Exploring Agent Based Modelling

In The New Age of Advanced AI and 3D Modelling

James Massey

May 2020

Abstract

This project attempts to explore agent-based model simulations and build upon best practise within the area, primarily focusing on the advantages that machine-learning based techniques can provide when utilized as a mechanism for control of agents within such systems.

Project Dissertation submitted to Swansea University
in Partial Fulfillment for the Degree of Master of Science

Department of Computer Science
University of Wales Swansea

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

May 1, 2020

Signed: James Massey

Statement 1

This dissertation is being submitted in partial fulfillment of the requirements for the degree of a MSci in Computer Science.

May 1, 2020

Signed James Massey

Statement 2

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

May 1, 2020

Signed: James Massey

Statement 3

I hereby give consent for my dissertation to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

May 1, 2020

Signed: James Massey

Contents

List of Figures

1 Introduction

It is generally accepted within the scientific community that life on Earth began around 4 billion years ago. In this time life has developed into a large spectrum of different forms, from us human-beings to micro-organisms. Over time each has developed its own impressive catalogue of behaviours and interactions. Small scale behaviours and interactions between independent actors within a system can often give rise to complex higher-level collective behaviours. Studying the emergence of these high-level complex behaviours provides insight into how and why they occur, and possibly lend hints as to possible unknowns within systems. However, such collective behaviours can be the product of thousands of different variables and can be extremely difficult for scientists to simulate and research.

By utilizing agent-based modelling (ABM), it is possible to model systems as a collective of their individual parts. These parts can be individual actors themselves or represent the functionalities of larger groups, such as organizations. These independent actors within such a system are dubbed agents. These modelling concepts can then be applied within the construction of ABM simulations, in an attempt to simulate how agents may interact with each other within a closed and independent system. There are many research papers that discuss how we can begin to think about what attributes might shape an agent an individual. In models concerned with studying social science, this could include an individual's abilities of perception or learning for example. Agents actions within their environment are often guided by a simple set of logic rules, while the aim is to recreate complex behaviours and phenomena. This information can be useful for us to both understand how phenomena can occur and to help us to make predictions on how they might appear in other environments. These ABM simulations are applied over a wide array of problem areas in order to explore how the behaviour of a system can be generated.

> "Generally, there is still a serious lack of understanding regarding the connections between the structure, dynamics, and function of complex networks such as technosocio-economic-environmental systems. Therefore, emergence often has an element of surprise. So far, for example, we do not understand emotions and consciousness, and we cannot even calculate the "fitness" of a behavioural strategy in the computer. The most exciting open puzzles in science concern such emergent phenomena. It would be interesting to study, whether social and economic phenomena such as trust, solidarity, and economic value can be understood as emergent phenomena as well. Agent-based simulations appear to be a promising approach to make scientific progress in these areas."[1]

The first agent-based model can be traced back to 1971, with the more widespread adoption of the modelling technique coming later in the 1990s. However being a relatively new technique coupled with the landscape of ever-advancing technology, there is much research to be done in how ABM simulations are best implemented, applied and how their results are represented in this modern era.

1.1 Motivation

Motivation for this project largely comes from two works of research which shall be heavily referenced throughout this project, Growing Artificial Societies by Joshua M Epstein and Robert Axtell and also Emergent Tool Use From Multi-Agent Autocurricula by OpenAi. These two works of research combine a number of areas that I as a researcher hold strong interests in. One such area is machine learning, which will feature heavily within this project.

2 Related Work

’Growing Artificial Societies’[2] is a work of research focused on the development of a societal agent-based model. The book discusses how we can represent an artificial environment, and how agents under the control of simple logical rules can give rise to interesting and complex behaviours, finally giving rise to an artificial society of sorts. The model that they build is aptly named the ‘Sugarscape’, after the resource which agents endeavour for. The Sugarscape is a 2D array where each index denotes a position within the environment. This array can then be represented by a 2D grid to visually represent the agent’s habitat. This allows the researchers to mimic a top-down view of their environment.

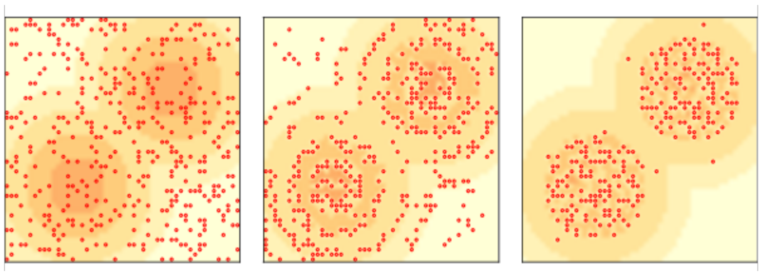


Figure 1: The Sugarscape

Figure 1 gives us a look at the Sugarscape. The red dots within the image represent agents, while positions with a yellow tint indicate that this is a position which sugar regrows at. The intensity of the yellow conveys the level of food regrowth at that position. Sugar is a valuable resource for agents living on the Sugarscape, and many of the behaviours displayed within the work of research revolve around agents affection for it. The figure above shows how such a model progresses over time, as we can see agents congregate towards areas with higher levels of sugar re-growth over time.

The behaviours which agents exhibit are driven by a set of simple logical rules, which dictate each action an agent takes. At an early stage agents actions are limited to just movement, with their movement being dictated by some logic that tells agents to move if certain conditions are met, the researchers dub these ’Movement Rules’. For example, the first set of movement rules which the researchers develop simply instructs agents to identify the grid position with the most sugar, and then relocate to this position. Agents are only able to access array positions within the same column or row that they are in at any given time. This means that agents are limited in that they can only move in horizontal and vertical channels, and also observe sugar in these channels. Actions taken by the agents are also heavily reliant on their environment, which has its own set of rules that it is governed is by. For example, sugar regrowth across the landscape is dictated by a set of rules the authors entitle 'Regrowth Rules' which change throughout the project. The combination of these two evolving rule sets then leads to the emergence of a range of varying behaviours.

Over the course of the book, the authors detail how the complexity of the Sugarscape is developed as new functionality is implemented, such as systems that attempt to model ideas of wealth inheritance and culture. The complexity of the model increases with each added functionality to the agents and their environment, and also in turn, the complexity of the emergent behaviours. However, the logical rules that govern each individual added functionality are kept simple, building a solid bottom-up view of their world. The authors do a very good job at documenting the implications each ruleset/behaviour has on the Sugarscape over time, lending us many ideas as to how we might study some virtual environment and present findings on emergent behaviours. However, the behaviours of agents on the Sugarscape can be considered to be 'hard-coded'. Agents are defined to take actions if certain conditions are met. The strategy for them to achieve their goals are already in place within the movement rules. As such, agents are not left to devise strategies of their own, and the behaviour of each individual agent can be heavily reliant on its the starting parameters within the Sugarscape. Variables such as an agent's starting position, individual values etc. can pigeon-hole these agents into loops due to a lack of adaptability. This is a very interesting read with heavy relation to the problem area that we shall be looking at. Overall the book provides insight on how we can construct an environment that leads to the emergence of complex behaviours amongst agents and also highlights that constructing such an effective environment is largely reliant on striking a balance between complexity and simplicity. In its later stages, the Sugarscape is made up of both a large number of agents and a variety of interacting units.

'Emergent Tool Use From Multi-Agent Autocurricula'[3] is a paper written by researchers at OpenAI, the focus of which is again on the development of a virtual environment, within which agents are tasked with playing a team-based game of hide-and-seek. This model is built with the Unity Game Engine, and as such, takes full advantage of the 3D visualisation and physics capabilities that this provides. An environment is set up within which there are both obstacles for agents to traverse around and tools for them to strategize with as they play. Obstacles are walls that restrict agents line of sight, with tools being large move-able objects that include both a ramp and a cube. Agents within this model have a limited number of actions; They can move, grab objects and also lock them in place. Objects that are locked in place can then only be unlocked by an agent on the same team. This allows for some interesting possibilities, as agents are able to use tools to block off areas of the environment, or instead traverse it. Agents decisions within this work of research are driven by techniques from the area of machine learning. In particular, they utilize reinforcement learning and self-play. Over multiple iterations, agents are left to develop different approaches to the game. A number of hypotheses can be constructed on what behaviours are expected to emerge. For example, it can be expected that the hiding team will figure out that it is possible to move boxes to block off doorways. Due to the utilization of self-play, we can then also hypothesize that in response we would expect the attacking team to adapt to using the ramps to manoeuvre over obstacles. This constant cycle of attack and counter-attack between teams of agents provides a constant source of learning for them. Agents are designed to have no information on the wider environment, with the only observations agents make coming from their own interactions with the environment alongside a limited field of perception, giving them a form of sight. They are then left to discover a strategy solely through their interactions with the environment whilst being constantly pitted against a worthy competitor. Over a number of iteration, agents displayed a variety of expected behaviours, showing credibility for the wider adoption of machine learning within the field. Interestingly, agents also developed a number of unexpected behaviours, the most notable of which came about due to agents exploitation of a bug within the physics of their environment itself. This gives us an idea for the use of machine learning for environment testing. The paper also makes good use of real-time 3D visualisation. We are clearly able to see how agents within the model develop their strategies as their perception field is visible. I find the contrasting approaches between the model presented here and that of the Sugarscape highly interesting, with these contrasts forming the basis of this research.

'Agent-Based Modeling: A New Approach for Theory Building in Social Psychology'[4] serves as a review of Agent-Based modelling and its use within the realm of social psychology. The paper introduces ABM and discusses its history, moving on the use of ABM against a popular theory-building approach within social psychology. The paper argues that ABM is a solid approach to theory building with respect to the theoretical concerns of social psychology. This is as within an ABM an agent is often represented as an individual. In ABM systems we can incorporate a diverse set of different system that encompasses some of the processes that social psychologists study, from cultural exchanges to economic systems. The paper provides more overarching statements on the topic area than the works we have previously discussed, in one case stating:

> 'ABM is particularly suited for those who seek to explain how a system's behaviour is generated by underlying processes or mechanisms.'

It can be seen how this makes ABM simulations very useful tools for social psychologists as it allows researchers access to more than a single layer of analysis. It is possible to study agents both as individuals and as collectives, and also discover what might link different theoretical models, as the paper poses the example - 'A study of interpersonal attraction might discover what factors make one individual prefer one potential mate as opposed to another.' This provokes me to think about the scope of ABM simulations and why they are applied. With tradition models focusing on the generation of overarching behaviours from underlying systems, with the work of Emergent Tool Use From Multi-Agent Autocurricula in mind, I notice that the introduction of machine learning may invoke the inverse of this; the generation of underlying processes

'How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design'[5] provides some guidelines that I can use as I start to think about attempting to build an ABM. The paper discusses how it is possible to model an agent as an individual, providing a list of properties that might be used to define an agent, such an individuals need for resources or their ability in competition. The paper also gives an overview of a broad set of problem areas that ABM can be applied to, stating -

> '..they can shed new light on social and economic systems from an "ecological" perspective. In fact, evolutionary ecological models can also reflect the feature of steady innovation, which is typical for socio-economic systems. Moreover, such models appear particularly suited to study the sustainability and resilience of systems. Finally, they can be well combined with other simulation methods used in the natural and engineering sciences, including statistical physics, biology, and cybernetics.'

I will later reflect back to some of these problem areas discussed in light of this research, hoping to expand this discussion with any new insights I have gained, and how these insights might apply across these various problem areas.

'Simulation Tools for Social Scientists: Building Agent-Based Models with SWARM'[6] discusses how an artificial environment is constructed where agents form a simple economy. Their model is developed using the Swarm toolkit developed at the Santa Fe Institute. Agents in this model are organised by architecture provided by Swarm mechanics. This from The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations:

> 'A swarm is a collection of agents with a schedule of events over those agents. For example a swarm could be a collection of 15 coyotes 50 rabbits a garden with carrots, and a simple schedule: the rabbits eating the carrots and hiding and the coyotes eating the rabbits'

Utilizing this toolkit, they build an environment of 20 agents split evenly into two types: vendor and consumer. At each timestep vendors and consumers are randomly matched and enact an exchange if a buying condition is met, if this condition is not met within the time-step then the condition is updated. They keep track of the mean price behaviour and notice a cyclical pattern of periods of chaotic behaviour. This paper refers to the results achieved with the use of a small number of agents with respect to Growing Artificial Societies, providing a quote from the authors that serves to remind us of the importance of complexity to some extent. The utilization of Swarm and a schedule of actions concretely defines a system of actions that will always take place, as such the underlying step-to-step behaviour that emerges under such concrete rule sets can be expected. The researchers also pose a problem that will become relevant to us:

> 'Are our simulation tools capable of dealing with a large number of agents in a reasonable computational time? Explorations in this direction are lacking at present, but one suspects that the principal constraint involves hardware characteristics more than software specifications.'

'From Artificial Societies To New Social Science Theory'[7] provides a literature review within the area of social science modelling. The paper poses some valid criticisms on the use of ABM for social science modelling, posing the question of whether it is possible to construct 'sensible abstractions for social and political structures within such a model.' As such this provides motivation to deviate the focus of the project away from passing insights within some problem area using a model we build, in favour of exploring the approaches researches can take to building an ABM, and the positives and negatives that they bring. For example, the focus shall be on evaluating the tools which can be used to build a model and their potential application across different problem areas rather than the emergent behaviours exhibited by agents within our model.

'Agent-Based Modelling of Forces in Crowds'[8] attempts to build an ABM that simulates the interactions of kinetic forces exerted by individuals in crowds. Their work extends upon the existing Kirchner model. The paper displays how we can construct the underlying systems needed for such phenomena to come to life, for example; systems required to model the application of force such as speed, and also rational resultant consequences such as injury. The resulting model of their work is, in the author's words 'capable of answering quantitative and qualitative predictions related to crowd injuries and egress dynamics.'

