Jihočeská univerzita v Českých Budějovicích

Ekonomická fakulta

Katedra aplikované matematiky a informatiky

Bakalářská práce

Simulace a vizualizace nabídky a poptávky

Vypracoval: Nguyen Nam Ha

Vedoucí práce: Mgr. Radim Remeš, Ph.D.

České Budějovice

2024

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Nguyen Nam HA**
Osobní číslo:	**E21130**
Studijní program:	**B0688A140010 Podniková informatika**
Téma práce:	**Simulace a vizualizace nabídky a poptávky**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Cílem práce je vytvořit desktopovou aplikaci simulující model ekonomického systému vývoje ceny na základě nabídky a poptávky. Aplikace bude realizována pomocí herního enginu Unreal Engine 5, simulace bude vizualizována ve 3D prostředí.

Metodický postup:

1. Studium odborné literatury.
2. Popis terminologie, postupů a nástrojů.
3. Návrh a popis vývoje a implementace výsledné aplikace.
4. Zhodnocení použitelnosti aplikace pro nasazení v reálném prostředí.
5. Závěry a doporučení.

Rozsah pracovní zprávy:	**40 – 50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná**
Jazyk zpracování:	**angličtina**

Seznam doporučené literatury:

1. Gregoire, M. (2021). *Professional C++.* 5th Edition. Wrox.
2. Gregory, J. (2018). *Game Engine Architecture.* 3rd Edition. A K Peters/CRC Press.
3. Marques, G., Sherry, D., Pereira, D., Fozi H. (2022). *Elevating Game Experiences with Unreal Engine 5.* 2nd Edition. Packt.
4. Venter, H., Ogterop, W. (2022). *Unreal Engine 5 Character Creation, Animation, and Cinematics.* Packt.
5. Pindyck, R. S. & Rubinfeld, D. L. (2018). *Microeconomics.* 9th Edition. Pearson.
6. Unreal Engine 5.1 *Documentation: Complete resources for learning to use Unreal Engine 5.* (2023). Dostupné z: <https://docs.unrealengine.com/5.1/en-US/>

Vedoucí bakalářské práce:	**Mgr. Radim Remeš, Ph.D.**
	Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: **20. ledna 2023**
Termín odevzdání bakalářské práce: **12. dubna 2024**

[illegible]

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13 (26)
370 05 České Budějovice

[illegible]

doc. RNDr. Zuzana Dvořáková Líšková, Ph.D.
děkanka

doc. RNDr. Jana Klicnarová, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 13. března 2023

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47 zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz, provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, 31. července 2024 Podpis: Nguyen Nam Ha

Poděkování

Velice rád bych poděkoval vedoucímu své práce panu Mgr. Radim Remeš, Ph.D. za jeho ochotu, trpělivost a vstřícné rady, které mi poskytl během vypracování bakalářské práce. Rád bych taky poděkoval rodičům za neustálou podporu během studia a také bych poděkoval Bc. Dagmar Náprstkové za konzultaci.

Content

1. Introduction

As the topic of my bachelor's thesis, I chose the simulation and visualization of supply and demand using the game engine Unreal Engine, also known by the abbreviation UE. It is a video development tool from Epic Games. I was very interested in this tool for creating a 3D world, and therefore I wanted to deepen my knowledge and gain experience in this environment. My fascinations with video games also influenced this decision. The popularity of Unreal Engine is increasing recently because it is free to use, and new updates are released frequently.

The aim was also to help colleagues in the first semester at the Faculty of Economics face the subject of microeconomics. To show that the theory of supply and demand is not something scary, but on the contrary, that it is very interesting and important to understand how the free market works. I aim to achieve this with a clear, simple visualization, but not overly complex.

The main goal of the bachelor's thesis is to create a functional executable application where the user can see in real time the economic course of supply and demand. The application will allow the option to select the number of buyers, the number of sellers to start the simulation, restart the simulation and exit the simulation. The application will be in English.

The bachelor thesis will be divided into a theoretical part and a practical part. The theoretical part will be devoted to the general issue of supply and demand, about the history of video games and their use in the real world, game engines to create a game and a general overview of starting a project from scratch inside unreal engine.

The practical part will be focused on the development of the application itself. Its problems and methods that have been added to solve. Documentation of its development. Discussion regarding the process, criticising the final game, the potential future development and real-life usage.

2. Microeconomics

Microeconomics deals with constraints. For example, consumers using their limited income to buy goods and services. Technology organizations using their limited budget for production, and the limited hours workers have for their labour or leisure. It also explores optimizing these limits, especially the allocation of scarce resources (Pindyck & Rubinfeld, 2018).

