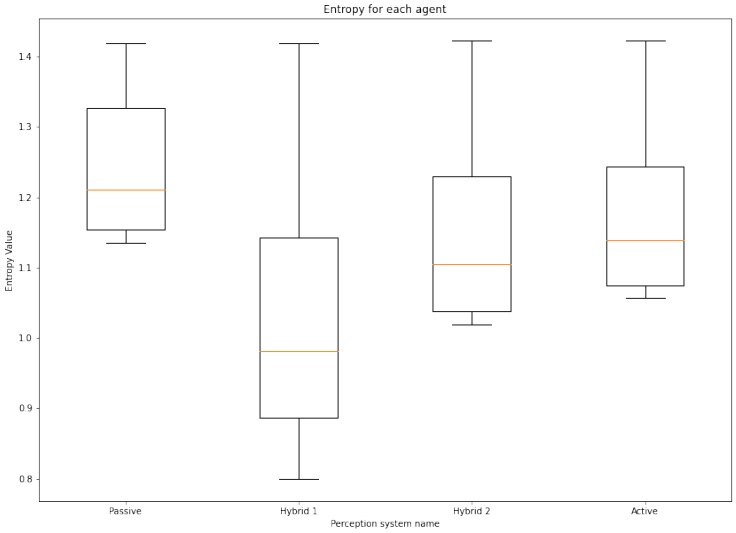

Figure 40: Entropy for each agent.

10

CONCLUSION

After looking at the results and analysis, it can be concluded that given each agent is provided with a separate set of hyperparameters tuned to the agents and their specific input to the network, there are two candidates for the best performing agent depending on the criteria that we are judging on.

If we are going to judge these agents with the only criteria being the amount of rewards received by the agents, then the clear winner is the passive perception system which leads the other agents by more than twice the amount of average rewards received but we can also not ignore the fact that the passive perception system has the highest entripy value and therefore it is most difficult to predict what the agents' next action will be.

However, if we also take into consideration the number of steps an agent takes to receive a constant positive reward and how quickly the agent learns to perform the task correctly along with the maximum, then the clear winner is the Hybrid 1 perception system because in this experiment while the passive perception system took about 2 million steps to reach constant positive rewards, the Hybrid one perception system was the earliest to learn the task properly with only 200,000 steps taken to start getting a constant positive reward. The hybird 1 perception was followed closely by the hybrid 3 perception system, but the average reward received by the hybrid 2 agent is the lowest so it is losing.

This experiment is still left open ended because the architecture of the neural network for each agent is different, so it could not be determined which perception system performs the best if all the agent have the exact same conditions and network architecture. What this experiment proves is that if we are allowed to fine tune the neural network according to each agent, although a passive perception system gets the most reward, it takes a long time to learn to complete the task. The hybrid 1 perception system on the other hand, where the agent has access to all the information available from the environment, if the agent is also provided with active perception capabilities, it can learn to perform the task fairly quickly and with a relatively higher average reward.
I think this is because in the passive perception case, although the symbolic information available to the agent does point to the nearest

flower and also tells the distance to the flower, the agent still has to navigate correctly to the flower and find the nectar collider and the entrance. However in the case of the hybrid perception system, once the agent has the information about the nearest flower provided to it, the task at hand for the agent changes from drinking nectar to maximising the red colour on the camera sensor (maximising the colour of the flower means that the flower is nearby). The agent then has to only choose any white coloured on the screen and move in such a way that the camera sensor has maximum white pixels rather than black.

What it means for agents deployed in a 3d environment along with non-deterministic actors like humans or other agents is that if there is perfect knowledge of the environment available to the agents along with perfect knowledge of the presence of other agents also present in the environment, a passive perception system, although will take a larger number of steps to perform the task at hand, passive perception will be the best performer when it comes to the average reward received as a metric. However, if we are low on time, then an agent with a perception that is a hybrid between active and passive perception will be the best performer because it takes the lowest number of steps to reach a state of constant positive rewards and the average rewards received is also the second best trailing only the passive perception system.

Part IV

APPENDIX

A SOURCE CODE

This section contains the source code for the different scripts used by the FlowerArea, The individual flowers and by each of the humming birds.