In 'Agent-based modelling of pedestrian movements: the questions that need to be asked and answered'[9] the authors develop a multi-agent model concerned with pedestrian movement flow in restricted spaces such as sidewalks. This paper goes into more in-depth detail on how their underlying model is constructed, how they came about formulating their rule sets and finally how we can relate the information taken back to the real world. The paper utilizes conventional control techniques for their agents, much like within Growing Artificial Societies a simple logical rule mindset is maintained here. Here ABM is applied to the problem area of pedestrian movement, showcasing the wide range of potential application areas for ABM simulations. However, what is interesting to notice here is the lack of visualisation or any argument on how they exactly developed any informative results from their model. While ABM can be applied across many problem areas, I will explore how new information and understanding be extracted from these models.

'A Survey of Agent Based Modelling and Simulation Tools'[10] provides a detailed survey of published tools used for building ABM simulations. This is helpful to us as it allows us to discover simulation tools built by other researchers to look at within our project for inspiration. From this paper, I have highlighted an ABM Developed suite called NetLogo, which we will discuss later within our project. This paper also contains details of possible application areas for ABM simulations which is important to keep in mind as we pose developments. Finally, the paper concludes by posing some future research challenges for the area of ABM simulations, stating that there is a "need for a collaborative computational project in ABMs where the community can share best practice and software." This follows closely to what I shall be attempting to achieve within our project. Whilst the project is not a collaborative effort, efforts shall be focused on the development of best practice within this area.

'Machine Learning Meets Agent-Based Modelling: When Not To Go to A Bar'[11] is a paper that discusses the integration of machine learning into agent-based modelling. The paper details two distinct cycles for each methodology and proposes an integrated cycle which allows for more adaptive agent-based simulations through the marriage of the methodologies. The paper allows us to think about how we can develop a system that incorporates both systems ourselves, detailing how such systems may interface with each other. In this paper, a small case study is carried out into the integration of some machine learning technique, namely genetic algorithms, into a model that attempts to tackle the El Farol Bar Problem. What is relevant here is the papers proposed integrated cycle. This cycle helps us to plan and understand our model. A figure of the development framework can be seen below.

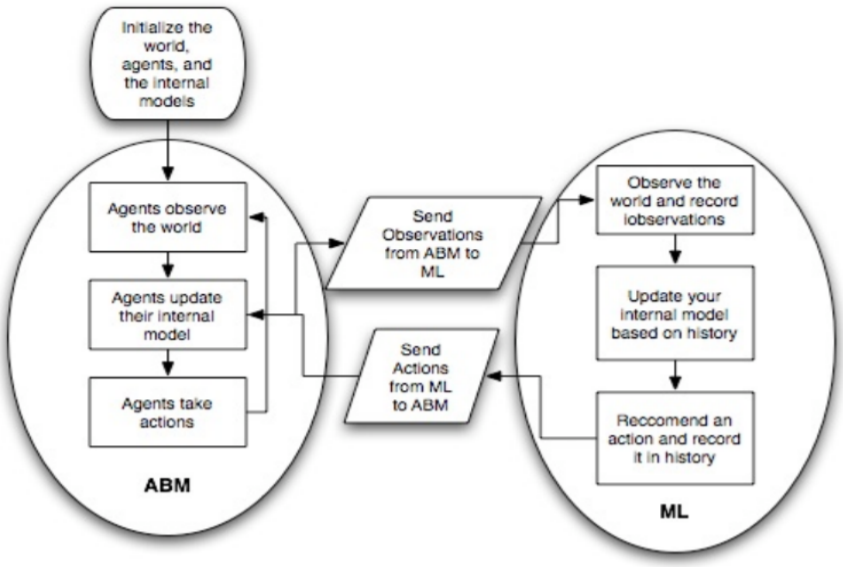

Initialize the world, agents, and the internal models
Agents observe the world
Agents update their internal model
Agents take actions
ABM
Send Observations from ABM to ML
Send Actions from ML to ABM
Observe the world and record lobservations
Update your internal model based on history
Reccomend an action and record it in history
ML

Figure 2: ABM and ML Integrated Cycle

I will be using this framework to aid in planning the development of my own model, and will later discuss how the Unity Machine Learning SDK compares to this framework.

'Agent-based models and individualism: is the world agent-based?'[12] Is a paper by David O'Sullivan and Mordechai Haklay at UCL that presents an overview of examples of the application of ABM to a range of scientific field, from social sciences to archaeology. The paper attempts to build a discussion whilst pointing out "important interpretative philosophical issues concerning social theories underlying these models." This discussion comes off the back of the researchers having completed their own model that is concerned with pedestrian behaviour in complex spatial environments. As the field progresses, however, models are becoming more and more complex, and the criticisms of this paper become all the more relevant. This paper provides a very good explanation of the role of the modeller -

> 'Are our simulation tools capable of dealing with large number of agents in a reasonable computational time? Explorations in this direction are lacking

at present, but one suspects that the principal constraint involves hardware characteristics more than software specifications.'

A model is entirely dependant on the assumptions made by the modeller, and as such, as mentioned by the paper "such models must be subject to critical examination of their assumptions, and that model builders should engage with social theory if the approach is to realise its full potential." Works such as this help guide my research by raising concerns about the validity of models if I were to focus my research on the model itself.

3 This Project

Since the project proposal, the project has evolved to more closely study agent-based modelling itself as a computational modelling technique and the tools that can be used within this area. In the project proposal, I listed a number of aims for the project:

- Develop a Visual Simulation of an Agent-Based Model and document the emergent behaviours that can be observed.
- Detail how this model is constructed, and how agents individual traits and possible actions within their environment are defined. I will also talk about our environment and how this evolves over the lifetime of this project.
- Discuss how we can apply our model and ABM in general across various scientific problem areas.

The choice I have made to alter the goals of the project comes in light of various concerns raised within the literature about the validity of insights that can be taken from an agent-based model, and how this is dependant on the underlying assumptions that the model is built upon. Within the exploration of the background literature, I have also highlighted a number of gaps in research that are highly interesting and could benefit from further research. As such, a revised list of the project aims goes as follows:

- Compare and contrast how ABM simulations are constructed within established literature.
- Develop a 3D visual simulation of an agent-based model, inspired by Growing Artificial Societies and document the emergent behaviours that can be observed.
- With the development of this model, I will define various target behaviours and try to achieve these behaviours within the model utilizing different methods recognized within the literature.
- Detail how this model is constructed, from the construction of an environment to how agents are represented. Highlighting how agents behavioural decisions within their environment are conveyed through real-time visualisation.
- Through the use of our model we will be evaluating and comparing two different mechanisms of control for our agents; traditional rule-based and machine learning-based methods,

- Discuss where these techniques that be utilized within ABM simulations, problem areas they may be applied to and the advantages that they bring.

In the proposal document, I also stated that the model will be developed in 2D. However, I have made a decision to switch to the development of a 3D model. I make this decision as the Unity Machine Learning Agents SDK is currently still in beta, and is not constructed with Unity's 2D packages in mind. the switch to development of a 3D model offers a larger array of packages that can utilized, which will greatly reduce the complications I may have during development. As I began constructing the model in 2D, when I have later finished construction of the 3D model, I will be able to compare the advantages that either can provide.

3.1 Tools

The model will be built within the Unity game engine. Whilst initially developed to serve the gaming industry, use of this platform has since been adopted in many other problem areas, such as its use by artificial intelligence research company DeepMind, who utilize the platform to train AI. Unity provides multiple tools that aid researchers in crafting visual simulations and advanced models. Using Unity allows me to focus on constructing the model and how the interactions that occur within it might be visualised without having the need to program logic for rendering graphics. The platform also helps to build up our model in a modular fashion, allowing us to swap components in and out for rapid prototyping.

Alongside this, I will also be using the Unity Machine Learning Agents Software Development Kit in order to control machine learning-based agents. This SDK provides functionality for reinforcement learning alongside other machine learning tasks within Unity both internally and through external communication to packages such as TensorFlow. TensorBoard functionality is built into the SDK, therefore I will be able to provide training graphs which will detail the training progress of agents.

4 Exploring Agent Based Modelling

4.1 Models Within The Literature

As highlighted by the background literature, there are many ways that researchers can attempt to approach both implementing an agent-based simulation and how the interactions within such a system can be presented. For example, let me compare the works of Growing Artificial Societies and Emergent Tool Use From Multi-Agent Autocurricula. These two papers have inherently different aims with their models. The first is a large scale model where agents represent individuals, with the aim to conduct a study on interactions between individuals within the area of social science. The study here is focused on explaining how the systems overarching behaviour between agents is generated from its smaller known parts. The second model is a machine learning study which uses a small scale model where agents are divided into pairs and engage in a game of hide-and-seek over multiple iterations. Whilst the focus of this model is also on studying the behaviour exhibited by agents, here a directive and a set of observations is supplied to agents, which are left to adapt their moment to moment actions.

While having different aims with their studies, both papers make use of ABM in order to conduct their research. Whilst this is indicative of the scope of application of ABM, it is imperative to highlight the different ways each paper attempts to tackle various areas of this modelling technique.

Firstly there is how agents within each model are controlled in order to produce the various intended behaviours. Growing Artificial Societies utilises hand-written scripts in a rather lower-level behaviour-centric approach to the control of their agents, with agents being told explicitly how to act under given circumstances. Emergent Tool Use From Multi-Agent Autocurricula, on the other hand, utilizes machine learning to control agents, forming a more goal-oriented approach. This highlights the different approaches it is possible to take when developing an agent-based model. Whilst the team at OpenAI were primarily utilising agent-based modelling to conduct research within the realm of machine learning, the underlying model is still concerned with getting agents to exhibit some wished behaviour, which their model is extremely successful with. As such, with their research they have inadvertently posed the question; should this approach for the control of agents be best practise for other applications?

It can also be noted that the models here are represented in different fashions. Axtel and Epstein's model is a 2D mathematical representation that can be visualised as a grid at each step, whilst OpenAIs model is a fully visualised 3D model that also makes use of the underlying physics engine. With this, comes different approaches to visualisation. Axtel and Epstein's 2D model often uses graphs and other forms of statistical analysis in order to visualise and evaluate the actions carried out by agents within their model over some given time period, whilst the other model largely focuses on visualising agent interactions in the moment. This key difference between the visualisation aims of both models is shown by OpenAIs choice to provide a mechanism that allows us to see what an agent is considering. Within Axtel and Epstein's model, at each stage of development, agents are told to actively seek out some other objects within the environment in order to carry out some form of interaction. Their real-time visualisation can then depict the sequence of actions that were carried out. For example, we are able to observe an agent travel across the Sugarscape to collect a piece of sugar. However, we are not aware of the piles of sugar that may have been considered by the agent during this journey that may have been collected by other agents. As such, it is difficult to build a clear picture of the finer behaviours exhibited by agents in such a scenario. On the other hand, OpenAIs model offers a full 3D scene, within which agents must physically interact with their environment to complete tasks which can be easily seen. Beyond this, a key inclusion is their choice to include the agent's perception field within the visualisation itself. Often, in machine learning tasks that make use of agents, agents require some sort of mechanism in order to make observations on their surroundings. This is often hidden from the researcher, however, in OpenAIs model they effectively incorporate this feature into their visualisation, allowing researchers to see exactly what each agent is considering when performing its actions. This is a wise choice here, when we consider that there may be some unexpected emergent phenomenon that may come about within the model, it would be insightful to be able to understand the observations an agent is considering when making such decisions.

A few of the papers that I have discussed have raised some concerns within the area. Multiple papers have mentioned the limitations that computational power has held over the field since its conception, particularly when attempting to visualise the model in real-time. I will be looking at how modern tools can be used to tackle this. Some papers as mentioned have also raised concerns about the quality of the information that can be taken from agent-based models, as they rely on the assumptions made by the modeller. If the underlying assumptions made by the modeller are founded on theories that are not necessarily true in the real world, then the information taken from the model is difficult to relate to real-world use. As such, with this in mind, alongside myself as a researcher having a history within computer science, I will remain apprehensive to make claims relative to the social sciences and other fields with the development of our project. Instead, I will be focusing on the tools that can be used to help develop agent-based models and to discuss best practice within the area.

With this review of the background literature surrounding our problem area, I have discussed some of the methodologies that have been combined with agent-based modelling for research. With this, I have highlighted that there is currently little work on best practice when it comes to developing these simulations. As such, from the aforementioned works I have recognised two key areas for further research; machine learning and visualisation. Both of which will hold a heavy presence within this project.

4.2 Machine Learning and Agent-Based Modelling

Within ABM simulations there must exist some artificial intelligence mechanism that drives each individual agents behaviour. Within Growing Artificial Societies this is done by explicitly programming each individuals behaviour into what they describe as behavioural rules. While this provides them with the benefit of an immediate understanding of the small scale behaviour of agents within a system, there may be a few drawbacks here. Emergent Tool Use From Multi-Agent Autocurricula, on the other hand, utilizes machine learning in order to control their agents. Machine learning algorithms allow for agents to learn from experience without their individual behaviours being explicitly programmed. Instead, researchers define an end-goal and limiting conditions for agents and allow agents to adapt and optimize their approach to this task.

The various options that are available for the control of agents begs the question; which is more suitable for a given model? It is important to note that ABM has multiple areas for application, and thus the choice of an optimal mechanism for control shall be highly application dependant. However, drawing upon the background literature it is possible to highlight some advantages/disadvantages that these mechanisms might have in comparison to one another.

In this project, I will be comparing the model I will build under the use of these different control mechanisms in order to evaluate their use within ABM simulations. It is important to note that the model that I will be developing is inspired by that of Growing Artificial Societies, and as such, this evaluation will mainly have heavy relevance to societal models. However, I will also discuss the possible application of these control mechanisms within other problem areas through a combination of evaluation of literature and my own summations developed throughout the project.