How consumers can better spend their incomes or how workers can best divide their free time between different activities or even how firms can best allocate their financial resources between hiring workers and purchasing machinery, or between producing different products (Pindyck & Rubinfeld, 2018).

2.1 Importance

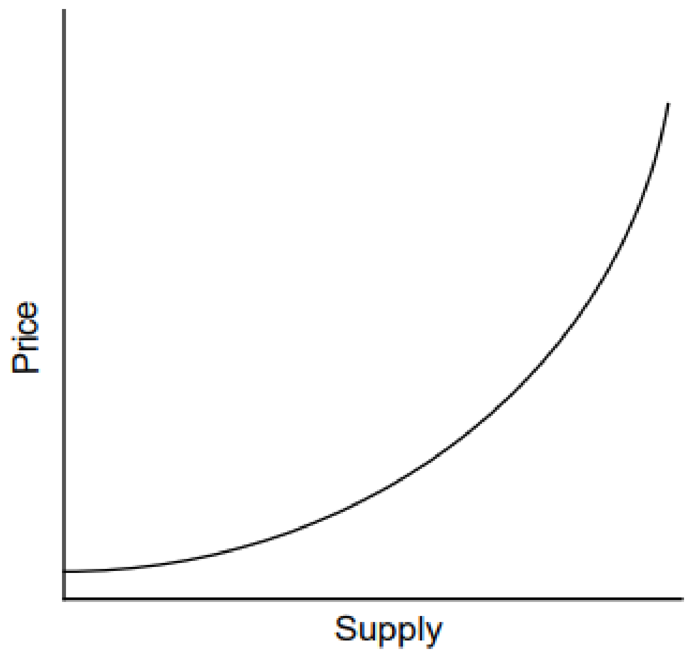
Having effective economic policies can reduce the causes of international friction and promote a peaceful world. Economic policies can promote global trade and prosperity, which further strengthens peace. Prosperity supports the growth of democracy and freedom, therefore with understanding economic principles, we can secure a better world (Clayton, 1946).

2.2 Supply

Supply represents the readiness and capability of sellers to offer goods. Higher prices usually lead to more goods being available on the market because suppliers can still make a profit despite increased production costs. In a real market, when the stock of goods that is available for sale is below the desired level, manufacturers will raise both supply and prices. Increases in supply raises production costs, further increasing prices and production rates (Whelan, Msefer, & Chung, 2001).

On the other hand, if stocks of goods are too high, the opposite happens. Classical economic theory shows this with the supply curve, as seen in figure 1. The curve goes upwards showing that with increased supply the price goes higher. The goods become more expensive (Whelan, Msefer, & Chung, 2001).

Figure 1: Supply Curve



Source: (Whelan, Msefer, & Chung, 2001).