A.1 FLOWER AREA SCRIPT

A.1.1 *The Reset Flowers method*

Listing 8: resetFlowers() method

```
public void resetFlowers()
    {
        //rotate each flower plant randomly around Y axis and
            slightly along the lateral axes
        foreach (GameObject flowerPlant in flowerPlants)
        {
            //set random rotation
            float xRotation = UnityEngine.Random.Range(-5f, 5f);
            float yRotation = UnityEngine.Random.Range(-180f, 180
                f);
            float zRotation = UnityEngine.Random.Range(-5f, 5f);
            flowerPlant.transform.localRotation = Quaternion.
                Euler(xRotation, yRotation, zRotation);
        }

        //reset each flower
        foreach (Flower flower in Flowers)
        {
            flower.resetFlower();
        }
    }
```

A.1.2 *Finding child flowers on the island*

Listing 9: findChildFlowers() method

```

private void findChildFlowers(Transform parent)
    {
        for (int i = 0; i < parent.childCount; i++)
        {
```

```
            Transform child = parent.GetChild(i);

            if (child.CompareTag("flower_plant"))
            {
                // found a flower plant; add it to flowerPlants
                    list
                flowerPlants.Add(child.gameObject);

                //look for flowers in flower plants
                findChildFlowers(child);
            }
            else
            //not a flower plant, look for flower component
            {
                Flower flower = child.GetComponent<Flower>();
                if (flower != null)
                {
                    //found flower; add it to flowers list
                    Flowers.Add(flower);

                    //add nectar collider to lookup dictionary
                    nectarFlowerDictionary.Add(flower.
                        nectarCollider, flower);

                    //note: there are no flowers that are
                        children of other flowers.
                }
                else
                {
                    //Flower component not found
                    findChildFlowers(child);
                }
            }
        }
    }
```

A.2 SCRIPT FOR EACH FLOWER

This section contains the code for the ***Feed()*** method for each of the flowers.

Listing 10: Feed() method

```
public float Feed(float amount)
    {
        //track how much is available. (cannot taker more than
            available
        float nectarTaken = Mathf.Clamp(amount, 0f, nectarAmount)
            ;

        //subtract the nectar
```

```
        nectarAmount -= amount;

        if (nectarAmount <= 0)
        {
            //no nectrar remaining
            nectarAmount = 0;

            //disable flower and nectar collider
            flowerCollider.gameObject.SetActive(false);
            nectarCollider.gameObject.SetActive(false);

            //change flower colour
            flowerMaterial.SetColor("_BaseColor",
                emptyFlowerColor);
        }
        //return amount of nectar taken
        return nectarTaken;
    }
    }
```

A.3 THE HUMMINGBIRD AGENT SCRIPTS

As stated earlier in the thesis, the 4 agents have the same code in their scripts except the ***CollectObservations()*** method. The common methods in all the scripts are:

A.3.1 *Common methods:*

1. OnEpisodeBegin():

Listing 11: OnEpisodeBegin() method

```
public override void OnEpisodeBegin()
    {
        if (trainingMode)
        {
            //only reset flowers in training when there is
                one agent per area
            flowerArea.resetFlowers();
        }

        //reset the amoount of nectar obtained
        nectarObtained = 0f;

        //zero out velocities so that there is no movement
            when a new episode begins
        rigidbody.velocity = Vector3.zero;
        rigidbody.angularVelocity = Vector3.zero;

        //default to spawning in fornt of a flower
```

```
            bool inFrontOfFlower = true;

            if (trainingMode)
            {
                //spawn in front of flower 50% of the time.
                inFrontOfFlower = UnityEngine.Random.value > .5f
                    ;
            }

            //move agent to random safe position
            MoveToSafeRandomPosition(inFrontOfFlower);

            //recalculate nearest flower since agent has moved
            UpdateNearestFlower();
        }
    }
```