More specifically I will be utilising the Unity ML-Agents SDK to generate a neural network that employs a reinforcement learning algorithm, taking inspiration from Emergent Tool Use From Multi-Agent Autocurricula. Reinforcement learning allows for a mechanism of control for agents without explicitly defining their actions, instead we employ a reward function. Reward functions provide 'incentive' for agents to engage in various different actions, if an agent engages in a behaviour that is deemed positive, they are attributed with some amount of positive reward. In contrast, negative reward can also be attributed to an agent if they engage in behaviour that results in an outcome that can be seen as detrimental to the target behaviour of the model. The agent will then adapt their behaviour in order to maximise their reward. As such, it is important to generalise away from attributing any sort of reward for small specific behaviours as this would be limiting the choices of the agent. I will then compare and evaluate the use of this neural network to a more traditional explicit artificial intelligence approach such as within Growing Artificial Societies, where agents actions are determined by simple logical rules. I will be looking at:

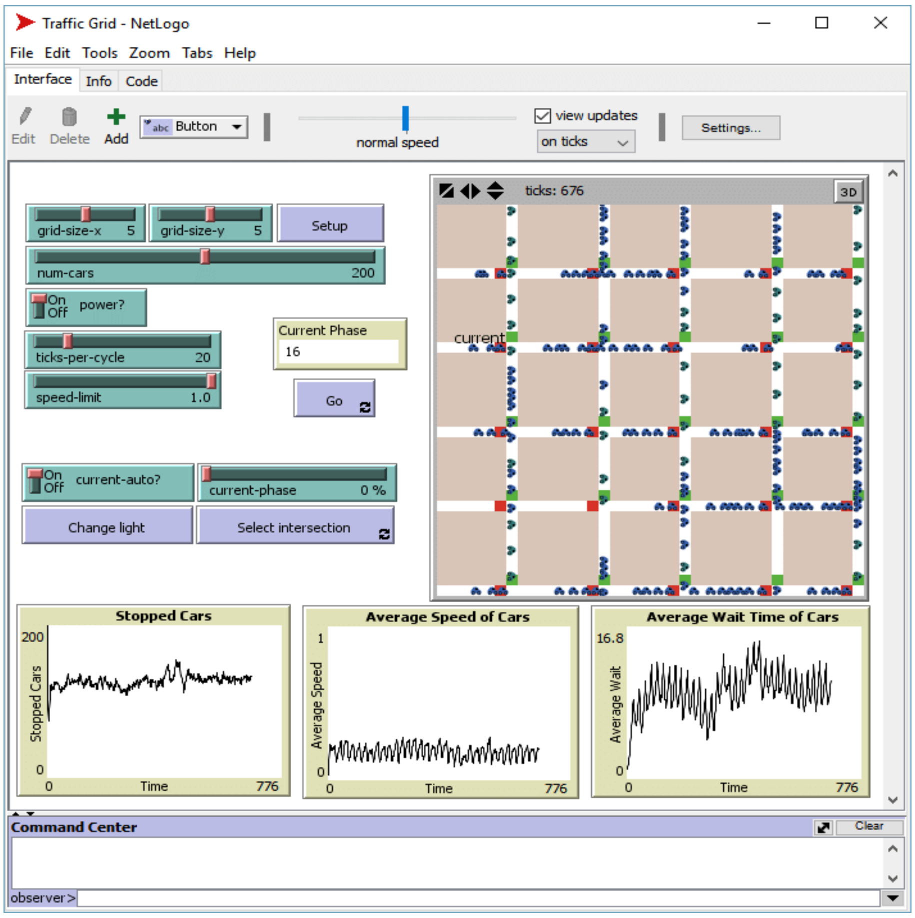
- Ease of implementation to achieve a behaviour resembling a behaviour.
- The level and nature of unexpected emergent phenomena.
- How closely an achieved behaviour resembles a desired behaviour.
- How computationally taxing utilizing machine learning for the control of agents compared to traditional rule-based methods is.

With this research, I hope to pose both some advantages/ disadvantages for the use of machine learning within ABM simulation across various application areas, proving that machine learning does indeed have a place within the field whilst developing some basic guidelines for the adoption of machine learning within societal ABM simulations.

4.3 Visualisation

Agent-based simulations can be comprised of any number of agents with varying individual attributes, these agents will also have a number of ways that they can interact with other agents and possibly other objects within their environment. The question that is present here, is how can we effectively visualise both differences between agents and interactions within these simulations?

Currently, there exists a number of pieces of software that attempt to make some headway within this area of agent-based modelling. NetLogo is such a piece of software used by researchers within this field, and provides a platform for which users can use to code agents and their possible interactions, visually displaying how the simulation and subsequent system behaviour evolves over time.



grid-size-x 5
grid-size-y 5
num-cars 200
power? On
ticks-per-cycle 20
speed-limit 1.0
Current Phase 16
current-auto? On
current-phase 0 %
view updates: on ticks
ticks: 676

Figure 3: Visualisation in NetLogo

The figure above shows a model of a traffic grid being simulated in NetLogo. It can be seen that the software provides graphs to help visualise how many cars are stopped at any given time, the average speed for cars and the average wait time that they must endure. The software also provides a representation of the model in real-time, displaying the agents moving around their environment. However, it is important to note here what information can be taken from this real-time visual representation. There is a lot of information which this visualisation struggles to convey, for example;

- What is the current speed of individual cars?
- Which cars are currently stopped?
- If a car is stopped, how long has it been waiting?
- What is the average wait time for cars along a specific route?

Focusing on how this information might be visualised allows researchers delve into the specifics of a model. For example, should cars be experiencing high average wait times, the visualisation of this information in real-time would allow us to identify specific problem areas on the grid that may need attention in order for traffic to flow more smoothly.

The visualisation of ABM simulations can be a challenge, however, as a system that is comprised of a complex model with a vast number of agents, and also the means to graphically represent this model can be extremely computationally demanding. Visualisation is an area of research within ABM that is currently limited at the time of writing. There exists other publicly available software much like NetGo which provides these capabilities, however, similarly, this is often provided in a rather limited fashion. Here, I have discussed the current capabilities of such visual ABM development suites, critiquing visualisation methods used, providing motivation to hopefully ourselves develop some heuristics for the visualisation of agents and interactions with their environments within ABM simulations.

5 Approach

In this paper I aim to discuss the area of agent-based modelling, primarily focusing our discussion on the use of machine learning and visualisation within the field. I hope that this discussion will allow me to develop some heuristics for the development of agents based simulations. As previously discussed there are two areas that we will be focusing on with the development of our model:

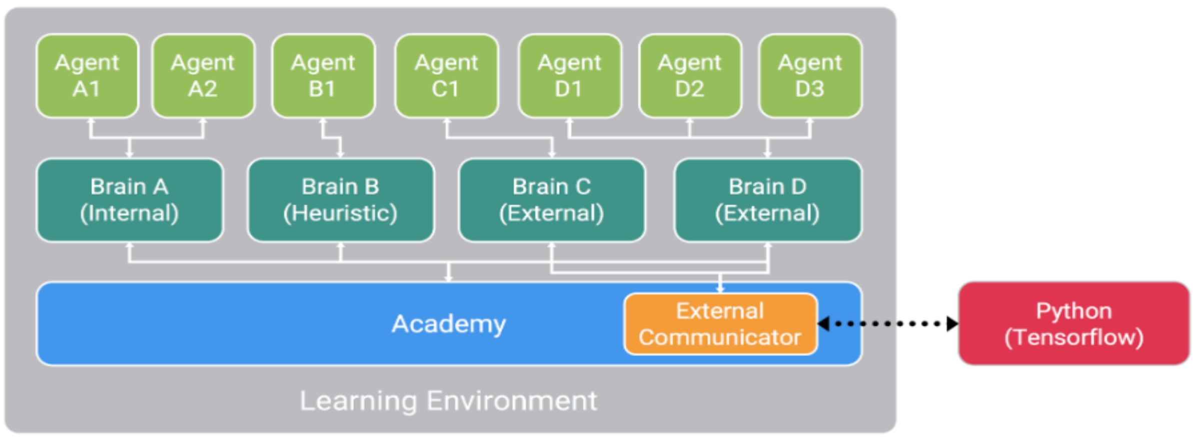
- The use of machine learning as compared to traditional rule-based within the development of ABM simulations.
- Visualisation of interactions of agents in real-time in 3D within ABM simulations.

Throughout this project I will develop an ABM simulation which will run visually in real-time, looking towards the model built within Growing Artificial Societies for inspiration on the behaviours that I will aim to display within this model. With the development of the model, I will primarily be exploring different approaches to agents can be controlled. For example, how do simulations differ when agents behaviour is controlled by machine learning over conventional scripted AI? Is the resulting behaviour closer to the intended behaviour? How fast does the simulation run under these techniques and how simple was the development process? I hope to tackle questions such as these in an attempt to build upon consensus within agent-based modelling with the varying different approaches to implementation displayed within the literature in mind. I also hope to be able to develop informative and intuitive real-time visualisations. For this, I will be taking inspiration from Emergent Tool Use From Multi-Agent Autocurricula, in developing a visualisation system focused on relaying the observations that an agent is considering.

The model I will be construction is tailored to demonstrate how the utilization of different control mechanism for agents within ABM simulations can cause models with the same behavioural target for agents to differ in both development and results. When making an alteration to the model or adding some new system agents can utilize, I will construct a hypothesis on the various behaviours that I expect may arise from such a change and compare our initial expectations against the resulting behaviours. I will pay keen attention to the emergence of unexpected behaviours, investigating how both the rules of our environment in combination with how we have implemented agent's behaviour could possibly lead to their fruition.

5.1 The Unity Machine Learning SDK

In order to implement the machine learning agents, I will be using the open-source toolkit developed by Unity. This SDK is designed to integrate into the Unity editor easing the strain on development. With this integration, developers are able to set up an environment with real-time graphics and physics, and also implement methods such as deep reinforcement learning and evolutionary strategies without having to interface with another API.



A visual depiction of how a Learning Environment might be configured within Unity ML-Agents Toolkit.

Figure 4: Unity ML SDK

The above figure displays how a learning environment may possibly be configured. Each agent has a set of actions and observations that they can make within their environment. All agents within this system at a given time will have the same set of actions and observations that they can make, so there is no need for a multi-agent system. Instead, I will be setting up two separate training environments, one for each type of brain that I will be evaluating, and also an environment where agents with both types of brain co-exist. I am doing this so that I am able to easily isolate each type of agent for evaluation and comparison, as the existence of multiple types of control mechanisms for agents within the same environment could affect each-other.

The SDK provides four different brain types; Player, Heuristic, Internal and External. We will be focusing on the Internal and Heuristic Brains within our project. A further breakdown of each of the brain types follows;

- Player - Agents are controlled by user input.
- Heuristic - These brains are traditional rule-based mechanisms, where the agent's actions are decided by hard-coded behaviour.
- Internal - With internal brains, the machine learning model is embedded into the project itself with TensorFlowSharp. Utilizing this brain means we can avoid using separate APIs and having to interface with them through Unity.
- External - These are brains that can be developed outside of Unity, using packages such as TensorFlow. External brains must make use of an external communicator in order to interface with the chosen library.

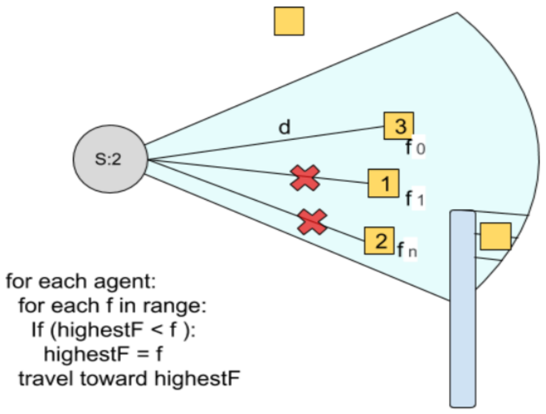
The academy object within a scene holds all brains as children, and thus allows us to configure our learning model and environment. Options that the academy allows us to configure include the number of engine steps between each agent decision and the length of each training episode etc. The SDK also offers batch computing for decisions for agents handles by the same brain, thus, the choice of two simultaneous single-agent environments alongside a simultaneous multi-agent environment will allow us to assess how efficiently the SDK handles each of these frameworks.

5.2 An Overview of The Model

As previously mentioned I will be looking towards the model used in Growing Artificial Societies for inspiration with our model. I will aim to recreate the initial behaviours from this model with some added complexities that I hope will highlight use cases for the different methodologies that we aim to study. Here, I detail the plans that I have for our model.

ABM simulations can often surpass 1000s of agents, as their aim is often to explore the behaviour exhibited by a collection of individual agents. However, due to the nature of this study, I have decided to use a reduced number of agents in order to avoid problems. ABM simulations can be extremely computationally expensive, such simulations are often not represented visually in real-time due to the combination of having to both simulate agent behaviour and also having to generate graphics at each time-step. However as my focus is on how a model is developed, over the model itself and the emergent phenomena exhibited, the number of agents that are present within the simulation is less important, thus it is appropriate to use a smaller scale model for the purpose of this research. Therefore, for the majority of the study, I will only need a single agent within the environment at any given time.

The earliest behaviour displayed by agents within Growing Artificial Societies, is each agent's desire to collect Sugar, which agents achieve by moving along a 2D axis, where their observations are also limited to their current x and y channels. My model will attempt to expand upon this by allowing agents to traverse along a continuous surface within a 3D environment, later adding complexity to this simple behaviour by altering agent's abilities of perception and adding obstacles to the environment that will inhibit agents movement and block their vision. Below shows an early plan for the perception field that agents will utilise to make observations, and also some pseudo-code for the traditional approach to resource collection that also considers other agents.



```
for each agent:
  for each f in range:
    If (highestF < f ):
      highestF = f
  travel toward highestF
```

Figure 5: Model Plan

The above figure gives a large insight into the initial basis for the model. Agents final goal will be a resource collection task, similar to that within Growing Artificial Societies, with added complexity. Agents will traverse around the environment in search of food. The complexity of the observations that our agents are capable of making within the learning environment will be increased incrementally as I develop the system, alongside the complexity of the learning environment itself. I wish for agents to be able to consider food sources of various sizes and also be able to traverse around obstacles which will inhibit their sight in search of new food sources.

