B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Slime World

By
Dylan Whelan

for
Mr. Joseph Corr

April 24, 2023

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

List of Figures

List of Tables

Chapter 1

Abstract

SlimeWorld is a Unity-based game which simulates the evolution of slime creatures in a 3d environment. The slimes have traits such as scale and speed and are controlled using neural networks. In an attempt to properly model evolution slimes inherit their traits and neural networks from their parents, with mutations made that are completely random and are not biased in any way by a slimes efficiency or inefficiency in its environmental niche, the guiding goal is that if a creature is most well suited to its niche, it will be the most successful at reproducing and hence it's children due to the inherited traits should have similar levels of success.

Chapter 2

Introduction

2.1 About This Project

This project aims to create an environment to simulate the evolution of simple creatures, controlled by neural networks. The repository for the project can be found here I have settled on this goal as I realize that neural networks as a concept are becoming an ever-more important tool to software engineers and also I personally find the subject of evolution interesting, as well as the ways it is shaped by the environment.

The goals of the project Slime World are as follows:

- To create a 2d plane for simulated life to live on.
- To populate the plane with creatures which can eat, starve and procreate.
- To include a system for the creatures to evolve throughout generations, passing traits along to their children.
- To have the creatures controlled by neural networks which themselves are also honed through this evolutionary process.
- To support multiple "species" of creatures, suited to different evolutionary niches
- To create a tweakable environment which can affect the evolution of the creatures.

My criteria for success are the ability to simulate the life cycles and evolution of these creatures, while also being able to incorporate an artificially intelligent controller for the slimes based on a neural network, which after being shaped

by their evolution through numerous generations, will also tailor itself towards whatever niche the creatures tend too.

Another factor that would be important towards considering the project truly successful would be ensuring that there are not only creatures tailored towards one niche, i.e. that not all creatures would develop towards a very similar outline.

2.2 Description of the end-product

Slime World is a unity-based game, developed for the Windows platform which functions as a simulator for evolution on a very small scale. It simulates slime-like creatures in a 3d environment which live on a flat plane. These creatures have no inherent motivations being governed only by the emergent behaviours of a neural network, shaped by a manifestation of survival of the fittest, wherein the traits and behaviours that lead to successful reproduction will naturally spread and evolve as they last across generations.

The simulation allows for the evolution of these creatures by providing the possibility for them to reproduce and have their traits inherited by their children. A creature's children will have traits inherited from their parents albeit slightly randomized, The slow changes in inherited traits over generations provide the potential for evolution in the population.

To emulate evolution more faithfully, there won't be any external motivations governing the evolution of the creatures. To this end, the evolution shall be purely random, with the hope that creatures with less ideal traits will be less likely to reproduce successfully and be selected out of the population, contrarily the more ideal traits will be more likely to reproduce as they will have an easier time getting an adequate amount of food to have children and avoid predation. In turn, it's expected that the same effect will be used to train the neural networks, with the slimes that display more effective behaviours in reaction to their environments, i.e. avoiding predators and efficiently finding food without expending too much energy.

Hence the simulation adheres to a concept developed by 19Th century English biologist Herbert Spencer, "Survival of the fittest" [3], in this case not simply applying to the fittest organisms lasting to the longest, but rather that the traits best suited to spreading themselves, are those best suited to last throughout generations of evolution.

The ability to tweak the settings governing both the environment and traits enables the user to see how different conditions can end up changing how evolution occurs, and what traits are selected for.

2.3 Inspiration for the project

I was inspired to do this project due to an interest in evolution and the factors that govern it. I thought it would be interesting to be able to see the process occur in real time and also to have an effect on it by manipulating different factors governing it. I was also heavily inspired by a video I have seen on the subject of simulated evolution, which can be found here [4] which first opened my eyes to the possibilities of simulating evolution using a virtual environment.

I decided to incorporate genetic algorithms into the project, both on the advice of one of my lecturers, Dr. John Healy, and due to the facts that it both heavily suits a simulation based on evolution and it's something becoming ever more important in the software industry and I felt it was something important to learn about.

2.4 Chapters Overview

- **Chapter 2: Methodology**

  The methodology chapter delves into the development methodologies that I chose to use in developing and testing this project alongside the different tools that I used to create the project such as my IDE and version control software.

- **Chapter 3: Technology Review**

  The technology review chapter delves into the research I did on the technologies that I decided to use in creating my project and why I decided to use them.

- **Chapter 4: Design**

  The Design chapter delves into the aspects of the system and its implementation in Unity, explaining how the various in-game systems work.

- **Chapter 5: Evaluation**

  The Evaluation Chapter compares how the end product compared with the goals I had initially set out with in making this project.

- **Chapter 6: Conclusion**

  The Conclusion Chapter follows on from the Evaluation in comparing to end state of the project to my initial goals and I talk about what I consider to be the biggest limitation in my approach.

Chapter 3

Methodology

3.1 Development Methodology

I used Agile as my software development methodology. I have chosen Agile as it is a methodology that is very commonly used in the software industry and I felt that it was important to get experience with the methodology.

3.2 How Agile was used in this project

To follow the Agile methodology, I planned and tracked my work using an Agile board on Jira by Atlassian, to this end, I made a list of all the issues or items of work that I thought essential to finish, ordered in terms of priority and expected difficulty and then I added them all individually to the Jira backlog, and once they were in the backlog I then portioned out the different to sprints to plan out my approach to development in a more organized manner. The usage of the Agile board it easier to approach my work on the project as it gave me more clearly defined goals to work towards, as opposed to having more vague ideas as to how I wanted to fulfil my goals and develop my project.

3.3 Research Methodology

For my research methodology, a great deal of the research I did was into different aspects of game development, particularly developing games on the Unity platform and hence the lion's share of the research I did was either through Unity documentation to uncover how to use different Unity native features, utilizing YouTube videos to educate myself on working to create a UI in Unity and I read a number of articles from the site Medium on the subject of developing games with Unity.

For the more academic aspect of my research that being my research into the technologies I wanted to use for my project and their advantages and disadvantages relative to competitors I made use of Google Scholar to find academic sources where possible to back up my decisions and I also did research through forums and articles to get a more first-hand account of developers experiences with the different platforms.

3.4 Version Control

For version control, I used GitHub to host my repository, as it was mandated by the college but also as it provided a very convenient solution to manage the different versions of my project, and provided the ability to revert changes that proved to negatively affect the project.