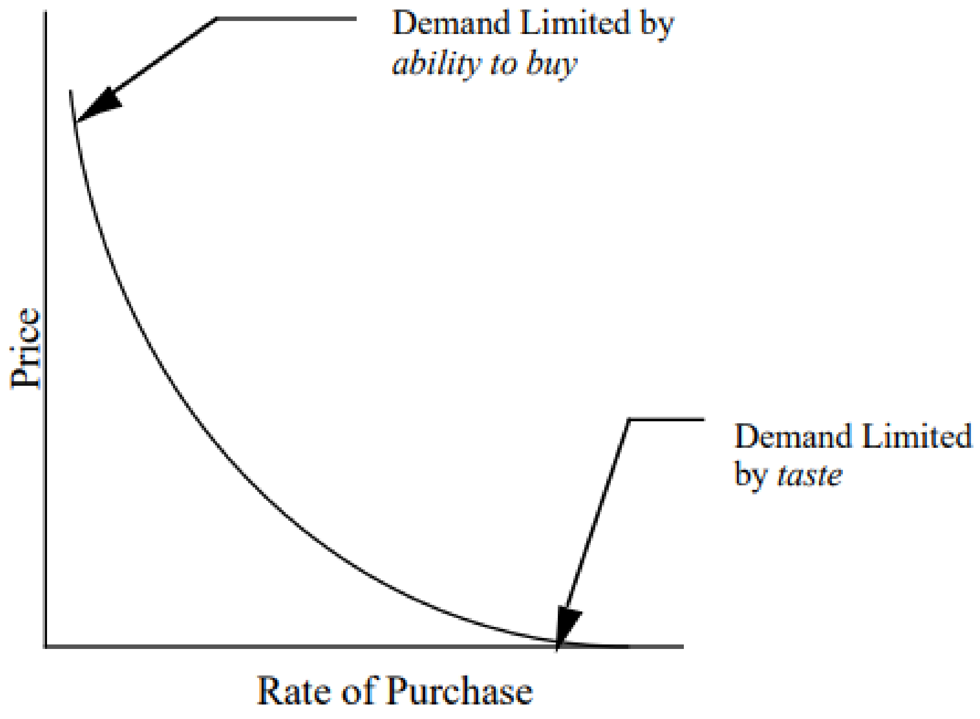
2.3 Demand

Demand, as a fundamental concept in economics, represents consumer's desire to purchase a product, determined by their taste and ability to afford it. Taste reflects the willingness to buy at a given price, while ability to buy hinges on having adequate wealth or income. These factors are influenced by market prices, where higher prices typically result in lower demand and lower prices typically result in higher demand (Whelan, Msefer, & Chung, 2001).

Very low prices may attract more consumers but there is a limit to how much of a product people desire. It can lead to diminishing satisfaction with additional purchases. Therefore, even at low prices, demand is constrained by taste and isn't unlimited, as individuals have finite resources. Conversely, high prices restrict demand due to affordability constraints, regardless of consumer's desire for the product (Whelan, Msefer, & Chung, 2001).

The demand curve, seen in figure 2 shows how much the quantity of a goods that consumers demand varies with its price, typically sloping downward because consumers buy more as prices fall. Demand also depends on other factors like income, weather, and prices of other goods. Generally, as income rises, the quantity demanded increases (Pindyck & Rubinfeld, 2018).

Figure 2: Demand Curve



Source: (Whelan, Msefer, & Chung, 2001).

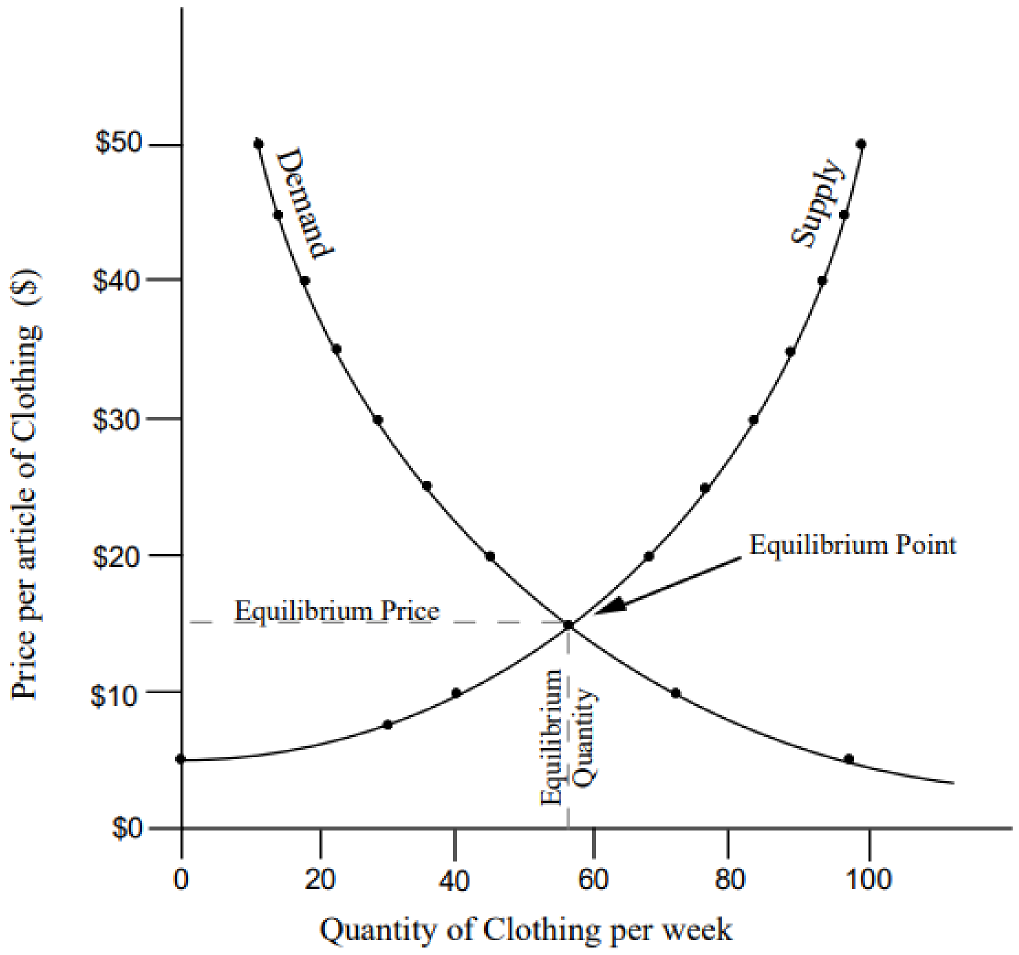
2.4 Market equilibrium

Demand is how much of a good or service people want to buy at different prices, while supply is how much of that good or service sellers are willing to sell at those prices. The relationship between demand and supply controls how the market works (Whelan, Msefer, & Chung, 2001).