The underlying model will be built with the following assumptions in mind:

- Each agent will have a perception field. This is the area within which agents can make observations. Agents cannot take into consideration any data not within this area at any given time.
- Agents have a movement speed which they cannot exceed. Agents will always aim to move at their maximum possible speed.
- The environment is a finite and bounded space.

5.2.1 Visualisation In This Project

What follows is a discussion on the methodology for the visualisations that I will implement. Within Open AIs paper on emergent tool use, the team opts to visualise what the agent is observing at any given time within the simulation itself. This is done simply by making their version of an agents perception field visible to the researcher. This is an effective technique as it allows us to see what an agent could be considering. As I am utilising machine learning within this project, it is important that we are able to identify the observations agents are making in order to understand the behaviour that is generated. I will be adopting this idea and adapting it for my purposes. At any given time I wish for the simulation to be able to be paused, and when paused I wish to be able to see the perception field of agents. There are a number of ways to handle this, however, for this, I will be making use of a component that is included within the Unity ML-Agents package entitled Ray Perception Sensor 3D which handles ray casts and observations for ML-Agents. Whilst this is the case, it also visually represents the rays that are cast, also making this component useful to traditional agents purely to visualise their perception field.

6 Hypothesis

The model is built with the intention of highlighting the difference in control mechanisms. With the development of this model, I believe the added complexity will allow agents that are controlled my machine learning to adapt to new challenges more efficiently than I am able to hard-code the heuristic brains to accomplish the same tasks. With the use of machine learning methods, there is also the possibility of experiencing unexpected emergent phenomena. This is behaviour exhibited by agents that was not initially expected, as such, I shall also discuss any interesting cases of this that we encounter.

In the model's later stages, I expect to witness the emergence of simple strategy in our machine learning agents that cannot be replicated within traditional agents, at least not with anywhere near the same level of complexity. I believe this will give rise due to the observations that agents are able to make, and also that the machine learning agents will begin to consider their perception space and be able to effectively work out how to maximise their vision. As the plan is for this perception field to also be impeded by obstacles, machine learning agents should be able to more effectively seek out new sources of food than traditional agents. However, in the model's early stages of development where target behaviours are easily hand-coded, here I expect my model will benefit more from the use of a traditional brain due to the trade-off between computational power.

7 Method and Results

What is in the following subsections chronicles the development of the model. I have already mentioned that the plan is to build upon the model with added layers of complexity. As such, I structure this explanation in stages, followed by the results garnered from both models. The overarching target behaviour of agents throughout shall largely remain the same, to become efficient at collecting resources. At each stage of development, I will be building upon complexity by altering agents and their environments. I shall discuss the development process of each stage while providing supporting thoughts for decisions made alongside a hypothesis on the differences I believe the models will show.

After completion of the planned model, I also include an extra stage based around an idea I earlier discussed, inspired by agents finding a bug within Emergent Tool Use From Multi-Agent Autocurricula, the idea for this stage came about during development when an opportunity to conduct this study presented itself. An early breakdown of the each stage goes as follows:

- Part 0 - Set Up: The setup process, this will apply to both types of agents.
- Part 1 - Simple Resource Collection: How each type of agents can be instructed to collect food in a surrounding area.
- Part 2 - Revised Vision Mesh: Adding to agents action space.
- Part 3 - Priorities: Exploring agent's behaviour when there are decisions to be made.
- Part 4 - Obstacles: Looking at how easily agents can be adapted for a new environment.
- Bonus Stage - Can We Break It: Proposing ML-Agents for testing.

Each layer of complexity that I add is designed to show some difference between the control mechanisms that are being showcased. I will also discuss the visualisations I create and the information I believe they can relay about agents decisions moment to moment as the simulation progresses.

The results for each stage will contain discussion on the behaviour exhibited by agents and how closely this matches the target behaviour. I will also discuss differences between the development process and how developers are aided from the use of each technique.

7.1 Part 0 - Set Up

The setup phase of development includes a discussion of how the initial environment and agents are constructed. The aspects of development discussed here are applied throughout the stages of development and do not see much change much throughout.

First I shall talk about the tools that I will be using and how these tools are set up within the development workflow for the purposes of this research. As mentioned, I shall be using the Unity3D Game Engine which provides many of the tools required in order to create a visual environment. Alongside this, I have also mentioned the use of the Unity Machine Learning Agents toolkit that this option makes available. To be able to work with this pipeline, it must be pulled from the SDKs GitHub at https://github.com/Unity-Technologies/ml-agents. The directory includes the ML-Agents package alongside a number of example projects. The learning environment is set up within the same directory as these example projects so that we can make use of a couple of the assets provided. Most notably there is an Area subclass that I am able to inherit from, which allows control of the training area in a code-first fashion. The example projects also contain some 3D models that I will be using to ease the strain on development, such as the meshes used to represent agent objects. In order to set up a learning environment within a new project, the ML agents package must be imported into the project.

The journey of the model's development begins with a number of requirements that will need to be in place before I can consider attempting to consider a target behaviour for agents. I must start by creating some sort of physical environment within for agents to call home. This environment shall be a flat continuous surface which is bounded by walls, on which agents will be able to move in whatever direction they please. This open plane that agents will live on is a mesh made up of a flat surface and some walls.

At this point, I currently have a 3D model which can be used to represent agents physically within the learning environment, which is included within the example projects. I will be creating two types of agent, one for each control mechanism. As such I create a script for each that hold individual variables and actions for each agent which will later be defined, I name these scripts EdenAgent and EdenTraditionalAgent after the agents, which I have taken to calling Eden's. Eden's are applied a RigidBody object and a collision detector that spans the entirety of the agents 3D model. RigidBody is a class provided with Unity that includes a number of functions that allow us to take advantage of the engine's capability for simulated physics. In this case, force is applied to the RigidBody component in order to move agents around the environment. Within each Eden's script, I provide functionality that allows movement both forwards/backwards and laterally. In order to easily differentiate between Edens, I thought it suitable to colour them. Below, the ML Eden can be seen in a rather dashing blue and orange combination on the left, whilst I opted for a striking red and white combination for our traditional rule-based Eden which can be seen on the right.

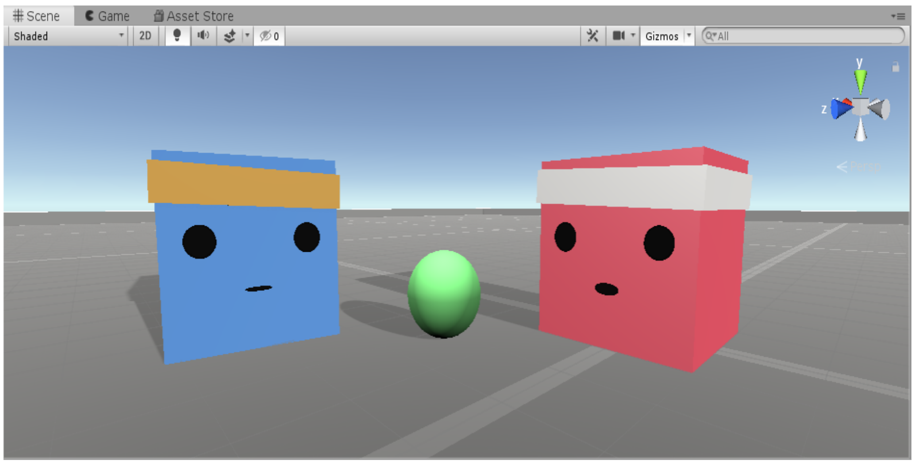


Figure 6: Agents

Finally, I want some way to test agents and environments functionality throughout development. Therefore, I create a method for Edens that allows me to take control of their actions myself. I have created two scripts for each control mechanism, as player functionality is a part of the Unity MLAgents SDK, I can implement this within the EdenAgent script and simply change the brain type to heuristic to activate this.

7.2 Part 1 - Simple Resource Collection

As previously mentioned, the first target behaviour I wish for Edens to exhibit is the ability to collect some resource around their environment, akin to that of agents collecting sugar within Growing Artificial Societies. For this to take place, I will first need to create some resource for agents to collect, and some means to distributes this across our environment.

The resource that Edens will be collecting shall simply be called food. To represent food within the environment, it has been created as a prefab object. As a prefab object, it is possible to store, configure and create new GameObjects using references to the prefab. Allowing the creation of a script that can distribute food across the environment. At this point, I want this script to simply take our food prefab object, create new GameObjects from it, and then place the objects at a random position within the area bounds. To do this, an enclosed area is created that agents will be living on, which I name EdenArea. The script that will be in charge of configuring this environment I shall give the same name, EdenArea.cs, and applied to this mesh. This script will be in charge of spawning both agents and food within our learning environment. It has functions CreateAgents(int num) and CreateFood(int num) which instantiates a number of the respective GameObjects, dictated by variable num, and distributes them with a random position within some defined range upon the EdenArea mesh. The food prefab objects are finally applied a collider so that they can be interacted with by Edens.

Next, Edens are in need of functionality that allows them to collect food, and also keep track of the food that they have collected. For this, I need to create a script which handles the logic behind food objects. I already have a Food GameObject stored as a prefab, so I now need to create a new script to be applied to this. I name this script FoodController.cs. The FoodController script holds logic that allows sources of food to 'respawn' when eaten if a Boolean is to True. If respawn is indeed set to true, my implementation simply relocates the food object in question to another spot within the defined range, else it destroys the game object. In the EdenAgent.cs script, I give Edens the ability to collect food utilizing the functionality provided by their collision detector. When an Eden is detected to be colliding with a food object, the food object in question is destroyed, and a food collected variable within the Edens class can be incremented. The OnCollisionEnter() method within agents class scans for collisions with GameObjects that hold the tag "food", upon collisions with these GameObjects, this logic is executed.

7.2.1 Traditional

Next, it is possible to begin to consider control mechanisms for Edens. In these subsections, I will be discussing the ongoing development of Edens traditional brain. At this stage, Edens needs are simple. To accomplish their task, they simply have to collect any food objects within range, and if there are not any food objects within range, to relocate by moving in some random direction.

To achieve this, a variable that dictates perception range is created within the traditional agent's script. This variable dictates the maximum area around an agent that it can be aware of food within. If there is a food object within this distance of an agent, the agent will be told to move towards the closest food source. I simulate the functionality of the agent's perception field within this classes Update() method. This method is called every scene update, allowing this logic to be run at every step. Within the Update() method I begin by creating a loop that will generate a random direction after some defined time, which can later be used to rotate agents so that they can roam for new food sources. I next use a method OverlapSphere(centre, radius), which will overlap a sphere over a target position and return a vector of colliders that was found within this sphere. For the centre, I use the agents current position. The vector of colliders that is returned will always be at least of length 2, as it will always pick up the collider of both the agents itself and the ground. Therefore the loop that will iterate through all the colliders will only need to run if the length of this list is > 2. It is also possible for agents to pick up the tag for the walls collider, as such, I then consult the collider vector for colliders that hold the tag food. I next need to provide logic for agents to decipher which is the closest out of the food colliders detected. As such, for each collider in the vector with the tag 'food', I calculate its current distance to the agent, storing the shortest distance. This shortest distance is then used as a target for agents to move towards by utilizing the LookAt() function whilst propelling our agent forward. If no colliders with the tag 'food' can be found within the vector, agents are told to relocate using the random direction that was defined earlier in combination with moving forward. This method is easily made visible by casting a wire circle around agents. Alternatively, it is possible to add a new GameObject to agents to represent their vision range.

7.2.2 ML Agents

Now, I move on to discuss how an internal brain was developed using the Unity Machine Learning Agents SDK to accomplish this same task. In these sections, however, I will be discussing the development solely of the model. Discussion of the method used for training of agents is to come later, as this is applicable to every stage. As discussed, I will be utilizing reinforcement learning in order to train agents, where a reward function is employed to incentivize agents to perform some set of actions based on a set of observations. As such, agents perception of the world is crucial to how they interact with it. In order for an agent to make rational decisions, it will require relevant data. For example, with this first goal of simple resource collection in mind, agents will need to be aware of the position of food objects relative to themselves in order to move towards them.