The usage of GitHub for hosting the repository had another key advantage as it enabled me to easily work on the project both on my desktop and laptop, as I could easily ensure the versions of the project on both computers were in sync whenever I envisioned I would be performing the next part of work on a different device I would push the changes to the repository and then fetch them on the other computer.

3.4.1 Advantages of GitHub Desktop

I used GitHub Desktop as the tool for handling all interactions with the repository as though I am familiar with Git and handling pushing, pulling and fetching from the command line interface, I found that GitHub Desktop was more pleasant to use for the processes as it made much quicker and easier to see what changes there were made to the existing files in the repository before each commit. It made it much easier to quash merge conflicts, which was something I frequently had difficulty with when using Git.

3.5 Testing

Due to the limitations of the Unity Test Framework, I concluded that it would be unsuitable for use in my project. As such I resorted to employing only manual testing. All testing was done using White Box Testing as I performed all tests manually, considering each test concerning the source code and the particular functions or systems that were being tested.

In the context of the project when I refer to White Box Testing, I mean that all testing was done with a close connection to the source code, including tests being

done every time I changed how any of the in-game systems worked or whenever I introduced new systems to add more functionality to the project. I performed testing in this manner even though it was laborious as I wanted to ensure that all added features worked within the context of the project and that I wouldn't end up discovering further down the line that there were issues with certain aspects of the code that would require entire sections to be rewritten.

The reasons that I judged the Unity Test Framework to be insufficient for the project were due to the many complex interactions present in a game environment which can't be satisfactorily tested using unit tests, in particular any interactions involving the Unity physics engine. There was also an issue in which many of the required tests would have been just as if not more difficult to write than the code being tested, particularly in the case of the multidimensional arrays which made up a large proportion of the work in developing the neural networks and which couldn't be easily tested due to how C# handles multidimensional arrays.

The disadvantages of my approach to testing were that manual testing was a rather slow process. It involved a great deal of time spent running and analyzing the game as it was running to ensure that all functions worked as expected and at certain points either due to insufficient attention on my part or a lack of consideration in what to look out for, certain bugs persisted after tests which I would only end up realizing much later, at that point taking more effort to solve the problems.

3.6 Development Environment

For my IDE I have chosen to use Visual Studio 2022 by Microsoft, Visual Studio includes a code editor which supports IntelliSense [5], a versatile code-completion aid, which can heavily boost a developers productivity by enabling the more write code more quickly, this also enables users to more easily keep track of parameters expected for methods or functions, and view any associated documentation enabling developers to stay focused and reduces the amount of hopping between windows or scripts that could be required while developing.

Visual Studio also includes native support for C# as C# was originally designed by Microsoft. This support includes IntelliSense support for C# native functionality such as built-in classes or functions and also the ability to compile code in C#, the built-in compilation feature being the factor that separates an IDE like Visual Studio from a code editor like the very popular Visual Studio Code.

Visual Studio is also easily configured due to the comprehensive install wizard, which fortunately for my own purposes contained a native package for development in Unity, including a debugger suited to Unity development alongside Unity-specific documentation and Intellisense packages, which sped up my development greatly as it enabled me to focus more directly on development.

Another key advantage of Visual Studio which I only started to make use of later in my project was its support for GitHub Copilot which I found useful for writing code more quickly as it could easily deal with the more repetitive aspects of writing my game or refactoring code.

Chapter 4

Technology Review

4.1 Game engines

In this chapter, I will be delving into the advantages and disadvantages of the technology I picked relative to the alternatives and ultimately the reasons why I picked Unity as my game development engine. The two game engines that I looked at were the Unity game engine and the Unreal game engine.

4.1.1 Overview of Unity

The Unity game engine is a game development engine created by Unity Technologies. It supports the creation of both 2-dimensional and 3-dimensional applications for over 20 different [6] platforms, including Windows, Mac OS, Linux Standalone, Android, and iOS.

A key advantage of Unity is its user-friendly interface and its accessibility making it useful to both novice and more experienced developers. The component-based architecture enables users to more intuitively manipulate in-game objects using assets and scripts to define their behaviours. The approach enables users to develop games more quickly by making it easier to focus on specific components of the project.

Unity uses C# as its scripting language, a widely-used object-oriented programming language which I will delve into more depth on further in the review as it was a large factor in my choice of project.

Unity includes a built-in asset store, which contains a variety of user-made assets, which can be anything from models and scripts to entire templates for systems. The asset has many built-in search filters to help users find the assets that they need for projects and there are both free and paid assets available.

Unity also has a very robust and clearly explained documentation [7]. Which includes a Unity Editor Manual Which provides a large variety of tutorials for

developers to learn how to utilize different aspects of the Unity editor and develop in-game systems and also the Unity Scripting Reference which contains details of Unity's scripting API, providing all the information a prospective developer needs for using any base Unity classes.

In addition to the excellent documentation that Unity provides, there is also a thriving community of game developers making written and video-based tutorials for Unity which can help users learn different aspects of game development, anything from the most basic of game development fundamentals to ways to solve more specific problems like handling the scaling of UI elements for different screen sizes and aspect ratios.

4.1.2 Overview of the Unreal engine

The Unreal Engine is a game development engine created by Epic Games. It has been used not only for the creation of games but also for a variety of other applications, notably including its use for the creation of visual effects on "The Mandalorian" Television series [8]. The Unreal Engine supports game development for 17 platforms, including Windows, Mac OS, Linux, Android and iOS. The latest version of the Unreal Engine at the time of writing is Unreal Engine 5[9].

A key advantage of the Unreal Engine is its visually impressive graphics as the Unreal Engine provides the capability of rendering extremely high-quality graphics in games, as showcased in games like Borderlands 3, Hell Let Loose and Back 4 Blood. As mentioned earlier, the high-quality graphics made possible by the Unreal Engine also make it a very popular tool for photo-realistic animations or visual effects in movies.

Unreal uses C++ as its scripting language and noticeably was also written in C++, this is particularly interesting as Epic has made the source code of Unreal Engine available, and the engine itself can be modified by skilled users to personalize it better for their goals. It also includes a robust visual scripting system called "Blueprints", though I won't delve into that as visual scripting systems have little carry-over to coding skills in other areas.

Unreal also provides a marketplace that contains a large variety of user-made assets, which can include anything from 3d models, and animations and even to entire templates for game-play components, which can either be free or paid and can help users develop games or other applications more quickly by providing an easy avenue for finding useful components instead of making their own.