Buyers and sellers react differently to price changes. When prices go up, sellers want to sell more, but buyers want to buy less. When prices go down, buyers want to buy more, but sellers want to sell less (Whelan, Msefer, & Chung, 2001).

Equilibrium happens when two lines, the supply curve, and the demand curve, meet each other at a certain price and certain amount, shown in the figure 3. This is where the amount that's offered for sale and the amount people want to buy are the same. In a free market, prices usually change until this balance happens. Once it does, there is no need for the price to keep changing because neither too much is being sold nor too little (Pindyck & Rubinfeld, 2018).

Figure 3: Supply and Demand Curves



Source: (Whelan, Msefer, & Chung, 2001).

3. Video games

A video game is a system with rules where players try to achieve different results. Each result has a different value, and players put in effort to reach a desired outcome. They care about the outcome, and the results can change based on how the game is played. This definition includes many types of games, like The Sims, where players set their own goals and are interested in the outcomes they create (Arias, 2014).

3.1 Evolution of Video games

The video games back in the day, were vastly different to how they are nowadays. The very first Noughts and Crosses game also called Tic-Tac-Toe was created by a British professor A.S. Douglas in 1952 as part of his dissertation (History.com Editors, 2022).

Back in 1967, Ralph Baer and his team made the Brown Box, a video game system for televisions. Sold to Magnavox in 1972, it became the first home video game console, the Odyssey. Pong, one of its games, inspired Atari's arcade success. Legal issues followed, resolved with Atari becoming a licensee. In 1977, the Atari 2600 was released, starting home gaming's popularity. Games like Space Invaders, Pac-Man, and Donkey Kong added to the fun. Microsoft's Flight Simulator also joined in. It was an exciting time for gamers (History.com Editors, 2022).

The North American video game industry crashed due to too many consoles and competition from computer games, but in 1985, Nintendo's NES revived it with quality games like Mega Man and Castlevania. Then, in 1989, Nintendo launched the Game Boy, starting a successful era of handheld gaming with hits like Tetris. (History.com Editors, 2022).

In the mid-1990s, gaming entered a new era with the rise of 3D technology. Sega introduced the Saturn, followed by Sony's PlayStation and Nintendo's Nintendo 64. Sony's strong support from third-party developers made the PlayStation a market leader, leading to the massive success of the PlayStation 2. Sega's Dreamcast, though innovative, couldn't compete and marked the end of Sega's console business (History.com Editors, 2022).

The modern era of gaming began in the mid-2000s with the release of Xbox 360, PlayStation 3, and Wii. Each brought high-definition gaming, with PlayStation 3 standing out for its Blu-ray capability. Despite Sony's competition, Xbox 360 excelled in online gaming and introduced Kinect motion control. Wii outsold both with its motion controllers. Gaming expanded to social media and phones, with hits like Angry Birds. In 2011, Skylanders: Spyro's

Adventure merged toys with gaming. Wii U started the next generation but failed. Nintendo rebounded in 2017 with the Nintendo Switch. Sony and Microsoft released new consoles in 2020 (History.com Editors, 2022).

3.2 Benefits of Video games

There is more to video games than what is often portrayed in popular press, like where they laser focus on the alleged negatives, such as addiction, becoming more prone to violence and various degrees of health deterioration. Video games can become a central tool to help students and children build and improve their existing skills. Games, where a person can play with an object can show them how certain things can look from a different point of view, further helping them understand new concepts. Their perspective largens in the three-dimensional spatial games (Griffiths, 2002).

Video games can provide to children not only education value but also entertainment value. By engaging with then using interactive elements inside a game, to stimulate their learning experience. A wide range of emotions play a role in the learning such as excitement, curiosity, and novelty (Griffiths, 2002).

Skills that many have acquired or improved during their video game sessions are basic math operations where they must interact with numbers. Reading and language skills, if they wish to play the game, change the settings, or simply make progress, they must interact with words on the screen. Video games can be played online with others, and thus it’s inevitable that it will lead to human interactions, which further promotes social skills (Griffiths, 2002).