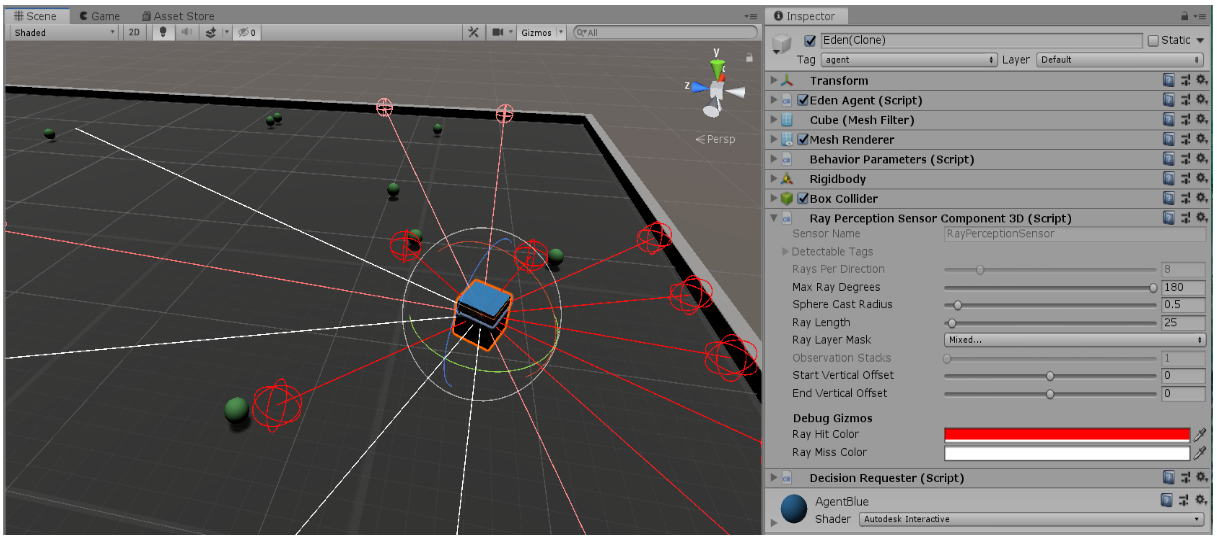
First, I add an import statement to the EdenAgent.cs class, and then changing this class to inherit from Agent rather than MonoBehaviour, providing the class with Unity ML-Agents SDK functionality. Applying this to the EdenAgent prefab automatically adds a Behaviours Parameter component to the agent which must be configured. To derive the information required for this, I must first set out our agents action space. At this point agents action space shall be discreet with just 2 branches:

- Forward Motion - Forward/No Action/Backward
- Sideways Motion - Left/No Action/Right

An agents action space is a vector of values which is decided upon by the neural network which will be generated. This vector is then passed to a function OnActionReceived(float[] vectorAction) when a new set of actions is decided upon. This method can then pass this vector to other methods that play a role in our agent's functionality. In this case, it calls MoveAgent(float[] act), which takes the values given by the neural network and converts them into forces to be used to move our agent around the environment. The next objective is to think about designing a reward function for our agents. There are a number of ways that a reward function could be designed for this task. I decided to simply attribute reward to our agents upon collection of some food.

The Unity ML-Agents package allows for the incorporation of a reward function into the logic of any script that has access to the agent. This is done with the use of the AddReward() method which will attribute agents with some amount of reward that is passed to it as a parameter. I want to attribute our agent with some reward whenever it collects a food object. As such, to the OnCollisionEnter() method that was defined within this class previously, if the colliding objects tag is equal to "food" agents are attributed some reward by calling AddReward(1f).

Finally, agents will also need a set of correlated observations to achieve their task. Initially, the behaviour I want to mimic would allow Edens to see all objects in a complete 360° arc, up to a certain distance. The observations that ML agents make within the environment itself come from a series of raycasts, created for us with MLAgents SDK functionality. To add these raycasts to agents, a Ray Perception Sensor 3D component must be added. This allows me to define a number of variables in order to construct an arc of raycasts. With the wish to cast rays in a complete circle around the Eden, the Max Ray Degrees variable is set to 180, which will distribute rays 180°on each side from an agents centre, creating a full 360°perception field.



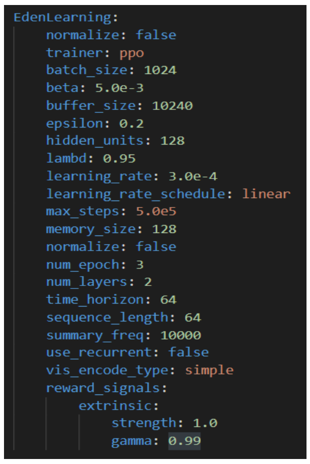
```
Eden(Clone)                                  Static
Tag  agent                  Layer  Default
Transform
Eden Agent (Script)
Cube (Mesh Filter)
Mesh Renderer
Behavior Parameters (Script)
Rigidbody
Box Collider
Ray Perception Sensor Component 3D (Script)
  Sensor Name             RayPerceptionSensor
  Detectable Tags
  Rays Per Direction      8
  Max Ray Degrees         180
  Sphere Cast Radius      0.5
  Ray Length              25
  Ray Layer Mask          Mixed...
  Observation Stacks      1
  Start Vertical Offset   0
  End Vertical Offset     0
  Debug Gizmos
  Ray Hit Color           🟥
  Ray Miss Color          ⬜
Decision Requester (Script)
AgentBlue
Shader  Autodesk Interactive
```

Figure 7: 360°View

Alongside these observations, a couple of observations are also passed to the agent within a CollectObservations(VectorSensor sensor) function which we define within EdenAgent.cs that allows the Eden to access its current velocity along the x and y-axis and also the transform.position of the Eden RigidBody component. These added data values are only passed if we set a Boolean in the agents class named vectorObs to true.

7.3 Training

To be able to train Edens, a configuration file must be first created for the training process. This configuration file is used to define parameters for the neural network. The configuration I have set out looks as such:



```
EdenLearning:
    normalize: false
    trainer: ppo
    batch_size: 1024
    beta: 5.0e-3
    buffer_size: 10240
    epsilon: 0.2
    hidden_units: 128
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    max_steps: 5.0e5
    memory_size: 128
    normalize: false
    num_epoch: 3
    num_layers: 2
    time_horizon: 64
    sequence_length: 64
    summary_freq: 10000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
```

Figure 8: Training Parameters

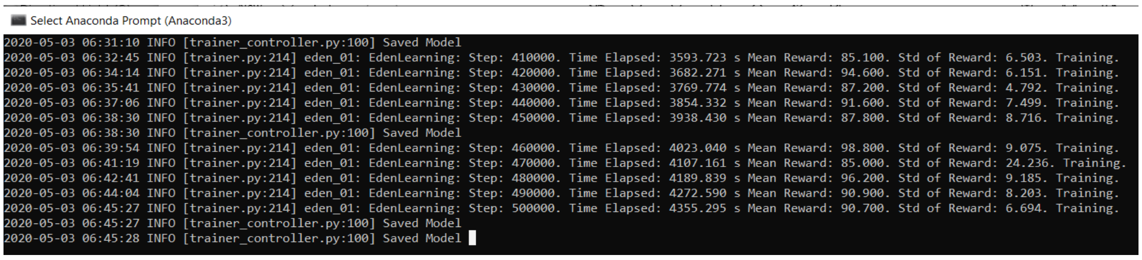
Full documentation of these parameters can be found at https://github.com/Unity-Technologies/mlagents/blob/master/docs/TrainingConfigurationFile.md. I experimented with these parameters extensively to find a result that allowed for a steady growth in mean reward, without training being too lengthy. I discovered that for my purposes, the default configuration for the trainer worked best, alongside setting normalise to false. I discovered the optimal parameters during stage 3 of development, and thus retrained the model from the first two stages with these. The increase in performance from this alteration is evident within my discussion on the results of this third stage of development. Once this configuration file is saved, Anaconda can be used in order to laugh training. I have set up a new Anaconda environment with ML-Agents functionality. Anaconda allows unity to communicate with TensorFlow in order to construct a neural network for the task we have laid out. Using the Anaconda Prompt, from within the ml-agents directory I can now use a command which then prompts us to begin training within the editor. ML agents will train until they meet one of three conditions, which I decided upon through experimentation of different training parameters in combination with the training results they produced:

- They reach the maximum number of steps that we have defined: 500,000.
- Their average increase in reward plateaus over 150,000 training steps.
- They reach an hour of training time.

7.3.1 Results

At each stage, the behaviour of each type of Eden shall be evaluated using a single agent alongside 30 sources of food that respawn at a random location within the environment upon collection. Both agent perception fields were given a radius of 25 units along the world axis and totals a 360°arc. The ML Edens use 32 rays per direction, totalling 64 rays used for observations.

The training progress of ML agents can be seen within the Anaconda prompt window, which gives a summary of agents reward every 10,000 steps. Training also generates a set of TensorBoard graphs which help me see the progress of training, here I will be concerned with cumulative reward. The ghost line shows the actual value, whilst the bold line is applied with a 0.5 smoothing value.


Select Anaconda Prompt (Anaconda3)

```
2020-05-03 06:31:10 INFO [trainer_controller.py:100] Saved Model
2020-05-03 06:32:45 INFO [trainer.py:214] eden_01: EdenLearning: Step: 410000. Time Elapsed: 3593.723 s Mean Reward: 85.100. Std of Reward: 6.503. Training.
2020-05-03 06:34:14 INFO [trainer.py:214] eden_01: EdenLearning: Step: 420000. Time Elapsed: 3682.271 s Mean Reward: 94.600. Std of Reward: 6.151. Training.
2020-05-03 06:35:41 INFO [trainer.py:214] eden_01: EdenLearning: Step: 430000. Time Elapsed: 3769.774 s Mean Reward: 87.200. Std of Reward: 4.792. Training.
2020-05-03 06:37:06 INFO [trainer.py:214] eden_01: EdenLearning: Step: 440000. Time Elapsed: 3854.332 s Mean Reward: 91.600. Std of Reward: 7.499. Training.
2020-05-03 06:38:30 INFO [trainer.py:214] eden_01: EdenLearning: Step: 450000. Time Elapsed: 3938.430 s Mean Reward: 87.800. Std of Reward: 8.716. Training.
2020-05-03 06:38:30 INFO [trainer_controller.py:100] Saved Model
2020-05-03 06:39:54 INFO [trainer.py:214] eden_01: EdenLearning: Step: 460000. Time Elapsed: 4023.040 s Mean Reward: 98.800. Std of Reward: 9.075. Training.
2020-05-03 06:41:19 INFO [trainer.py:214] eden_01: EdenLearning: Step: 470000. Time Elapsed: 4107.161 s Mean Reward: 85.000. Std of Reward: 24.236. Training.
2020-05-03 06:42:41 INFO [trainer.py:214] eden_01: EdenLearning: Step: 480000. Time Elapsed: 4189.839 s Mean Reward: 96.200. Std of Reward: 9.185. Training.
2020-05-03 06:44:04 INFO [trainer.py:214] eden_01: EdenLearning: Step: 490000. Time Elapsed: 4272.590 s Mean Reward: 90.900. Std of Reward: 8.203. Training.
2020-05-03 06:45:27 INFO [trainer.py:214] eden_01: EdenLearning: Step: 500000. Time Elapsed: 4355.295 s Mean Reward: 90.700. Std of Reward: 6.694. Training.
2020-05-03 06:45:27 INFO [trainer_controller.py:100] Saved Model
2020-05-03 06:45:28 INFO [trainer_controller.py:100] Saved Model
```

Figure 9: Stage 1 Results - Command Line

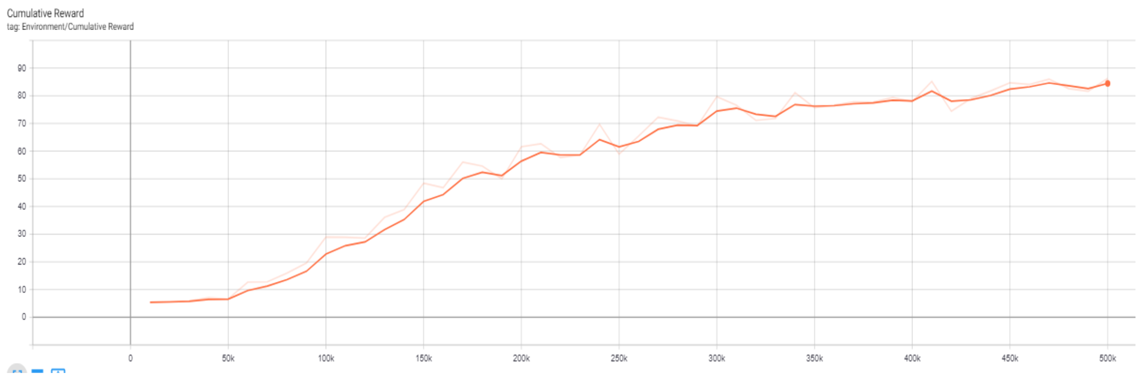


Figure 10: Stage 1 Results - TensorBoard

The Eden started at an average reward of 4.4 over the first 10,000 steps, whilst managing to reach it's maximum training score of 98.8, with the average increase in mean reward trailing off towards this point it would seem that this is the limit of our agent's capabilities under its current circumstances. A peculiar behaviour can be exhibited by the ML agents where they would circle the food which they were targeting. This came about due to a tendency for the neural network to instruct agents to apply forward motion while rotating, causing agents to orbit their target under Unity's physics. Interestingly, while agents also showed the capability to stop and turn in position, in this scenario they would often decide to abandon their target in favour of a nearby food source, to quickly return for another attempt at the target from a more direct entry angle. As expected, ML agents far outperformed traditional agents at the task. I gave each agent one minute to collect food at normal speed, ML-Agents managed an impressive 49, while traditional agents scored just 14. The results of this test should be taken with a pinch of salt however, as both food and Edens are initialised at random positions around the environment with each reset of the area. This extension test is purely to confirm that EdenAgents truly do perform better than EdenTraditionalAgents.

What I did not expect here was for the implementation of the ML-Agents at this early stage to actually be easier than that of the traditionally controlled agents. Whilst this takes some knowledge of machine learning and fine-tuning of neural network parameters, I found this far easier and quicker than working with Unity methods and attempting to get them to interface with one another to accomplish the task I had imagined for them. In order to create the logic that drives the EdenTraditionalAgents within this 3D environment, one must be able to define an agent's actions step-by-step, and do so utilising Unity's physics. For the extensive work that needs to be done in order to replicate this simple behaviour, the resulting behaviour of traditional Edens is rather crude and unbelievable, as there is no rationale to their relocation in pursuit of new food sources. Traditional Edens would often bump into walls and get caught in corners, much unlike their machine learning counter-parts.

7.4 Part 2 - Revised Vision Mesh

I earlier presented a diagram that depicted a design for what the model might look like. In this, I depicted an agent's range of perception to be limited to an arc. I have chosen to include this alteration to the model as it presents an opportunity to assess how each of agents action space can be expanded, and also presents an interesting issue I will discuss later within the results for this section. Now, agents of both types will be required to consider rotation when seeking out food.