Unreal provides quite in-depth documentation [10]. Which is split into a number of sub-topics to make it easier for users to find what they are looking for if they aren't entirely sure of what query to search, and includes a large section titled "Understanding the Basics", which exists for the purpose of helping to get newer developers up to speed and developing games on the Unreal platform.

Unreal also has a thriving community of developers providing a large number of written and video-based tutorials for new users to learn different aspects of game development as it is on the Unreal platform, including how to implement anything from more simple mechanics to more complicated concepts that developers should keep in mind when developing their games.

4.1.3 Comparison of the two Engines

Now that I have summarized both of the game engines, I will go into more depth in contrasting their different advantages and disadvantages and how they affected my choice of game engine for developing this project.

Ease of Use

One of the first aspects that I took into account when choosing my game engine for the project was the ease of use and learning curve, as I wished to use a game engine that would not require a great deal of learning before I could effectively use it for my project work, admittedly this biased me towards Unity as a result of my previous experience with it.

The Unity game engine is well known for its very short learning curve, being easy for users to pick up and start developing games in relatively short order, the aforementioned component-based architecture is very intuitive and is very easy to pick up for any programmers with experience in the Object-Oriented Programming paradigm. In addition, the large community of creators producing tutorials and guides for Unity-based development enables users to easily research anything they might wish to implement or any problems they might need to troubleshoot to solve them more quickly.

The Unreal Engine however has a much longer learning curve, Unreal Engine is not only written in C++ but also uses C++ as its scripting language with a variety of Unreal-specific scripting macros, and in many aspects, the variety of C++ used by Unreal differs to the standard variant of C++, and it's worth noting that according even to Epic Games themselves, the documentation isn't yet up to date for their C++ API as the documentation is an early work in progress [11].

The much easier learning curve for Unity alongside my existing experience with it ensured that particularly this category would leave me more inclined towards using Unity as the game engine for my project.

Sr No.	Parameters	Unity	GameMaker	Unreal	CryEngine
1.	Cross Platform	Consoles (Xbox, PlayStation, Wii U, Nintendo), OS or Desktop (macOS, Windows and Linux), Mobile devices (Android, Windows, iOS, Blackberry), WebGL	Consoles (Xbox, PlayStation, Nintendo), OS (Windows, macOS), Mobile devices (Android, iOS), HTML5	Consoles (Xbox, PlayStation, Switch), OS (Windows, macOS, Linux), Mobile devices (iOS, Android), HTML5	Consoles (Xbox, PlayStation, Oculus Rift), OS (Windows, Linux), Mobile devices(not supported)
2.	OS Support	Windows, Linux, Mac	Windows	Windows, Mac	Windows, Linux
3.	Programming Languages	C#, JavaScript, Boo	C#, C++	C++	C++, Lua
4.	Multifunctionality	2D, 3D	2D, 3D(FPS)	2D, 3D	3D
5.	Documentation	Best	Good	Good	Poor
6.	Difficulty Level (for Beginner)	Low	Moderate	High	High
7.	Artificial Intelligence	RAIN	Kynapse	Kynapse	Lua Driven AI
8.	Physics Engine	PhysX	Built-in	PhysX	Soft-Body
9.	Network/ Multiplayer	Supported	Supported	Supported	Supported
10.	Development Tools	Visual Studio, MonoDevelop	GameMaker Studio	BluePrint Editor, Visual Studio	FlowGraph, Visual Studio

Figure 4.1: Game engine comparison, courtesy of [1]

Graphics

This is a big area of comparison between the two game engines, with Unreal engine being regarded very well in the industry for its very impressive graphics system due to its far more advanced rendering system compared to Unity that out of the box can produce much more impressive visual fidelity and finer looking results, however, this factor was not very important to me personally and as such, the increased graphical fidelity offered by the Unreal Engine was not a significant enough factor to affect my decision.

Learning Value

As the goal of the final year project is to further my own skills in software development according to my personal goals, one of the largest factors as regards choosing game engine was in fact the associated scripting language. I have a goal in the future to get a career in .net-based development and as such Unity's use of C# was a major benefit for me in terms of learning the skills I desired to possess for my future career. And the fact that the C# used in Unity isn't too dissimilar from standard C# was a big bonus for me.

4.2 Languages

As mentioned earlier in the review language was a large factor in my choice for this project and the language that I intended to use was C#. C# is an object-oriented programming language developed by Microsoft which is used for developing desktop applications, games and web applications. It is frequently used to develop for Windows, iOS or Android. It is very heavily used in the software industry, according to a survey done by Stack Overflow in 2022, 29.7% of developers claimed to use it [12], meaning that it ranked as more popular than C and C++ in the same survey and less popular than Java.

C# is a strongly typed programming language, meaning that all variables must be declared with a specific type unless they are declared as either var or dynamic, variables specified as var will however still function as strongly typed and the type of variable is assigned at compilation based on how the variable is used. variables declared dynamic are instead resolved in run-time, however, I will not delve into these variables as I feel their use detracts from the advantage of a strongly typed language, that being that code is more readable and self-documenting, leaving less guesswork to the developer.

As C# is used with the .NET framework, it runs on the .NET Common Language Runtime which means that it avails of the CLR's built-in garbage collector. The presence of an in-built garbage collector ensures that I don't have to worry about the manual management of memory when writing code, it ensures that any unused objects will be cleared from memory avoiding memory leaks and improving the security of written code. The garbage collector works by checking if an object can be accessed through the application's code and if not, the memory used by the object is freed up, this behaviour can be modified using weak references, in that objects which can be accessed by code through a weak reference can still be collected using the garbage collector.

Benefits of Object-Oriented Programming

C# is an object-oriented programming language, meaning that code is largely classed into classes and objects, this naturally complements game development very well as all the individual components making up a game can easily be thought of as objects, and the nature of object-oriented programming with each class containing its own associated data and methods makes it much easier to develop game components in a modular fashion and then integrate with each other.

Additionally, aspects like encapsulation, meaning that the variables or properties that affect the internal state of an object can be isolated, meaning that we can ensure these instance variables that we wish to be private can only be accessed through the use of specific methods designed for classes. This makes the code more

reliable and less bug-prone as it makes it easier to keep track of the different ways that objects can interact with one another.