2. OnActionReceived():

Listing 12: OnActionReceived() method

```
    public override void OnActionReceived(float[] vectorAction)
        {
            //Don't take any action
            if (frozen) return;

            //calculate movement vector
            Vector3 move = new Vector3(vectorAction[0],
                vectorAction[1], vectorAction[2]);

            //add force in the direction of the move vector
            rigidbody.AddForce(move * moveForce);

            //get the current rotation
            Vector3 rotationVector = transform.rotation.
                eulerAngles;

            //calculate pitch and yaw rotation
            float pitchChange = vectorAction[3];
            float yawChange = vectorAction[4];

            //calculate smooth rotation changes
            smoothPitchChange = Mathf.MoveTowards(
                smoothPitchChange, pitchChange, 2f * Time.
                fixedDeltaTime);
            smoothYawChange = Mathf.MoveTowards(smoothYawChange,
                 yawChange, 2f * Time.fixedDeltaTime);

            //calculate new pitch and yaw
            //clamp pitching to avoid flipping upside down
            float pitch = rotationVector.x + smoothPitchChange *
                 Time.fixedDeltaTime * pitchSpeed;
```

```
        if (pitch > 180f) pitch -= 360f;
        pitch = Mathf.Clamp(pitch, MinPitchAngle,
            MaxPitchAngle);

        //yaw
        float yaw = rotationVector.y + smoothYawChange *
            Time.fixedDeltaTime * yawSpeed;

        //apply the new rotation
        transform.rotation = Quaternion.Euler(pitch, yaw, 0f
            );
    }
    }
```

3. MoveToSafeRandomPosition():

Listing 13: MoveToSafeRandomPosition

```
private void MoveToSafeRandomPosition(bool inFrontOfFlower)
    {
        bool safePositionFound = false;
        int attemptsRemainng = 100; //prevent infinite loop
        Vector3 potentialPosition = Vector3.zero;
        Quaternion potentialRotation = new Quaternion();

        //loop until safe position found or we run oput of
            attempts
        while (!safePositionFound && attemptsRemainng > 0)
        {
            attemptsRemainng--;
            if (inFrontOfFlower)
            {
                //pick a random flower
                Flower randomFlower = flowerArea.Flowers[
                    UnityEngine.Random.Range(0, flowerArea.
                    Flowers.Count)];

                //position 10-20 cm i from=nt of it.
                float distanceFromFlower = UnityEngine.
                    Random.Range(.1f, .2f);
                potentialPosition = randomFlower.transform.
                    position + randomFlower.flowerUpVector *
                     distanceFromFlower;

                //point beak at flower; bird's head is
                    center of transform
                Vector3 toFlower = randomFlower.
                    flowerCentrePosition - potentialPosition
                    ;
                potentialRotation = Quaternion.LookRotation(
                    toFlower, Vector3.up);
            }
```

```
            else
            {
                //pick random height from the ground
                float height = UnityEngine.Random.Range(1.2f
                    , 2.5f);

                //pick a random radius from center of the
                    area.
                float radius = UnityEngine.Random.Range(2f,
                    7f);

                //pick a random direction roatating from y
                    axiz
                Quaternion direction = Quaternion.Euler(0f,
                    UnityEngine.Random.Range(-180f, 180f), 0
                    f);

                //combine height, rotation and radius to
                    pick a potential position
                potentialPosition = flowerArea.transform.
                    position + Vector3.up * height +
                    direction * Vector3.forward * radius;

                //choole random starting pitch and yaw
                float pitch = UnityEngine.Random.Range(-60f,
                     60f);
                float yaw = UnityEngine.Random.Range(-180f,
                    180f);
                potentialRotation = Quaternion.Euler(pitch,
                    yaw, 0f);
            }

            //check to see if agent will collide with
                anything
            Collider[] colliders = Physics.OverlapSphere(
                potentialPosition, 0.05f);

            //safe position found if no colliders overlap
            safePositionFound = colliders.Length == 0;
        }
        Debug.Assert(safePositionFound, "A safe position was
             not found to spawn");

        //set the position and rotation
        transform.position = potentialPosition;
        transform.rotation = potentialRotation;
    }
    }
```