Arias (2014) noted that video games in education promoted critical thinking skills of students. Video games can greatly improve critical thinking by offering learning experiences that extend beyond the traditional classroom. Three dimensional virtual environments like Second Life engage students in activities that not only meet but often exceed learning objectives, by encouraging active participation in the learning process. These virtual spaces also allow students to interact with a wider community, giving their work more exposure and increasing their enthusiasm for learning (Arias, 2014).

In one high school history class, the video game Making History was used to shift the classroom from a lecture-based format to a learner-focused environment. Students managed countries during World War II, making strategic decisions and critically analysing historical events. This interactive method sparked excitement and discussions among students, even outside the classroom (Arias, 2014).

Similarly, the use of Civilization III in an afterschool program showed that video games could make learning more engaging. Students, who were usually disengaged and had average grades, started understanding and using academic terms related to ancient cultures and sought out additional information on their own in their free time. These cases illustrate that video games can enhance critical thinking by encouraging students to explore complex concepts and make informed decisions in a dynamic and interactive setting (Arias, 2014).

Alzubi et al. (2018) conducted a study on young children to find out whether interactive games can have an impact on children between the age of five and six. The outcome showed a positive change in children who used the technology improving their memory and mathematical skills. Children did better than those who haven't played the interactive game. Thus, concluding that it can be a beneficial way to help kids to learn.

3.3 Different types of Video games

In the world of video games, there are various genres, each with distinct characteristics and styles. Here are some examples of the popular ones currently in the market:

Role-Playing Games or known under their abbreviation (RPGs) send players on a journey in a fantasy world assuming a main character, often involved in detailed storylines known as quests. Popular examples include "The Elder Scrolls IV: Oblivion", "The Elder Scrolls V: Skyrim" and "Minecraft" (Värtinen, Hämäläinen, & Guckelsberger, 2024).

Adventure games offer a different flavour of immersive gameplay, focusing on exploration and puzzle-solving over character progression and combat. Players in adventure games often stepped into multiple connected locations, trying to solve the current objective to progress into the next location. Titles like "Deja Vu: A Nightmare Comes True (1985)", "King's Quest III: To Heir Is Human (1986)" and "Mean Streets (1989)" (Wolf, 2011).

Horror games transport players into scary situations where fear is the main entertainment factor. These games sole focus is on suspense, tension, and psychological horror to evoke intense emotional responses from players. The main enemy or antagonist tends to be monsters in forms like zombies, ghosts or other fictional, fantasy characters from popular media. Titles that would fit into this genre are "Requiem: Memento Mori" and "DayZ" (Geraci, Recine, & Fox, 2016).

3.4 Simulation games

Simulation is a very broad statement that can hold a wide range of interpretations ranging from different models with contrasting levels of structure and complexity. Basically, any model could be seen as a simulation, which makes it very difficult to create a characterization or categorization (Solomon, 1980).

Dorn (1989) discussed that the definition of simulation games is one the hardest things to do. Many tried before him, and no one could reach a conclusion that would satisfy everyone. Some consider simulation games to be all machine types of simulations, others consider roleplaying or any decision-making games to be. The terms “Simulation game”, “teaching games”, “gaming simulations” et cetera is used mutually. Definitions vary widely from person to person.

4. Game engines

A game engine is a software framework designed to make the development of video games easier. It includes a set of tools, libraries, and components that take care of the essentials of gameplay, such as graphics rendering, scripting system, physics simulation, user input management, audio system and networking system. Choosing the right game engine is a key decision for the success of the entire project. This is true for both two- or three-dimensional games where the engine plays a significant role in the overall optimization and functionality of the game. To make the right decision, several factors need to be considered. The importance and detail can vary greatly depending on the requirements of the final product (Lahtinen, 2016).

Figure 4 shows popular game engines used by developers to create games. Amongst them is Unity, Unreal Engine, Godot, CryEngine and Gamemaker.

Figure 4: Popular game engines.

Source: (VANAS, 2023)

4.1 Unity

Unity game engine developed by Unity Technologies holds a dominant position in the game development landscape, as evidenced by its prevalence in various sectors. Unity's appeal is not only attributed to its widespread usage but also its accessibility and affordability, particularly for smaller-scale projects (Dealessandri & Calvin, 2023a).

According to Unity in 2022, 70% of the top 1000 mobile games and over 60% of all augmented and virtual reality content were powered by Unity. Additionally, the company claims that half of all games across different devices utilize its technology, highlighting its widespread adoption. Unity’s reach extends globally, with users in 195 countries, showcasing its broad appeal and accessibility on a global scale (Dealessandri & Calvin, 2023a).