Other object-oriented principles such as polymorphism also make it easier to write more versatile code which can be modified in different ways depending on the particular goal, function overloading is the case that was most interesting to me, as with the purpose of the game being around evolution, there was the important aspect that the creatures in-game where e

The object-oriented nature also made it easier to implement different performance-improving features that I figured would be necessary for the project, such as object pooling, which due to inheritance meant a generic object pool could be written to store any kind of game objects/

4.3 Neural Networks

In this chapter, I will be delving into Neural Networks which are an important aspect of my project. I will be writing a bit about them and several key aspects and justifying why in the end I chose to use a rather simple implementation of one.

4.3.1 Introduction of Neural Networks

Neural networks which at least in part were inspired by attempts at understanding the manner in which natural neurons operate and work in organic organisms. Neural networks are a broad class of machine-learning models which can be used to recognize patterns and create systems which can be trained to perform certain roles or tasks either using pre-obtained training data or through live feedback.

Neural networks can be represented as directed graphs, and are most often composed of a variety of nodes or neurons which are linked together through connections which can have their own weights affecting the values of the outputs that they transmit. The 4.2 provides a sample structure of a feed-forward neural network, meaning a network wherein there are no cycles formed by any looping connections between nodes. This sample structure is a neural network with an input layer, 2 hidden layers and an output layer.

I have included an image of a simple feed-forward neural network, as in the context of my project I intend only to include a small feed-forward neural network without even utilizing the advantages of more advanced training techniques like back-propagation as I feel that they don't fit in properly with the goals of my project.

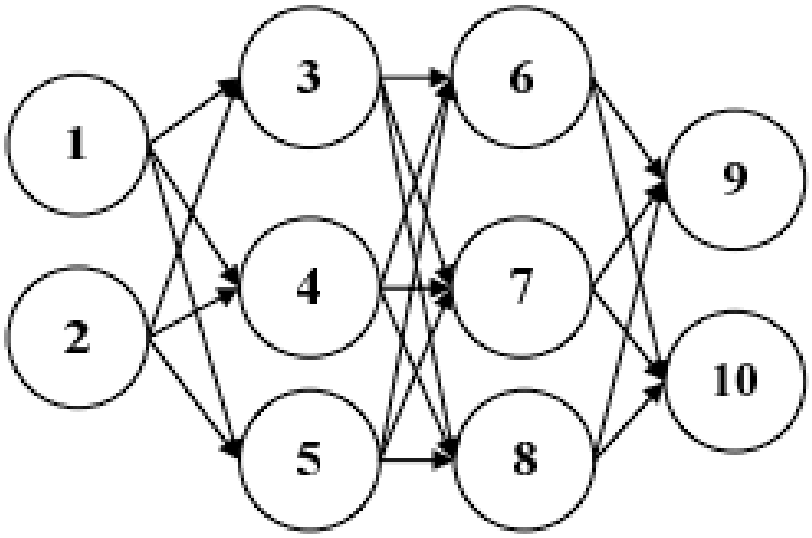

Figure 4.2: Sample Structure of Feed-Forward Neural Network, courtesy of [2]

4.4 GitHub Copilot

GitHub Copilot is an AI-powered coding assistant developed by a collaboration between GitHub and OpenAI. It is based on the Codex model developed by OpenAI [13], which itself was built upon the GPT-3 architecture.

4.4.1 Advantages of GitHub Copilot

GitHub Copilot can offer developers intelligent suggestions as they type, anything from singles lines of code to entire blocks of code, which if using the Copilot extension in your IDE can simply be used with auto-complete, for example by using the tab key in Visual Studio.

GitHub Copilot can also synthesize entire blocks of code or functions based on comments, taking the comments as suggestions and providing auto-complete suggestions that will meet the request.

GitHub Copilot also displays an apparent ability to learn from context based on a user's project, with the suggestions becoming more accurate and appropriate to the work being done if the user engages with Copilot by accepting relevant suggestions and rejecting those that are inaccurate or irrelevant to the developer.

An advantage of GitHub Copilot that was particularly relevant to me was that GitHub Copilot as of the time of writing this dissertation is available for free use by students, giving students an opportunity to get experience with GitHub Copilot

for free so that we will likely grow accustomed to its use and continue to use in professional careers as software developers.

4.5 Summary

In Summary, I have decided that I will use Unity as my game engine for development, with C# as the programming language of choice and that I will use a simple feed-forward neural network for controlling the creatures in my simulation.

Chapter 5

System Design

This project was written entirely within Unity so in this section I will be dealing with the structure of the Unity game and I will go into how different aspects of the game work with explanations and references to relevant snippets of code or algorithms used.

5.1 Scenes

For SlimeWorld there are two scenes, the main menu which the player enters upon starting the application and the game scene in which the simulation occurs. The main menu contains, buttons to enter the main game, to quit the game and also to open the settings menu where the parameters of the simulation can be altered using a set of sliders.

5.2 Menu Scene

The menu scene is the simpler of the two scenes and it is responsible for giving the users the opportunity to change the settings of the simulation and to launch the simulation. This image 5.1 shows the hierarchy of the scene in the Unity editor. As seen here there are two sets of menus, the "main" menu and the settings menu.

Main Menu

The main menu as seen here 5.2 is very simple containing the title of the game at the top of the screen and 3 buttons: Play, Settings and Quit. The play button sends the player into the Game scene, the Settings button opens the settings menu and the Quit button causes the game to close.

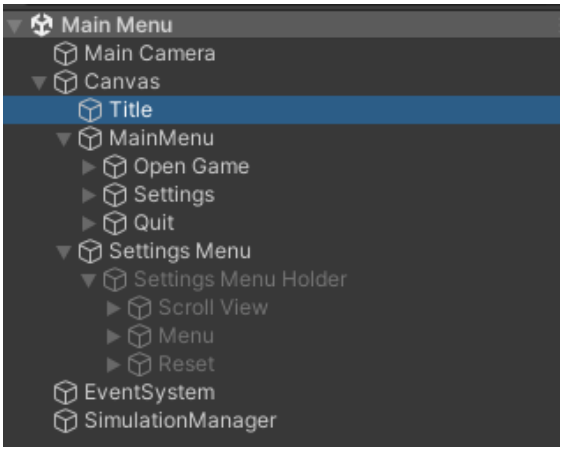

Figure 5.1: Main Menu Hierarchy

Figure 5.2: Picture of the Main Menu

Settings Menu

The settings menu as seen here 5.3 contains a scrollable view which contains the settings for the different parameters of the simulation, which are altered using the sliders. When the settings menu is opened, the SettingsMenu script gets all of the settings properties which are associated with the SimulationManager script and assigns them to the relevant sliders. When a slider is used, it calls the relevant method attached to slider, passing in its updated value and the updated value is in turn assigned to the relevant property of the SimulationManager and saved as a player preference. If the user clicks Menu the settings menu will be deactivated and the main menu will be reactivated. If the user clicks Reset, all playerprefs will

be wiped, resetting settings to default values and also returning the player to the main menu.