4. TriggerEnterOrStay() method:

Listing 14: TriggerEnterOrStay() method

```
private void TriggerEnterOrStay(Collider collider)
    {
        //check if agent is colliding with nectar
        if (collider.CompareTag("nectar"))
        {
            Vector3 closestPointToBeakTip = collider.
                ClosestPoint(beakTip.position);

            //check if the closest colision point is close
                to the beaktip or not
            //note: a collision with anything but the beak
                tip should not count
            if(Vector3.Distance(beakTip.position,
                closestPointToBeakTip) < BeakTipRadius)
            {
                //look up flower for this nectar collider
                Flower flower = flowerArea.
                    getFlowerFromNectar(collider);

                //attempt to take 0.1 nectar
                //this is per fixed time step. i. it happens
                     every .02 seconds or 50x per second
                float nectarReceived = flower.Feed(0.1f);

                //keep track of nectar obtained
                nectarObtained += nectarReceived;

                if (trainingMode)
                {
                    //calculate reward for getting nectar
                    float bonus = .02f * Mathf.Clamp01(
                        Vector3.Dot(transform.forward.
                        normalized, -nearestFlower.
                        flowerUpVector.normalized));
                    AddReward(0.01f + bonus);
                }

                //if flower is empty update nearest flower
                if (!flower.hasNectar)
                {
                    UpdateNearestFlower();
                }
            }
        }
    }
```

A.3.2 *CollectObservations() methods for each hummingbird*

1. ***HymmingbirdAgentSymbolic.cs and HummingbirdAgentHybrid1.cs -Passive and Hybrid1 perception system*** Since the symbolic agent and the hybrid 1 agent share the same set of collected observa-

tions, the ***CollectObservations()*** method is common for both of them:

Listing 15: Symbolic and hybrid 1 agents collected observations

```
private void TriggerEnterOrStay(Collider collider)
    {
        //check if agent is colliding with nectar
        if (collider.CompareTag("nectar"))
        {
            Vector3 closestPointToBeakTip = collider.
                ClosestPoint(beakTip.position);

            //check if the closest colision point is close
                to the beaktip or not
            //note: a collision with anything but the beak
                tip should not count
            if(Vector3.Distance(beakTip.position,
                closestPointToBeakTip) < BeakTipRadius)
            {
                //look up flower for this nectar collider
                Flower flower = flowerArea.
                    getFlowerFromNectar(collider);

                //attempt to take 0.1 nectar
                //this is per fixed time step. i. it happens
                     every .02 seconds or 50x per second
                float nectarReceived = flower.Feed(0.1f);

                //keep track of nectar obtained
                nectarObtained += nectarReceived;

                if (trainingMode)
                {
                    //calculate reward for getting nectar
                    float bonus = .02f * Mathf.Clamp01(
                        Vector3.Dot(transform.forward.
                        normalized, -nearestFlower.
                        flowerUpVector.normalized));
                    AddReward(0.01f + bonus);
                }

                //if flower is empty update nearest flower
                if (!flower.hasNectar)
                {
                    UpdateNearestFlower();
                }
            }
        }
    }
```

2. ***HummingbirdAgentHybrid2.cs - Hybrid 2 perception system***

Listing 16: Hybrid 2 agents collected observations

```
public override void CollectObservations(VectorSensor sensor
    )
    {
        // if nearest flower is null before
            CollectObservations is called.
        if(nearestFlower == null)
        {
            sensor.AddObservation(new float[10]);
            return;
        }

        //observe agent's local rotation (4 observations)
        sensor.AddObservation(transform.localRotation.
            normalized);

        //observe a fot prooduct that indicated whether a
            beaktip is in front of flower (1 observation)
        // +1 means beakTip is directly in front of flower,
            -1 means it is directrly behind
        sensor.AddObservation(Vector3.Dot(toFlower.
            normalized, -nearestFlower.flowerUpVector.
            normalized));

        //observe as dot product whether a beak is pointing
            toeards flower (1 observation)
        // +1 means beak is pointing directly at flower, -1
            means directly away
        sensor.AddObservation(Vector3.Dot(beakTip.forward.
            normalized, -nearestFlower.flowerUpVector.
            normalized));

        //total 6 observations
    }
```

BIBLIOGRAPHY

D. Terzopoulos and T. F. Rabie, "Animat vision: Active vision in artificial animals," in *Proceedings of IEEE International Conference on Computer Vision*. IEEE, 1995, pp. 801–808.

T. F. Rabie and D. Terzopoulos, "Active perception in virtual humans," in *Vision Interface*, vol. 2000, 2000.

——, "Modelling active vision systems for dynamic simulated environments," in *IASTED International Symposium on Modelling and Simulation (MS'2001)*, 2001, pp. 34–41.