The popularity of Unity is further underscored by its user base and project activity. As of March 2023, Unity Editor boasted around 1.3 million monthly active users, indicating a thriving community of developers leveraging the platform for game development. Moreover, an average of 3,300 new projects were initiated daily in 2022, reflecting the engine’s continuous growth and relevance in the industry. Apps utilizing Unity were downloaded over four billion times each month in 2022, showcasing the extensive reach of games and applications developed with the engine (Dealessandri & Calvin, 2023a).

The engine offers a Personal license that allows individuals to create commercial games free, provided their revenue or funds raised do not exceed a certain threshold. This approach makes it available for game developers, inspiring developers to create and publish games without financial barriers. With support for over 17 platforms, including iOS, PS5, Microsoft HoloLens, and Android TV, Unity provides a versatile and inclusive environment for developers to bring their creative visions to life (Dealessandri & Calvin, 2023a).

Unity Visual Scripting is a valuable resource, offering a user-friendly alternative to traditional coding for game development. This official scripting tool empowers developers and graphic designers to craft game mechanics and character interactions without the need for proficiency in Unity’s C# programming language. Through the linking of logic nodes, Unity Visual Scripting significantly lowers the entry barrier for individuals without programming experience, facilitating their participation in game development projects (Unity Technologies, 2024)

4.2 Godot

Godot, a game engine created by Argentinian developers Linietsky and Manzur, has seen a surge in popularity, particularly among indie developers, overtaking larger engines like GameMaker and Unreal Engine in recent years. Its appeal extends beyond indies, with companies like Sega and Tesla adopting it for projects (Dealessandri & Calvin, 2024).

The engine, named after Beckett’s play, “Waiting for Godot” is free and open source, relying on donations. Godot boasts versatility, excelling in 2D games, with a node-based interface and intuitive GDScript language, inspired by Python. Its open-source nature allows for flexibility, which helps developers to modify the engine to suit their needs. The release of

Godot 4 with improved 3D support further fuelled its growth, alongside changes in competitor policies, driving more developers to explore its capabilities (Dealessandri & Calvin, 2024).

Godot being open source means the community is safe from big changes, that can happen with other engines like Unity. Even if something strange happened and the main team tried to change things, the community could keep using the engine. No one owns it all, which is a big advantage of open-source software (Dealessandri & Calvin, 2024).

4.3 Unreal Engine

Unreal Engine is widely known for its photorealistic graphics and is highly favoured in the gaming sector for these reasons. The introduction of Unreal Engine 5 has brought forth advanced features such as Lumen dynamic lighting and Nanite for intricate object details. Beyond gaming, the engine finds utility in fields like architecture, product design, and filmmaking. Epic Games' evolving approach to pricing has made Unreal Engine more accessible to developers of varying scales. With its extensive platform compatibility and adaptable nature, Unreal Engine boasts a significant user base (Dealessandri & Calvin, 2023b).

Unreal Engine has different options for who wants to use it. If you earn less than $1 million USD a year, you can use it for free. This includes individuals, small businesses, and schools. You get access to all features and support from the community (Epic Games, 2024a).

If you make more than $1 million USD, you pay a 5 percentage of your earnings as a fee, but if you sell your game on the Epic Games Store, you don't have to pay anything. For big companies using Unreal Engine but not selling games, there's a different fee. It's CZK47,457.41 per person each year. This gives access to Unreal Engine and other software's like Twinmotion and RealityCapture (Epic Games, 2024a).

4.4 Usage of game engines

The rendering system plays a vital role in game engines, transforming calculated graphics into visual elements on the screen. It's essential for bringing game world objects to life in a visible form. This creates actual imagines on the screen (Lahtinen, 2016).

Scripting systems enable developers to define game logic and add intelligence to game elements through programming. Bringing ideas to life. Different engines offer various programming languages and tools, along with pre-made scripts that developers can modify for their applications (Lahtinen, 2016).

Input systems manage user interactions from devices like keyboards, microphones, or touch screens. Instead of hard-coding inputs, game logic can be tied to abstract input names (Lahtinen, 2016).

Physics engines model the physical behaviour of objects by applying principles like gravity and friction. This requires significant computational power and should be applied only where necessary (Lahtinen, 2016).

Audio systems handle sound effects and music, adjusting qualities based on the listener’s position and environmental factors. Effects can be added to enhance the audio experience (Lahtinen, 2016).

A GUI system helps make and manage user interface parts like buttons and text boxes. Developers may implement these functions themselves or use external third-party libraries (Lahtinen, 2016).

Networking systems assist network communication in multiplayer games, providing tools for handling multiplayer interactions effectively (Lahtinen, 2016).