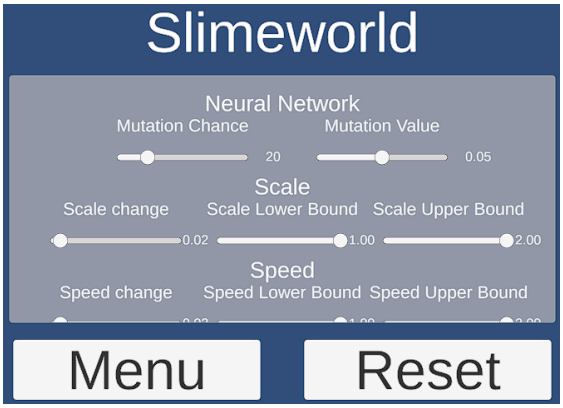

Figure 5.3: Picture of the Settings Menu

Simulation Manager

The Simulation Manager is a singleton script which is shared across both the Menu and Game scenes, but I shall put it in the Menu scene as in normal usage of the application, the menu scene is where it is initialized. The SimulationManager script is responsible for loading and storing saved user settings and handling any modifications of the properties used to store them.

The SimulationManager makes use of the singleton design principle to enable easier inter-script communication as access to its properties is required by many of the scripts in the game scene, including the SlimeManager script, FoodManager script and NeuralNetwork script.

The SimulationManager makes heavy use of C# properties to handle manipulation of settings and an example of one such property is found in this code snippet 5.1. Where the properties of the property make it easy and convenient to handle modification of _mutationChance variable and make it easy to determine when a property is modified.

```
1     private int _mutationChance;
2     public int MutationChance
3     {
4         get => _mutationChance;
5         set
6         {
7             _mutationChance = value;
8             PlayerPrefs.SetInt("MutationChance", value);
```

```
9            }
10       }
```

Listing 5.1: C# Property

5.3 Game Scene

The Game scene is the main scene of the game and is responsible for the simulation of evolution by providing a small environment in which the slime can spawn and eat and reproduce or die. This image 5.4 shows the hierarcy of the scene in the Unity editor.

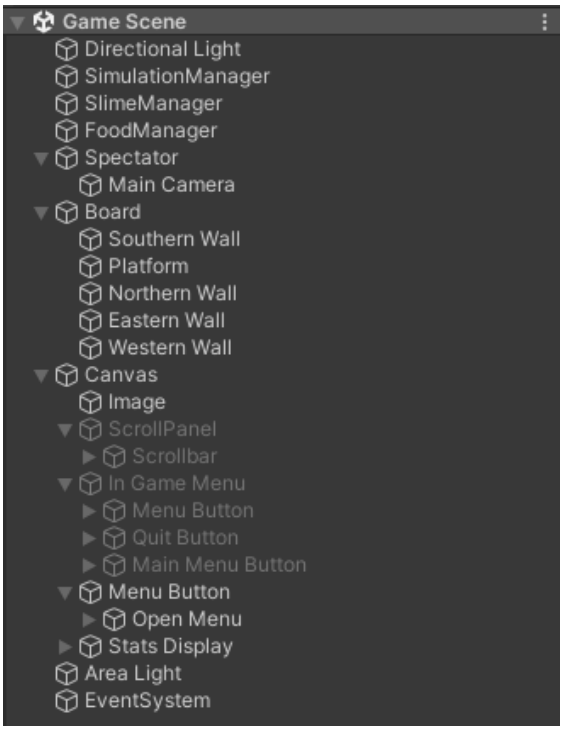

Figure 5.4: Game Hierarchy

5.3.1 Slime Manager

The SlimeManager is a singleton script which is responsible for the spawning and disabling of slimes, upon entering the game scene or if the number of slimes dips too low, the SlimeManager will trigger a method called SpawnWave, which will

spawn a number of slimes decided by the setting in the settings menu mentioned earlier.

The SlimeManager is responsible for the spawning of slimes both with and without parent slimes, and to this end, the SlimeManager contains an instance of ObjectPool which stores slime GameObjects. When the SlimeManager spawns a slime it first must calculate the position of the new slime, which is a random position anywhere on the board if the slime is parent-less and is within 2 units on the x and the y axis of the parent if the slime has a parent. The SlimeManager then gets an instance of the slime GameObject from ObjectPool, assigns it a name which is Slime_ followed by a number representing the number of slimes that had spawned before it.

For slimes without parents, the scale and speed values are determined randomly with regard to the lower and upper bounds for each value determined using the settings menu, a code snippet explaining this alongside the names is here 5.2.

```
1     spawnedSlime.name = string.Format("Slime_{0:00000}",
          _slimeCount++);
2         Slime spawnedSlimeScript = spawnedSlime.GetComponent<
              Slime>();
3
4         //float scale = UnityEngine.Random.Range(0.5f, 2f);
5         //float speed = UnityEngine.Random.Range(0.5f, 2f);
6
7         float scale = UnityEngine.Random.Range(
              SimulationManager.Instance().ScaleLowerBound,
              SimulationManager.Instance().ScaleUpperBound);
8         float speed = UnityEngine.Random.Range(
              SimulationManager.Instance().SpeedLowerBound,
              SimulationManager.Instance().SpeedUpperBound);
9
10        // pseudo constructor method for game object
11        spawnedSlimeScript.Init(scale, speed);
```

Listing 5.2: Parent-less Slime

For slimes with parents, the method is much more complicated and takes the parent slime as a parameter. Seen here is the code 5.3, some of which has been removed for conciseness, but as can be seen, the child slime takes the parent's generation plus 1, receives a copy of their slimeInfo and a copy of their neural network.

```
1         // generation = parents generation + 1
2         int generation = parentSlimeScript.GetGeneration() +
              1;
```

```
3
4           // parent's slime info is saved to be passed in to
                save variable
5           SlimeInfo slimeInfo = parentSlimeScript.GetSlimeInfo
                ();
6
7           // parent's neural network is saved to be passed in
8           NeuralNetwork neuralNetwork = parentSlimeScript.
                GetNeuralNetwork();
9
10          // As it is unwise to create instance of classes
                extending monobehaviours using the new command,
                the init command is used as a pseudo constructor
                after the gameobject has been instantiated
11          spawnedSlimeScript.Init(scale, speed, generation,
                slimeInfo, neuralNetwork);
```

Listing 5.3: Slime with Parent

The last responsibility of the SlimeManager as mentioned earlier is to deactivate slimes when they've either been eaten or rotted away and to this, the SlimeManager simply calls the DeactivateObject method on the slime in question, returning it to the object pool.