7.4.1 Traditional

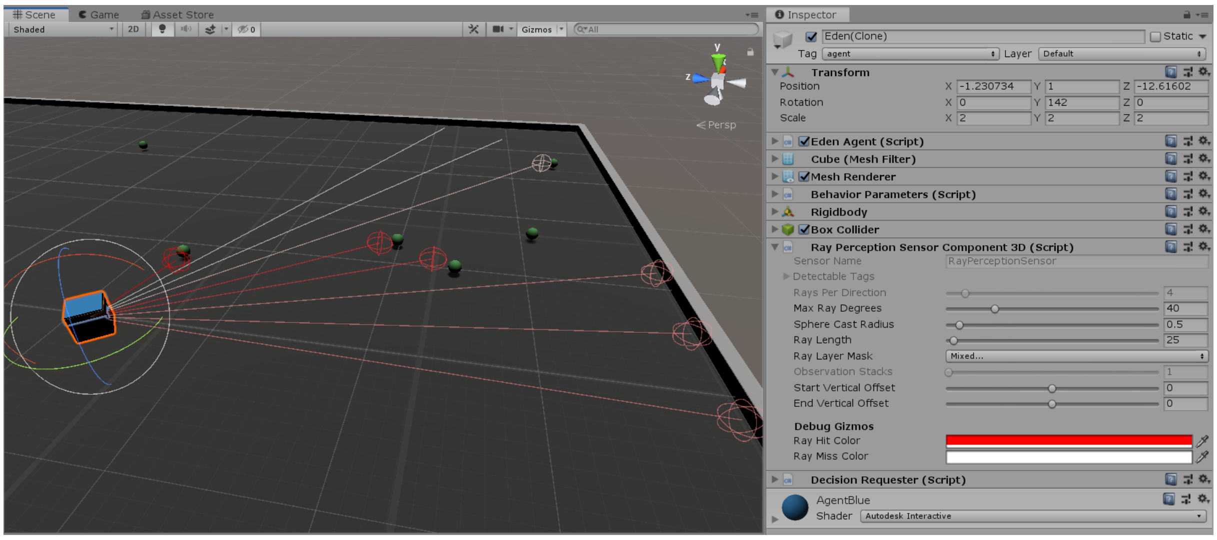
The functionality behind agents vision is simulated for traditional agents in the previous example through the use the OverlapSphere method, which allows me to retrieve the colliders of objects in the vicinity of the agent. However, I now need for agents to only consider colliders within a certain angle, determined by the agents current position. Thus, to keep the implementation simple I will still be utilising the OverlapSphere method in order to gather the colliders within range. Previously, the use of the OverlapSphere method was followed by logic that utilised the resulting information to work out what decision agents would make. Here I will be inserting new logic between this and the use of the OverlapSphere method, which will take the vector of colliders generated from this method and return a new vector only containing those colliders that fall within our agents new vision range. To do this, I will now iterate through the collider vector returned by the OverlapSphere method, for each collider getting the current position of the GameObject that it is attached to. For each GameObject then detected by the agent, I take its position and perform a check to see if it falls in range based on the agents current forward transform by calculating a dot product, if indeed falling within a range of > 0.5 which simulates a 90 degree arc, I continue on to again calculate which of the in range food objects is closest, which can then be used as described in the previous stage.

Currently, however, agents are not capable of appropriately making use of their new perception range as they are not capable of rotation. Thus, I need to create functionality that will allow agents to rotate and also decision logic for agents to dictate when this happens. I am able to make tradition agents rotate towards their food using the LookAt() method. This method takes and objects transform and rotates it to face a target transform, which is passed to the method as a parameter. Much like in the previous stage, when agents are in search of new food sources, they will be programmed to move randomly using a variable that updates every 2 seconds to a new random value.

7.4.2 ML Agents

As mentioned, I have created a perception field for the traditional agents where their vision is restricted to an arc. I do not have to make any great changes to our ML agents in order to mimic this, as these agents functioned with ray casts originally. The perception field created for ML agents is entirely handled by the MLAgents package, as such the variables that dictate the number and positions of rays that our agents cast must simply be adjusted. There are no changes that need to be made to our agent's actual observation space. What this change does alter, however, is the requirements for agents action space, as agents now need to consider rotation in order to efficiently collect food.

To make this alteration, I take the functionality that was defined for our traditional agents that allows them to rotate and include it in the class of our ML agents, allowing the value that is passed to it to now be decided by the neural network through the action vector. By just making this addition to agents action space, they should be able to now learn how to rotate in pursuit of food.



```
Eden(Clone)                         Static
Tag  agent                Layer  Default
Transform
  Position   X -1.230734   Y 1     Z -12.61602
  Rotation   X 0           Y 142   Z 0
  Scale      X 2           Y 2     Z 2
Eden Agent (Script)
Cube (Mesh Filter)
Mesh Renderer
Behavior Parameters (Script)
Rigidbody
Box Collider
Ray Perception Sensor Component 3D (Script)
  Sensor Name            RayPerceptionSensor
  Detectable Tags
  Rays Per Direction     4
  Max Ray Degrees        40
  Sphere Cast Radius     0.5
  Ray Length             25
  Ray Layer Mask         Mixed...
  Observation Stacks     1
  Start Vertical Offset  0
  End Vertical Offset    0
  Debug Gizmos
  Ray Hit Color
  Ray Miss Color
Decision Requester (Script)
AgentBlue
Shader  Autodesk Interactive
```

Figure 11: Restricted Vision

7.4.3 Results

Here I follow the generally the same training and testing format as the last stage, a single agent alongside 30 respawning food sources. Agents perception fields still have a 25 unit radius, however, they are now restricted to an arc that totals 90°, with 22 rays per direction. I will first discuss the results of the ML agents training process, before going on to discuss how the models compare.

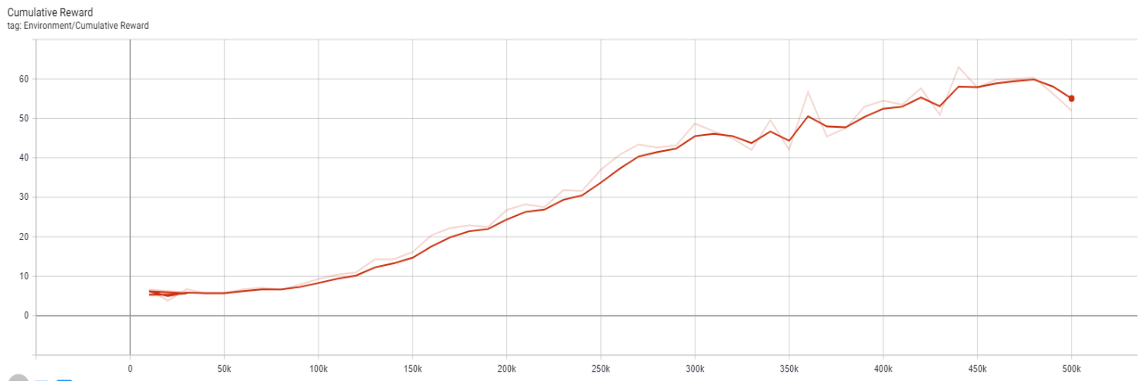


Figure 12: Stage 2 TensorBoard Results

From a starting average reward of 4.0 within the first 10,000 steps, the agent climbs up to an average reward of 71.7 after 47 minutes training time. I again repeated the previous test, allowing agents 60 seconds to gather as much food as possible. Again, ML agents far out preformed traditional agents with a score of 38 to 19. Surprisingly, traditional agents showed an improvement in this task, however, after running similar tests it is clear that this is simply due to chance, based on the random spawning locations of the food objects in combination with traditional agents random movement when not able to detect a food object. Unexpectedly, ML agents also exhibited an unforeseen behaviour. The purpose of our model is not to make discoveries on interesting on behaviours, but none the less over multiple iterations ML agents seemed to pick up a consistent strategy. Agents would learn to constantly circle the environment in the pursuit of new food targets. This is a very effective method for discovery, however, what is most interesting is that over separate training sessions, the agents chose to do this in an almost exclusively clockwise rotation. Agents would often double-back on themselves in order to pick up food sources they may have missed or may have respawned behind them, but then always continue back to this counterclockwise rotation. Agents would also stick to the inside perimeter whilst doing this, as to maximise the area of their perception field. This behaviour is well defined within agents and clearly visible, this contrasts greatly to the traditional agents, which display a random pattern of exploration, often bumping into the boundaries of the area due to the underlying logic that they are governed by. It is interesting that the machine learning agents have developed a believable behaviour whilst in the pursuit of efficiency.

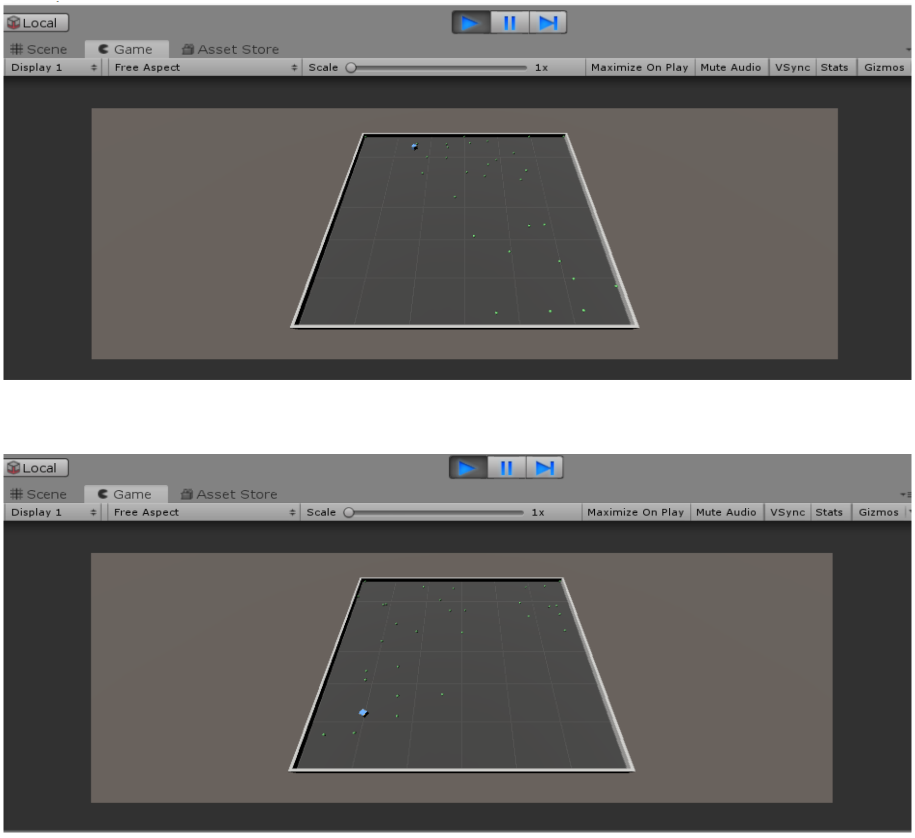


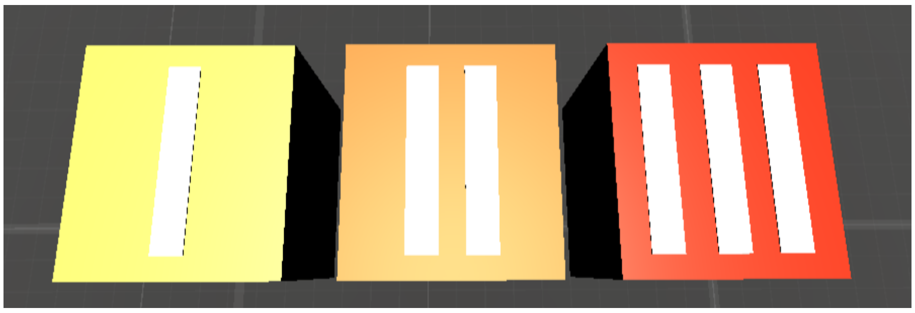
Figure 13: Stage 2 Agent Behaviour

Above shows a figure of an ML Eden in action. In the top image, it can be seen that there is a sparse area where no food objects can be seen, this is the path the agent has just taken. It can be seen that ahead of the agent, there is a large distribution of food object, which makes sense considering the random food spawning locations coupled with the agents cyclical route around the training. The second image is captured a short few moments later, it can be seen that the agent follows this circular path, clearing sections of the area as it goes.

7.5 Part 3 - Priorities

The next layer of complexity I wish to add to the model introduces the need for agents to prioritize. Here, I alter food sources so that they now have associated values. On collision with a food object, an agent will increment its food collected variable by the amount of food held in the pile. With sources of food within the environment now being held in piles of different amounts, the target behaviour is for agents is to most efficiently collect the largest amount of food.

The easiest way to achieve this so that it can later easily integrate into our ML-Agents logic, is to create a new prefab object for each food source. With this, I also decide to create a new model for each of these different food piles so that we can easily differentiate between them. I have created a new method within EdenArea that controls the creation of these objects within the training area. This method is called CreateNumberedFood() and takes three integer parameters which dictate the number of each food source to spawn.



7.5.1 Traditional Agents

Suppose that there were now two piles of food of different masses(Which we'll name A and B) at opposite sides of some agents perception range, yet both A and B are an equal distance to the agent. Under current circumstances our agent could not be aware of either food sources mass and as such would neglect to take this into consideration when deciding what its next action is. How can we represent this new change to agents? It is important to note that there are multiple ways we could attempt to tackle this.