Z. Zhang, D. Weng, H. Jiang, Y. Liu, and Y. Wang, "Inverse augmented reality: A virtual agent's perspective," in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 2018, pp. 154–157.

J. D. N. Dionisio, W. G. B. III, and R. Gilbert, "3d virtual worlds and the metaverse: Current status and future possibilities," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, pp. 1–38, 2013.

GlobalData. (2021) Ar to see 22-fold revenue growth by 2030 while ar-based devices set to one day replace the smartphone, says globaldata. [Online]. Available: https://www.globaldata.com/ar-see-22-fold-revenue-growth-2030-ar-based-devices-set-one-day-replace-smartphone-says-globaldat

M. Billinghurst and H. Kato, "Collaborative mixed reality," in *Proceedings of the First International Symposium on Mixed Reality*, 1999, pp. 261–284.

E. Weitnauer, N. M. Thomas, F. Rabe, and S. Kopp, "Intelligent agents living in social virtual environments–bringing max into second life," in *International Workshop on Intelligent Virtual Agents*. Springer, 2008, pp. 552–553.

D. Voyager. (2021) Second life stats. [Online]. Available: https://danielvoyager.wordpress.com/sl-stats/

M. Shah, "Guest introduction: the changing shape of computer vision in the twenty-first century," 2002.

N. J. Nilsson *et al.*, "Shakey the robot," 1984.

T. Owen, "Active vision edited by andrew blake and alan yuille the mit presslondon, 1992, 368 pages incl. index (£ 44· 95)," *Robotica*, vol. 11, no. 5, pp. 487–487, 1993.

D. Marr and L. Vaina, "Representation and recognition of the movements of shapes," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 214, no. 1197, pp. 501–524, 1982.

Y. Shirai, "Recognition of polyhedrons with a range finder," *Pattern Recognition*, vol. 4, no. 3, pp. 243–250, 1972.

R. A. Jarvis, "A perspective on range finding techniques for computer vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 2, pp. 122–139, 1983.

P. Besl and R. Jain, "Computing surveys, 17 (1985), 135-145.[2] i. gargantini," *Computer Graphics Image Processing*, vol. 20, pp. 365–374, 1982.

R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.

M. E. Spetsakis and Y. Aloimonos, "Closed form solution to the structure from motion problem from line correspondences." in *AAAI*, 1987, pp. 738–743.

J. R. Quinlan, "The effect of noise on concept learning," *Machine learning: An artificial intelligence approach*, vol. 2, pp. 149–166, 1986.

T. Kuhn, "The structure of scientific revolution (: 1970)," *Kuhn2The Structure of Scientific Revolution1970*, 1962.

R. G. Burton, *Natural and Artificial Minds*, 1st ed. SUNY Press, 1993.

J. W. Davis, "The molyneux problem," *Journal of the History of Ideas*, vol. 21, no. 3, pp. 392–408, 1960. [Online]. Available: http://www.jstor.org/stable/2708144

R. Held, "The newly sighted fail to match seen with felt," 2011.

R. Kozma, "Computational aspects of cognition and consciousness in intelligent devices," *IEEE Computational Intelligence Magazine*, vol. 2, no. 3, pp. 53–64, 2007.

D. T. Tamer F. Rabie, "Modelling active vision systems for dynamic simulated environments," 2001.

OpenSimulator. (2021) Opensimulator. [Online]. Available: https://www.opensimulator.org/

Gazebo. (2021) Gazebo. [Online]. Available: https://gazebosim.org/

Vrep. (2021) Vrep. [Online]. Available: https://www.coppeliarobotics.com/

WeBots. (2021) Webots. [Online]. Available: https://cyberbotics.com/

Unity. (2021) Unity. [Online]. Available: https://unity.com

Unreal. (2021) Unreal. [Online]. Available: https://unity.com

Unity-Technologies. (2021) ml-agents. [Online]. Available: https://github.com/Unity-Technologies/ml-agents

I. Limit. (2021) Ml-agents: Hummingbirds. [Online]. Available: https://learn.unity.com/course/ml-agents-hummingbirds

Anaconda. (2021) Anaconda. [Online]. Available: https://www.anaconda.com/