5.3.2 Slime

The Slime is the most important class in the scene and game, as it represents the creatures which are being evolved in the simulation. The slimes have a number of persistent variables associated with them, their scale and speed which are floats, their generation which is an int, alongside their slimeInfo which is a SlimeInfo and their neuralNetwork which is a NeuralNetwork. All of these are inherited from a parent if said parent exists.

The slime's scale represents its size and if a slime is 10% larger than a smaller slime, it can eat the slime upon collision and convert it to saturation, which is a measure of the number of calories the slime has. The slime's speed represents its movement speed. Both scale and speed have a metabolic cost in the simulation with slimes with higher numbers in either trait having a proportionally higher energy upkeep.

The SlimeInfo is a simple class used to store a slime's name, scale, speed, generation and number of children alongside the instance of its parent SlimeInfo, this is used to create a sort of linked-list of slimeInfos which can be used to keep track of the ancestry of slimes. Here is a code snippet 5.4, which contains the constructor for SlimeInfo when a slime has a parent.

```
1         // Constructor for slime with parent, takes the
              generation and parentSlime as additional paramters
2     public SlimeInfo(string slimeName, float slimeSize, float
          slimeSpeed, int slimeGeneration, SlimeInfo
         parentSlime)
3     {
4         SlimeName = slimeName;
5         SlimeScale = slimeSize;
6         SlimeSpeed = slimeSpeed;
7         SlimeGeneration = slimeGeneration;
8         // The SlimeInfo parentSlime is a way to set the
              connect the parentSlimeInfo to this slimeInfo
              approximating a linked list
9         ParentSlime = parentSlime;
10        SlimeChildren = 0;
11    }
```

Listing 5.4: Slime Info with Parent

The slimes behaviours are controlled by a neural network which is a mutated copy of their parent's neural network, though the mechanics of this will be analyzed more in-depth in the Neural Network section.

5.3.3 Neural Network

The Neural Network implemented in SlimeWorld is a simple neural network which does not make any use of back-propagation or other more advanced training techniques, the reason for this is that in order to more accurately approximate evolution, the neural network has no scoring parameters for evaluating its performance, and with the sandbox nature of the simulation it would be rather awkward to try and set any parameters by which to measure the success of the neural network.

To this end, the neural network is one of the simplest variants, with the modifications being an unmodified random chance, which is applied equally to every weight and bias in the network. The downsides of this approach are obvious in that the neural network will neither be trained quickly nor to a high degree of accuracy.

Inputs and Outputs

For the neural network, one decision I made early on was that I wanted to ensure that every input was normalized to be within the same range, this was a design decision that I could implement without too much difficulty as all inputs would

be generated by the slimes in-game. As I had settled on tanh as my activation function, I decided to keep all inputs and outputs of the neural network between -1 and 1.

The topology of the neural network that I have used for this project is 12 nodes in the inputs layer, 8 nodes in the first hidden layer, 5 nodes in the second hidden layer, and 2 output nodes. I have classified the input nodes into three sections based on what kind of information they contribute to the neural network, the first being internal information 5.5, being information that pertains to the slime itself and contains 3 input nodes.

```
1    // Represents slimes current hunger, updates every in-
        game update
2    _inputsToNeural[0] = _saturation / 50f - 1f;
3
4    // Setting neural inputs that persist for the life of the
         slime, as neither _scale nor _speed can change, these
         inputs are set once and never changed
5    _inputsToNeural[1] = _scale;
6    _inputsToNeural[2] = _speed;
```

Listing 5.5: Neural Networks inputs for The Slime

The next category of information for the slime is that related to food which contains 3 input nodes 5.6. and the final category of information being that related to other slimes which contain 6 input nodes 5.7.

```
1     // Represents saturation of food, -0.5f due to following
          formula
2    // value = (saturation / 50) - 1, so for food which has
        default 25 = -0.5f
3    _inputsToNeural[3] = -0.5f;
4    // Represents angle of food relative to slime in pi
        radians to fit in with scheme of inputs being -1 to 1,
5    // Negative values are on the left side, positive values
        are on the right
6    _inputsToNeural[4] = (Mathf.Deg2Rad * Vector2.Angle(
        transform.forward, (Vector2)(_closestFood.transform.
        position - transform.position))) - 1;
7    // Represents distance to food from slime, with values
        between -1 and 1,
8    // And distance greater than 25 is capped at an input
        value of 1
9    _inputsToNeural[5] = Mathf.Clamp((Vector3.Distance(
        _closestFood.transform.position, transform.position) /
         12.5f) - 1f, -1f, 1f);
```

Listing 5.6: Neural Networks inputs for Food

```
    Slime closestSlimeScript =
    _closestSlime.GetComponent<Slime>();
    // Represents scale of closestSlime, values are clamped
       between 0.5 and 2 typically
    // So take away one to fit within established scheme for
       neural net inputs
    _inputsToNeural[6] = closestSlimeScript.GetScale() - 1;
    // Represents speed of closestSlime, values are clamped
       between 0.5 and 2 typically
    // So take away one to fit within established scheme for
       neural net inputs
    _inputsToNeural[7] = closestSlimeScript.GetSpeed() - 1;
    // Represents angle of closestSlime relative to slime in
       pi radians to fit in with scheme of inputs being -1 to
        1,
    // Negative values are on the left side, positive values
       are on the right
    _inputsToNeural[8] = (Mathf.Deg2Rad * Vector2.Angle(
       transform.forward, (Vector2)(_closestSlime.transform.
       position - transform.position))) - 1;
    // Represents distance to closestSlime from slime, with
       values between -1 and 1,
    // And distance greater than 25 is capped at an input
       value of 1
    _inputsToNeural[9] = Mathf.Clamp((Vector3.Distance(
       _closestFood.transform.position, transform.position) /
        12.5f) - 1f, -1f, 1f);
    // Represents saturation value of closestSlime,
       calculated with following formula,
    // input = (saturationIfEaten / 50) = 1
    _inputsToNeural[10] = (closestSlimeScript.
       GetSaturationIfEaten() / 50f) - 1;
    // Represents status of closestSlime, 0 if dead, 1 if
       alive
    _inputsToNeural[11] = closestSlimeScript.IsAlive() ? 1 :
       0;
```

Listing 5.7: Neural Networks inputs for other slime

The training of the neural network is performed with a very naive and inefficient method, where when a NetworkNetwork is cloned from another, each bias and

weight has a chance of changing a given amount based on the input parameters, which are set in the settings menu mentioned earlier.