In my implementation, I decided that agents would simply prioritize the pile with the most food. Agents would consider all food within its perception range, and aim to collect the largest. If there is no pile within range, the agent re-positions as normal. If there are multiple sources of food within range of an agent, agents simply compare the tags of the food object, setting the largest as the target destination. If there are multiple food objects of the same amount, agents are told to collect the closest. I already have most of the code to handle this functionality in place. I must simply make an alteration to agents logic in order to make them favour the largest food pile in range. To do this, as each varying sized food source now has a unique tag that indicates its size, I must edit the logic I have defined previously. This previous logic checks that the GameObject belonging to each collider has the tag 'food', before calculating it's distance to the agent. Now, I will have to replace this so it checks for items that have the tag 'food_one', 'food_two' and

’food_three’. I count the number of appearances of each of these tags within the collider vector, then use these counts to determine what happens next. If the number of colliders that have the tag ’food_three’ is > 1 then I calculate which is the closest and set this as our agent’s target. For efficiency, if the number of colliders is $= 1$ then the agent has only detected a single food source and thus does not need to check its distance. If the number of occurrences of this tag is 0 however, I simply allow agents to move on to consider colliders with the tag ’food_two’, and so on until a target is found.

7.5.2 ML Agents

Should an agents perception ray collide with an object within the scene then the agent is able to observe its object tag (If we have made this available within the Ray Perception Sensor Component 3D), and be able to relate attributed award back to the data gathered. Now, I also want our agents to be able to observe the amount of food in a pile. The simplest way to do this is to create a series of new tags for agents to observe that will correlate to the amount of food is in a pile. I have already created prefab objects for each food pile that agents will be able to collect, these prefab objects are then applied with a correlating tag. e.g. food_one, food_two and food_three.

I must only slightly alter agents reward function for them to be able to adapt to this alteration. Previously, the reward function attributed reward to agents whenever they collected some food. Instead, I now want to attribute reward to the agent dependant on the amount of food they have collected. I alter the OnCollisionEnter() function so that it now scans for collisions with each of the different food tags, and attribute increasing reward for the collection of larger food piles.

7.5.3 Results

I think that it is safe to say that ML agents were able to reach an optimal strategy for this task within the training time when we look at the training results. It can clearly be seen that there is a plateau in the increase of average reward towards step 500,000. Agents were much quicker at training for this task over multiple training periods. I believe that the added complexity here provided agents with more data to be able to relate to the gain of reward, thus increasing their capability for training. This training session is named eden_04. This is as the training session eden_03 is reserved for a study that I conducted in between these two sessions which I will discuss later.

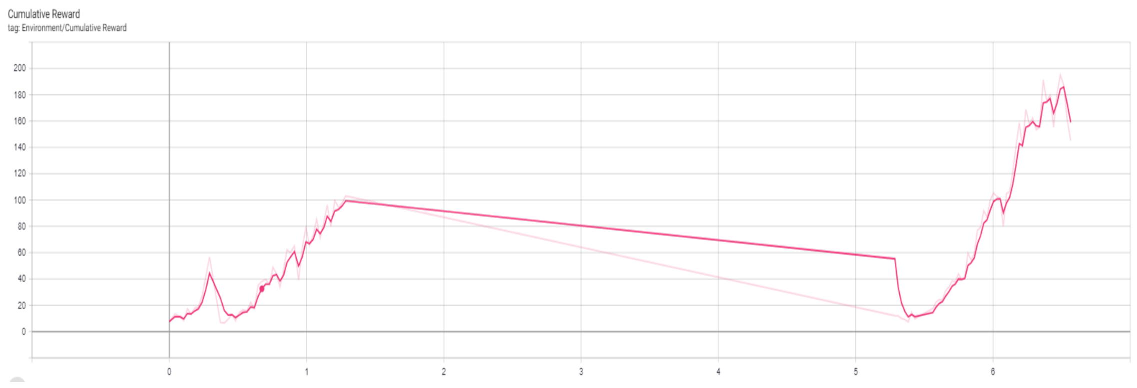


Figure 14: Stage 3 TensorBoard Results

During initial training of agents at this stage of development, whilst watching the results of training being produced, I hypothesised that agents could train faster with experimentation of the training parameters. This is where I decided upon the final training parameters that I had discussed earlier, and retrained both previous stages with these parameters. I decided to instead display the results over relative time, which details the progress of agents training across a timescale rather than across a number of training steps. This allows me to show the initial training session, which I halted, followed by the second training session which was completed. It can be seen that during the initial training, whilst agents did indeed make a steady improvement, they did so at a rate which seemed rather slow. It can be seen that during the first training session, agents would reach a cumulative reward of 100 after 1 hour and 17 minutes. During the second training session, agents reached this same level of accomplishment after just 42 minutes, a drastic increase in training speed. This is a result of altering the level of entropy introduced into agents decision making, which is controlled by the beta parameter, which I increased to 5.0e-3.

I again decided to pit agents against each other in a timed test. Much like before, agents were given 60 seconds to maximise the amount of food they could collect. This time, as food sources have different values, we will be reporting agents cumulative achievement, I will also report the number of each food source that each type of agents was able to collect. For this test, I utilised 30 respawning sources of food, broken down into the following amounts: Single Food - 15, Double Food - 10, Triple Food - 5. To no surprise ML agents again well out-performed that of their traditional counterparts. I also noted that at this stage agents ceased an odd behaviour that they displayed in earlier stages where they would orbit food sources. I believe in this case this has more to do with the alteration I have made to the food models. As the models for food are now cubes, the edges allow for something for agents to get caught on as they approach at an angle which would have previously made them 'enter orbit'. This is interesting, as it was not until this point that I realised that this behaviour that agents had previously engaged in could have been a factor of the models rather than a habit that they picked up through training. ML agents also displayed the behaviour I described in the previous section, where they would patrol the inside perimeter of the area in a counter-clockwise fashion.

7.6 Part 4 - Obstacles

The final layer of complexity that will be added aims to further explore and round together the ideas that I have already discussed. My hypothesis is that this addition will highlight how machine learning can allow for highly adaptable agents.

Here, I simply add a number of obstacles to the scene that agents will have to traverse around in order to complete their resource collection task. I also wish for agents vision to be now be blocked by these obstacles. In order for agents to efficiently collect food, they will now have to strategise so that they minimise their time spent with a blocked perception field so that they can effectively search for food. I have laid out the area so that there is a number of obstacles around the area, but also a specifically designed enclosed area. I hypothesize that ML-Agents will be able to regularly enter this enclosed area in order to collect food that may have spawned here, whilst traditional agents will only be able to enter this enclosed area by chance due to their logic.

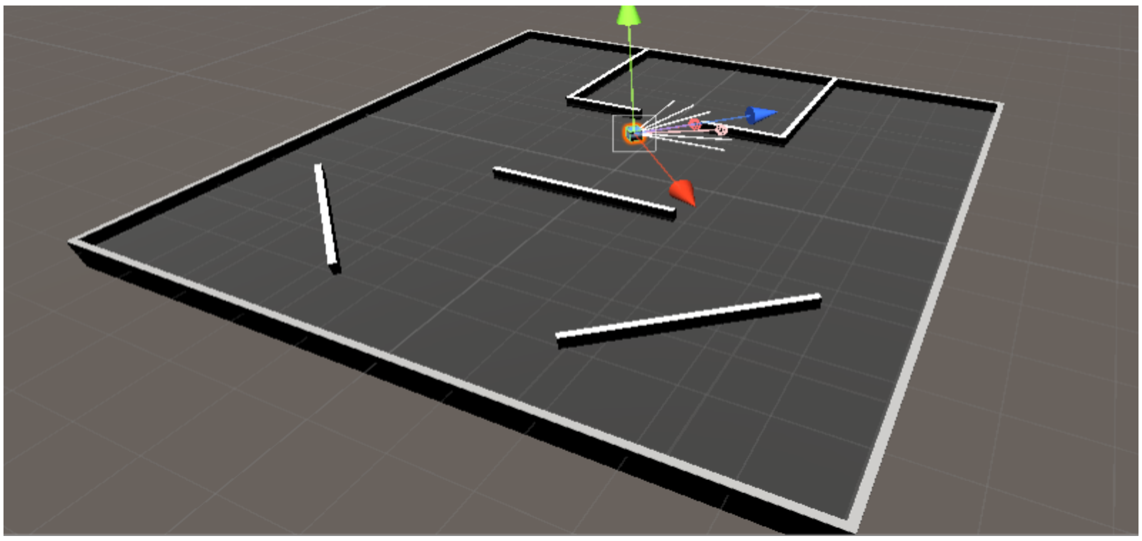


Figure 15: Obstacles

7.6.1 Traditional Agents

This is a complex task to handle for traditional agents, and also many ways I could attempt to tackle this. One such way would be to set up pathfinding for agents via Djikstras algorithm, however, the model is built under the assumption that agents are only aware of the area within their immediate perception space. As such, I am limited in options, and can only hope to adopt some form of brute force approach to obstacle avoidance. As such, I opt to not alter the logic of these agents, which is highly fair at this stage of development, as the ML-Agents will also be seeing very little change. All that needs to be done for the traditional agents is to alter their perception field so that it is now blocked by obstacles.

To do this, I will be extending upon the method described in stage two. The functionality behind agents perception field is currently simulated by first using the OverlapShere method to return colliders within range of an agent, and then checking that each of these objects are within the desired arc with respect to the direction of the agent, before going on to calculate the closest collider. To allow agents vision to be inhibited by agents, for each collider that satisfies this directional check, we cast a ray between the agent and each collider. Any rays that are then cut off before they reach the target food object indicates that this target should not be visible to the agent as there is either a wall or another food object blocking this path.

7.6.2 ML Agents

Machine learning agents are able to detect the obstacles on collision with rays. They are able to differentiate obstacles from other objects within the scene by the 'wall' tag we have given them. Thus, I only make a small alteration to the agent's observation space within the code. The CollectObservations(VectorSensor sensor) method within the agents class allows me to pass data to the agent that doesn't come from its Ray Perception Sensor 3D component for it to observe. Here, I allow agents access to added observations by setting a Boolean entitled vectorObs to true, allowing it to consider it's current position by adding the transform.position of its RigidBody component as an observation. I hypothesise that this will allows agents to relate different positions around the area to the positions of walls, slightly improving its performance. Thankfully, ML-Agents perception field is handled by the Ray Perception Sensor 3D component, and these rays are already blocked by obstacles within the scene.

7.6.3 Results

From the training results, it can be seen that agents had more of a struggle training for this task. Throughout training, agents would often show periods of both small improvement and regression, before sporadically making a large improvement in their ability for resource collection.

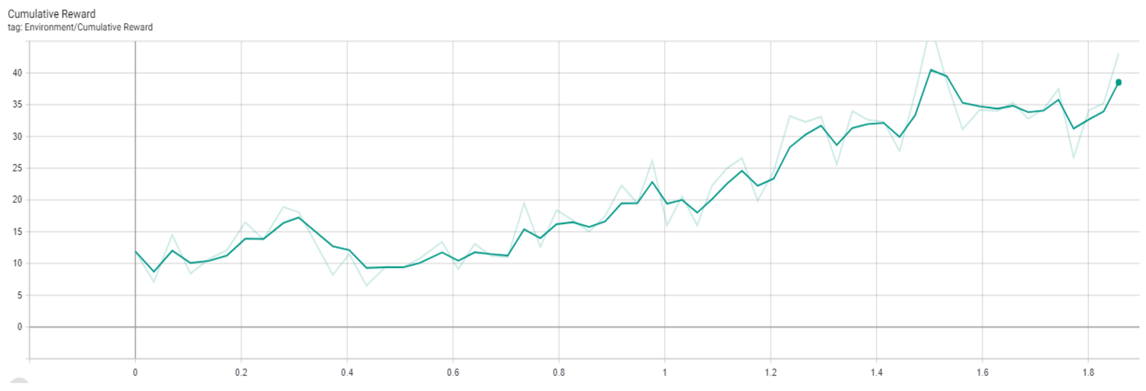


Figure 16: Stage 4 TensorBoard Results

The ML-Agents show a good ability to both navigate around obstacles and clear enclosed areas effectively. Agents are able to easily navigate into the enclosed space and around the obstacles that I have set out. Upon entering an enclosed space the agents would perform a few sweeps of the area to ensure it has found all the food that might have spawned within this space before relocating to a new area. However, agents would often quickly revisit areas they have recently visited, which would be still relatively void of food after the agent's last visit, hampering their efficiency. This occurs as agents have no concept of memory, and are unable to correlate data with time, thus not being able to recognise they had just visited a location and that it is unlikely that new food would have spawned within this time.

7.7 Can We Break It

In this section, I look at if it is possible for us to re-purpose agents for the use of testing. I will be altering agents reward function with the aim of breaking their environment, in the hopes that the agent will be able to discover bugs.

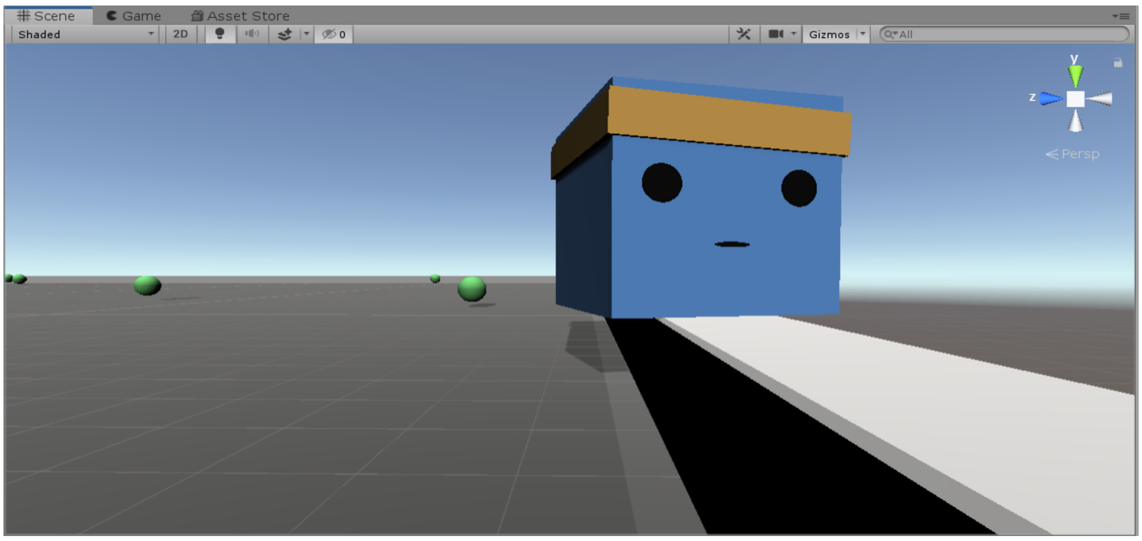


Figure 17: An Agent That Has Found an Environment Bug

Within the first stage of development, training of the ML agents leads to the discovery of a bug within the physics of the training area. In the earlier stages of training when agents are trailing different behaviour, they would sometimes clip over walls and thus off the edge of the training area. This presents the opportunity to explore an idea which I discussed earlier. Within the OpenAI paper, agents exhibited an unexpected behaviour where they would take advantage of a bug in the physics of their environment, giving me an idea for the use of ML agents and environment testing. My hypothesis is that if I alter agents reward function so that they gain reward every time they pass into negative y coordinates, and reset their position after they have received some reward, then our agents should be able to learn and convey to us where and why this bug occurs.