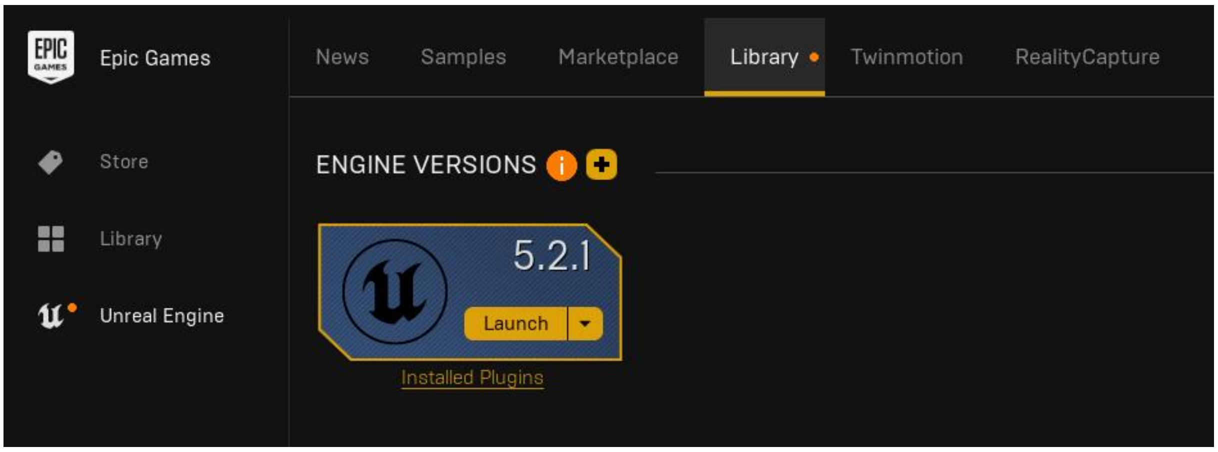
4.5 Usage of game engines outside of video games

Game engines are not only used for creating video games for the sake of entertainment, but they are featured in other fields too. Serious games are developed and used in fields like the military, healthcare, education, and training. Examples include a firefighter training environment using Delta3D, a virtual evacuation training game with NeoAxis, and a VR environment for emergency simulations with Unreal Engine. There is also a spacewalking simulation made with Unity and NASA models for education, and rehabilitation simulations for stroke patients using Unity and Oculus Rift. Additionally, there are driving simulators made with Unity for training ambulance drivers and improving general driving skills (Lahtinen, 2016).

5. Unreal Engine 5

To first begin a project, it's needed to download The Epic Games launcher from the official epic game's website. To sign in, it's necessary to create an account. Once an account is created, find "Unreal Engine" on the left vertical bar under the Epic Games logo, as seen in figure 5. On the upper side lays a horizontal bar, select "Library". Here you can choose which Unreal engine version you wish to install and use.

Figure 5: Unreal Engine version



Source: Author (2024), screenshot from Epic Games Launcher

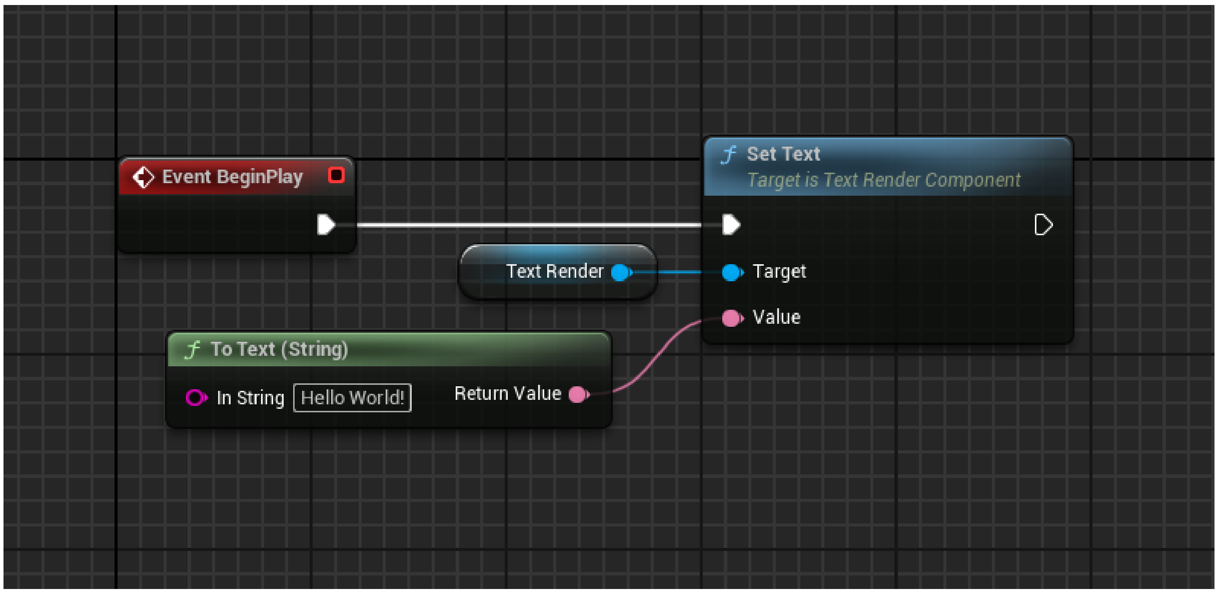
The development was started with the Unreal version 5.2.1, and it was not updated during the development out of fear of something breaking or not working due to newer versions of the engine not being compatible with older versions of the project.