5.3.4 Food Manager

The FoodManager script is a singleton responsible for the creation and removal of food in the simulation. The FoodManager has an ObjectPool instance containing a prefab for a food GameObject. Upon starting the game and every 10 seconds thereafter a specified amount of food, determined by the foodPerInterval setting is spawned unless the foodCap is reached at which point no more food will spawn until the food dips below the cap. When food is spawned, a food GameObject is pulled from the objectPool and placed at a random location on the map. And when a piece of food is eaten, it calls the FoodManager's DeactivateFood method which in turn calls the ObjectPool's DeactivateObject method.

5.3.5 Object Pool

I created my own implementation of an object pool as I felt that the object pool incorporated in Unity didn't properly satisfy my requirements. One of the key reasons was that I desired the ability to keep track of all active objects in the pool so that I could more easily perform checks to determine which entities were closest to one another by iterating through every entity on the list of active objects.

The object pool is composed of list of GameObjects containing the active objects in game and a queue of GameObjects representing all the deactivated objects in the game. I picked a list to represent the active objects as I wished to be able to iterate through the list easily, while I felt that a queue would suit the inactive objects better, as all that was necessary was to store objects that were deactivated and then pop them from the front of the queue once a new object was needed.

To make use of the object pool, I created a constructor which would take in the gameObject representing the prefab to be stored as a parameter as can be seen in this listing 5.8. I then made use of the object pools to store prefabs of both the foods and the slimes as both are frequently created and destroyed in the simulation.

```
1    public ObjectPool(GameObject pooledObject)
2    {
3        _activeObjectPool = new List<GameObject>();
4        _inactiveObjectPool = new Queue<GameObject>();
5        _pooledObject = pooledObject;
6    }
```

Listing 5.8: Object pool

After making the object pool, I had to make methods to use the object pool to provide objects when necessary which can be seen in this code snippet 5.9. The purpose of the DeactivateObject method was to take in objects that were to be deactivated, set them to inactive, remove them from the list of active objects and then add them to the queue of inactive objects. The GetPooledObject method would check if the queue of inactive objects was empty and if so it would instantiate a new object based on the stored prefab, otherwise it would pop the deactivated object at the head of the queue and activate it, before returning it.

```
1  // Passed in object is deactivated, removed from the
       activepool and added to inactivepool
2      public void DeactivateObject(GameObject
           objectToDeactivate)
3      {
4          objectToDeactivate.SetActive(false);
5          _activeObjectPool.Remove(objectToDeactivate);
6          _inactiveObjectPool.Enqueue(objectToDeactivate);
7      }
8
9      // Either returns an object from the inactiveObjectPool
           or instantiates a new object if the inactiveObjectPool
           is empty
10     public GameObject GetPooledObject()
11     {
12         // Object variable is declared
13         GameObject objectToReturn;
14         if (_inactiveObjectPool.Count != 0)
15         {
16             // if inactivePool isn't empty then take object
                   from queue
17             objectToReturn = _inactiveObjectPool.Dequeue();
18             objectToReturn.SetActive(true);
19         }
20         else
21         {
22             // if pool is empty then instatiate new object
23             objectToReturn = Object.Instantiate(_pooledObject
                   );
24         }
25         // object is added to activeObjectPool
26         _activeObjectPool.Add(objectToReturn);
27         return objectToReturn;
28     }
```

Listing 5.9: Object pool activate and deactivate

5.4 Spectator

The spectator represents the user's camera and is a flying object which is controlled using WASD, E and Q for movement forwards, left, back and right and up and down. The Spectator uses the mouse for rotation, though if the escape is pressed, the cursor is unlocked so that it can interface with the UI. If the left mouse button is clicked while the user is moused over a slime, the statistics relevant to the slime and their ancestry is displayed 5.5.

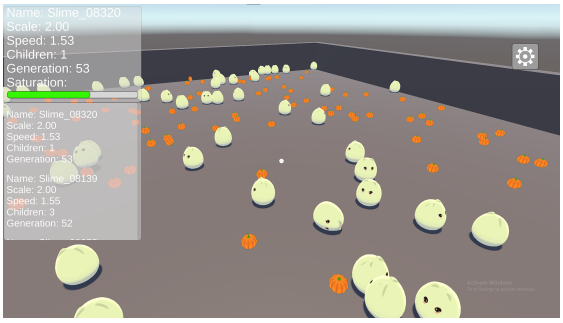

Figure 5.5: In-Game UI

The ancestry data, being populated using the slime's slimeInfo, and the attached reference to the previous generation's slime info, to iterate through them all and generate a list. Which is then assigned to a scroll-able list on the left side of the screen.

if the user's mouse is unlocked and they click on the gear on the right side of the screen they open the in-game menu from which they have buttons to return to the main menu, quit the game outright or close the in-game menu 5.6.

Figure 5.6: In-Game Menu

Chapter 6

System Evaluation

6.1 Creating a 2d Plane for Simulated Life

This was the easiest of the goals that I set out to do with my project and though it is an environment free of flair or any complexity I do feel that the project has succeeded in creating a simple environment which contains simulated life.

6.2 Populating the Plane with creatures

I feel that though life in the context of the simulation is rather simple, I do think that it satisfies my initial goals as life is capable of evolution albeit in a simple manner and only governing a few traits, and I am happy with the fact the creatures can both eat, starve and be eaten. I am also happy with the manner in which the creatures can reproduce and the fact that many of a parent's traits are successfully passed down to their offspring which small changes occurring as required to manifest evolution.