7.7.1 Result

Agents were indeed able to learn how to replicate this bugged behaviour consistently under the new reward function. Upon watching the agents learn, it is easy to see how this bug is triggered. Agents quickly learn that if they adjust their rotation at maximum speed whilst applying forward force into a wall, then the agents would clip over it. Here the combination of ABM and ML has helped in highlighting that a bugged strategy is possible and discover how this takes place so that a fix can be derived. In this case, as this happens at any wall on the scene, it is easy to narrow the bug down to the height of the bounding walls coupled with Unity's physics.

8 Discussion

Whilst I have chosen to represent the final model I have developed in 3D, I had initially begun development of a 2D model. Traditionally, ABM simulations within the established literature have always been only represented mathematically or within 2D. However, recent studies have begun to explore the use of 3D models. There are a number of trade-offs here, as I have mentioned, at first we had decided to develop a 2D model due to concerns of the computational power needed, as I developed the model on a system with limited capability. However, there were a number of advantages the use of 3D models could offer. One such advantage is the incorporation of full 3D physics, which can be seen within OpenAIs paper on emergent tool use. This allows researchers to explore agents interactions with real-time physics, or with the use of machine learning, it is possible to develop agents that are capable of manipulating physics to achieve goals, such as learning to walk. 3D models also offer another dimension to encode visualisations. These models can encode data on all sides of the object, and in some cases around the object. For example, in this case, I could provide an indicator above each agent that tracks its capability in collecting food. The reason we decided to develop a 3D model, however, came down to the number of packages that are provided for 3D Unity development as compared to 2D, meaning development in 3D proved easier in practice.

With the development of a suitable reward function for agents, I came to realise that this reward function should attempt to generalize towards the target end behaviour as much as possible. Earlier, during stage 1 of development, I offered a couple of examples of how a reward function for the task at hand might be designed, stating that one possible option would be to attribute reward to agents for getting closer to some food object. This is the first implementation that I attempted, with the reasoning that agents would be quicker to train in order to achieve the target behaviour to a high standard. However, I quickly noticed a case of an unexpected emergent phenomenon due to the design of the reward function. Upon bumping into each other on their route to a target food source, it seemed that agents would realise that they can gain reward more efficiently by repeatedly moving towards and away from the same food object, rather than collecting it and having to find another.

Generally, when attempting to construct some ABM simulation, there are a number of pressing questions when it comes to the choices made by modellers during implementation. When defining agents actions through the use of traditional rule-based techniques, how do we decide what is the most efficient/ plausible method for our agent to achieve its intended task? I provide an example of this dilemma that I encountered within the development of my model:

- In part 3, I wished to alter the model such that agents would now consider the size of the resources within their perception. This leaves us with the question, what is the most efficient way for agents to go around collecting food? Should agents favour closer or larger food sources or some combination of the two?

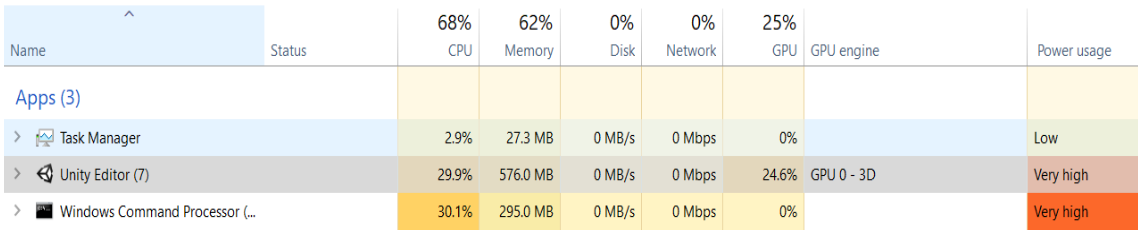
With this example, I showcase a concern that is raised within the literature. As modellers, the model that we build is entirely dependent on the underlying assumptions that we make. As such, our assumptions should be subject to critical evaluation for insights taken from any model to be considered valuable. Within the development of this stage, we decided that agents would simply prioritize the largest source of a resource, akin to that of the early rules with Growing Artificial Societies. ML-Agents provide an advantage here, as with the use of reinforcement learning, agents are now able to devise their own strategy to most efficiently maximise their reward function. Machine learning can be of help here as instead of defining the exact rules on how our agents should interact with their environment we can instead take a more goal-oriented approach. This means that developers no longer need to completely understand how the underlying decision making should function in order to develop an ABM simulation. This opens up new avenues for potential research, as this allows researchers to utilize ABM as a tool to explore the emergence of phenomena within enclosed systems, without necessarily being required to understand the phenomena itself prior to completion of the model in order for said phenomena to be exhibited by agents within the simulation.

Another advantage that this also showcases is how the adoption of ML agents can also ease the strain on development. With the development of traditional agents, one is required to provide means for agents to not only calculate whether one source of food is larger than another, alongside logic that tells the agent when to apply this and what to do with the resulting information. Not only are we left with the previous question of how can this target behaviour be achieved, but I must also endeavour to implement it. As displayed, for this particular stage, this is a complex behaviour to simulate. For ML-Agents, it is possible to achieve this target behaviour simply by adjusting agents observation space, greatly easing the strain on development. As the complexity of the behaviour model builds, it can also be seen that the difficulty in establishing logic in order for traditional agents to accomplish these target behaviours increases drastically. As mentioned, our implementation here required calculations to determine whether one food source is larger than another. ML-Agents both greatly eases the strain on development here as we only need to consider our agent's observation space and reward function, also, developers do not require an explicit understanding of the underlying mechanisms. To further emphasise this, suppose I wished to alter the model with the target to allow agents to consider other agents that may be competing for the same food source, and to abandon the food source if the competing agent is going to win the race to the location of the food.

For traditional agents, this would require an understanding of time/distance functions in order to implement the logic to handle this. However, I would hypothesize that ML-Agents could adapt to this alteration simply by expanding the agent's observation space.

One caveat here, however, is that researchers do necessarily wish for agents to exhibit the most efficient behaviours possible at all times. For example, humans are known to be not the most efficient creatures within a number of areas. As such, within models concerned with problem areas such as social science, the behavioural aims of researchers may favour believably over efficiency. In this situation, plausibility is paramount. I believe that overall, the results that I have produced display that there is indeed a place for machine learning within ABM simulations, especially within the realm of the social sciences. Through the combination of these techniques, agents are capable of generating their own unique behaviours, with relative ease to the developer.

It can also be mentioned that this combination of the use of Unity in training a neural network is rather computationally expensive and time-consuming. Below, I share a screenshot of the performance of the applications involved during training at the first stage of the model's development.



Name	Status	68% CPU	62% Memory	0% Disk	0% Network	25% GPU	GPU engine	Power usage
Apps (3)								
Task Manager		2.9%	27.3 MB	0 MB/s	0 Mbps	0%		Low
Unity Editor (7)		29.9%	576.0 MB	0 MB/s	0 Mbps	24.6%	GPU 0 - 3D	Very high
Windows Command Processor (...		30.1%	295.0 MB	0 MB/s	0 Mbps	0%		Very high

Figure 18: System Usage

As such, I had to limit training to a smaller number of steps than generally used in fear the heat generated over such a consistent time frame would damage my system, this could potentially be a limiting factor in what ML agents are able to achieve. However, considering the system used to develop the model utilizes an integrated graphics chip, this usage is slightly better than I had anticipated.

Finally, in light of my research, I have decided to take a look back at a piece of literature that I have previously discussed. I earlier discussed a proposed integrated cycle for combining the areas of machine learning and agent based modelling, presented within the paper entitled 'Machine Learning Meets Agent-Based Modelling: When Not To Go To A Bar.' I found that Unity's ML-Agents general pipeline mirrors this integrated cycle, whilst encompassing all of the mentioned functionality seamlessly into one package, which validates the author's research.

9 Conclusion

In conclusion, throughout this project, I have primarily explored the use of machine learning within ABM. With this study I have conducted into control mechanisms for agents within ABMs, I believe I have highlighted that there is certainly a place for machine learning to be highly useful in further research within ABMs, with possible application across a wide array of problem areas. The model which I have built and discussed within this project is based upon the societal model within Growing Artificial Societies. By utilising reinforcement learning, agents have been able to develop effective strategies to accomplish each task I have defined. Utilising this method of control, agents are driven to maximise their efficiency. I have discussed how this does not necessarily equate believably, as such, refinements may have to be made to the technique used here in order to replicate behaviours that are not necessarily primarily concerned with efficiency, such as cultural exchange for example. However, for my purposes, the strategy of using a simple reinforcement learning algorithm sufficed. In regards to the specific method that I used, I believe this efficiency-focused behaviour generation could prove extremely useful for researching specific target behaviours or used to model specific situations within wider problem areas. For example, in regards to societal models, there are a number of specific behaviours that can be regarded as 'efficiency-focused', from resource collection to trading. There are also a number of wider application areas for ABMs that can be regarded in the same fashion, such as economic models and those concerned with pedestrian movement flow, like within a paper I discussed earlier 'Agent-based modelling of pedestrian movements: the questions that need to be asked and answered'. As such, when attempting to model some target behaviour, there should be some heuristics developed to aid developers in distinguishing the focus of the behaviour, in order to strategize the most effective method to mimic it in a believable fashion. Not only have I highlighted that machine learning can help modellers to achieve desired results, I also believe that I have displayed how utilizing machine learning can lead to a simpler development process as compared to traditional techniques, especially when the complexity of the target behaviour starts at a high level, or will increase throughout development. This is especially true through the use of the Unity ML-Agents SDK, which greatly eases development. With the final stage of the models development, I also displayed how ML agents may aid in bug testing an environment.

I have also touched upon visualisation within ABMs, opting to build the model in 3D under real-time physics. As displayed within the surrounding literature, there have long been concerns about visualising models in real-time due to the extensive computation power it would take, however, I believe that with recent advancements in technology, the future of ABMs will adopt this technique more. Machine learning also has a part to play here, as the choice of control mechanism alters what we as researchers need to be able to see within the simulation in order to understand the decisions of agents. For ABMs that make use of machine learning, I propose that careful attention is made in order to effectively visualise what makes up an agents observation space, in order to quickly and effectively provide researchers with the information needed in order to create valid insights on actions taken by agents within a simulation.

9.1 Further Work

With the inspiration for this project in mind, I believe that ABM and machine learning are destined to share a long and fruitful marriage within future works of research. Neural networks are named after the mechanism that they intend to model - the human brain. Agent-based models in their infancy were highly popularised because of their usage within the field of the social sciences, often utilized in order to help us further understand human behaviour. As such, it is a surprise to me that there has not already been an extensive study that attempts to replicate an entire social model which utilises some machine learning algorithm as a control mechanism for agents. Further work could also extend to the exploration of the application of this combination to wider problem areas, such as economic and evolutionary studies.

10 References

1. Helbing, Dirk and Balietti, Stefano, How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design (October 13, 2013). Chapter "Agent-Based Modeling" of the book "Social Self-Organization" by Dirk Helbing (Springer, Berlin, 2012), pp. 25-70.

2. Epstein, J. M., Axtell, R. (1996). Growing artificial societies: social science from the bottom up. Brookings Institution Press.

3. Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., Mordatch, I.(2019). Emergent tool use from multi-agent autocurricula. arXiv preprint arXiv:1909.07528.

4. Smith, E. R., Conrey, F. R. (2007). Agent-based modeling: A new approach for theory building in social psychology. Personality and social psychology review, 11(1), 87-104.

5. Helbing, D., Balietti, S. (2011). How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design Why Develop and Use Agent-Based Models?. Santa Fe Institute Working Papers, (11), 1-55.

6. Terna, P. (1998). Simulation tools for social scientists: Building agent based models with swarm. Journal of artificial societies and social simulation, 1(2), 1-12.

7. Silverman, E., Bryden, J. (2007, September). From artificial societies to new social science theory. In European Conference on Artificial Life (pp. 565-574). Springer, Berlin, Heidelberg.

8. Henein, C. M., White, T. (2004, July). Agent-based modelling of forces in crowds. In International Workshop on Multi-Agent Systems and Agent-Based Simulation (pp. 173-184). Springer, Berlin, Heidelberg.

9. Kerridge, J., Hine, J., Wigan, M. (2001). Agent-based modelling of pedestrian movements: the questions that need to be asked and answered. Environment and planning B: Planning and design, 28(3), 327-341.

10. Allan, R. J. (2010). Survey of agent based modelling and simulation tools (pp. 1362-0207). Science Technology Facilities Council.

11. Rand, W. (2006, September). Machine learning meets agent-based modeling: when not to go to a bar. In Conference on Social Agents: Results and Prospects.

12. O'Sullivan, D., Haklay, M. (2000). Agent-based models and individualism: is the world agent-based?. Environment and Planning A, 32(8), 1409-1425.