6.3 To include a System for Creatures to Evolve

I feel that this is an area where I had fallen short of my desired goals as I had originally wished to implement a greater variety of inheritable traits that could affect the creature's lifestyles and had hoped to do this through a more complicated system based on specific genes that might be passed down, though I thought this, in the end, unnecessary as it was excessive to model in a simulation with asexual rather than sexual reproduction.

6.4 To have Creatures controlled by a Neural Network

I feel that this was the greatest failure of the project as I feel that my approach of using a small feed-forward neural network didn't provide enough potential for the creatures to learn to display intelligent behaviour in regards to trying to stay alive and that the lack of a specialized training method due to the desire to avoid any inherent motivations in the system meant that the neural network would either train far too slowly or always overshoot the "optimal" weights and biases for its purposes.

6.5 To support multiple "species" of Creatures, suited to different niches

I feel that my project failed in reaching this goal due to shortcomings of the neural network and the shallowness of the traits I implemented which meant that there wasn't enough depth in the simulation for any sort of ecological niches to form, and to be filled, furthermore, I couldn't develop an algorithm which would be able to define species in the context of the simulation.

6.6 To create a Tweakable Environment

Though I pursued it in a simple manner I am happy with the fact that many of the aspects of the simulation can be modified by changing the settings of the simulation, leading me to believe that the environment is at least somewhat tweakable.

Chapter 7

Conclusion

7.1 Overview

This section is where I'll summarize the goals of my project and how I feel my project performed as regards meeting them, I will also discuss what I think was the biggest flaw in my project as regards approaching these goals.

7.2 Goals

- To create a 2d plane for simulated life to live on.

  I feel that this goal was met satisfactorily.

- To populate the plane with creatures which can eat, starve and procreate.

  I feel that this goal was met satisfactorily.

- To include a system for the creatures to evolve throughout generations, passing traits along to their children.

  I believe that this goal was achieved though not with the same level of depth as I'd desired.

- To have the creatures controlled by neural networks which themselves are also honed through this evolutionary process.

  I believe the neural networks are the largest failure of the project so I will detail why further on in this chapter.

- To support multiple "species" of creatures, suited to different evolutionary niches

I believe that I have failed to achieve this goal as the environment created by the application was insufficiently complicated to provide ecological niches for slimes to shape themselves to and the slimes themselves were insufficiently intelligent.

- To create a tweakable environment which can affect the evolution of the creatures.

  Though the end implementation of tweakable settings for governing the environment was done rather simply, I do feel that the settings as they are, work reasonably well within the system and I considered the goal satisfied.

7.3 Faults with the Neural Network

7.3.1 Inefficient Training Methodology

A big aspect of the inefficiency was the intentionally simplistic model in which the neural networks were trained, as utilizing more efficient methods like back-propagation in the training of the neural networks would have detracted from the goal of modelling evolution, by necessitating specific scorable criteria for the slimes which would need to be used to evaluate their relative fitness and hence the degree of change which should be made in the neural network.

As while the training method implemented in the system is currently a very naive training method, which doesn't take anything into account regarding the performance of the neural network, I couldn't come up with a better method for training neural networks which didn't include any scorable or defined goals which would be used to measure it's performance.

7.3.2 Rigid nature of Neural Network

However, what I think was actually the larger shortcoming of the neural network was that despite the free-form nature of neural networks in that the ways they process data are undeniable but the main issue was actually with the rigid nature of the inputs. The inputs of the neural network govern its potential alongside a greatly increased training period with a larger amount of input nodes.

The manner in which this issue has manifested in the project has actually been in regard to how the slimes track other entities, for as is seen in this code snippet 5.7, half of the neural network's inputs are dedicated to observing the information of just one other slime. And the most crucial downfall of this, is that no matter how refined the neural network would become, through potentially millions of cycles of

training, it could never properly learn to tackle situations with multiple competing slimes as the slime can only ever perceive one other slime at a time.

This is a fundamental issue with the neural network as I see no manner in which to attempt to solve the issue while keeping any aspects of the neural network as it is currently outside of retaining the 2 output nodes. Because should the developer decide to tackle this issue by providing additional sets of input nodes to handle more slimes, it will risk drowning out the other inputs of the neural network, and the number of slimes that could be tracked would still be rigidly defined by the size of the inputs.

In Summary, the issue regarding the rigid nature of the neural network is that at least in the implementation that I have gone with, there is no way in which the input systems can be modified to work with a dynamic number of entities. Hence if I were to tackle this project again, the main differences I would make would be in regards to the topology of the neural network and performing far more research into the different forms of neural network or machine learning systems which can be made to more easily work alongside systems such as arrays or other lists of inputs.

Bibliography

[1] Chaitya Vohera, Heet Chheda, Dhruveel Chouhan, Ayush Desai, and Vijal Jain. Game engine architecture and comparative study of different game engines. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–6, 2021.

[2] Erkam Guresen and Gulgun Kayakutlu. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3:426–433, 2011. World Conference on Information Technology.

[3] Herbert Spencer. *The Principles of biology v. 1, 1866*, volume 1. D. Appleton & Company, 1866.

[4] Justin Helps Primer. Simulating natural selection. Available at .

[5] n/a. Intellisense in visual studio. Available at `https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2022`, last accessed on the 2023/04/21.

[6] Jead. What platforms are supported by unity? Available at `https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity-`, last accessed on the 2023/04/17.

[7] n/a. Unity documentation. Available at `https://docs.unity.com/`, last accessed on the 2023/04/21.

[8] Jeff Farris. Forging new paths for filmmakers on "the mandalorian". Available at `https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian`, last accessed on the 2023/04/21.

[9] n/a. Frequently asked questions. Available at `https://www.unrealengine.com/en-US/faq`, last accessed on the 2023/04/21.

[10] n/a. Unreal engine 5.1 documentation. Available at `https://docs.unrealengine.com/5.1/en-US/`, last accessed on the 2023/04/21.

[11] n/a. Unreal engine api reference. Available at `https://docs.unrealengine.com/5.1/en-US/API/`, last accessed on the 2023/04/21.

[12] n/a. 2022 developer survey. Available at `https://survey.stackoverflow.co/2022/#developer-profile`, last accessed on the 2023/04/21.

[13] Wojciech Zaremba and Greg Brockman. Openai codex. Available at `https://openai.com/blog/openai-codex`, last accessed on the 2023/04/21.