First, you launch Unreal Engine of your chosen version and choose the starting template, which you want to use for your project. Unreal Engine uses both C++ and Blueprints for development (Epic Games, 2024b).

5.1 Blueprints Visual Scripting

Unreal Engine's Blueprint Visual Scripting system offers a node-based interface within Unreal Editor, as seen in blueprint 1. Like traditional scripting languages, it defines object-oriented classes or objects within the engine. This system empowers designers with a wide array of programming tools and concepts, typically reserved for programmers (Epic Games, 2024b).

Blueprint 1: Hello World Example



Source: Author (2024), screenshot from Unreal Engine Editor

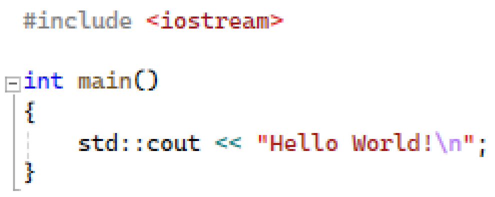
5.2 Programming language C++

C++ is a programming language. It's used by programmers to make software, like games or apps, that run on computers. Many big programs and games you use are made using C++ because it's powerful and flexible (Stroustrup, 2013).

In C++, programmers can write instructions for the computer to follow. These instructions tell the computer what to do, like showing images on the screen or calculating numbers. C++ is known for being fast and efficient, which means it can handle a lot of tasks quickly. Programmers like using C++ because it gives them a lot of control over how their software works (Stroustrup, 2013).

Source code 1 shows us a program written in C++, which after execution prints "Hello World!" in the console. C++ inside Unreal Engine is like regular C++ but with added features tailored to Unreal Engine's objects and actors (Epic Games, 2024b).

Source code 1: Hello, World! in C++



```
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}
```

Source: Author (2024), screenshot from Visual Studio 2022

6. Version Control

Version control is crucial in game development for several reasons. Firstly, game development projects typically involve numerous developers working on different aspects of the game simultaneously. Version control systems like Git allows developers to collaborate efficiently by providing a centralized repository where all code, assets, and other project files are stored. This ensures that team members can work at the same time without the fear of overwriting each other's changes or losing progress (Bryan, 2018).

Firstly, version control enables developers to track changes made to the project over time. This means that if a bug is introduced or a feature breaks, developers can easily pinpoint when and where the change occurred, making it much simpler to identify and fix the issues (Bryan, 2018).

Additionally, version control allows developers to experiment with new features or make significant changes to the project while still maintaining a stable version that can be released (Bryan, 2018).

Lastly, version control systems provide a safety net for game development projects. By regularly committing changes to the repository and creating branches for experimental features or bug fixes, developers can make sure that even if something goes wrong, they can always revert to a previous version of the project. This can be especially important in game development project, where complex systems and interactions can sometimes lead to unexpected results (Bryan, 2018).

6.1 Git

Git is a distributed version control system that manages, archives, and distributes different versions of files. Created by Linus Torvalds in 2005, Git has become the most popular version control software and is the backbone for platforms like GitHub and Bitbucket. It allows multiple developers to work on a project at the same time without overwriting each other's changes. By tracking changes in the source code during development, Git enables developers to keep a historical record of every change, revert to previous versions, and collaborate without conflict. Its distributed nature means every developer has a full copy of the project repository (Haman & Miller, 2022).

Git's key features include its powerful branching and merging capabilities, which supports parallel development. Developers can create separate branches for new features or bug fixes, work on them independently, and later merge them back into the main project. This allows for efficient management of multiple simultaneous versions, making collaboration easier. Getting started with Git involves creating a repository, storing it on platforms like GitHub, tracking changes via commits, sharing changes, retrieving other's changes, recovering old versions and managing branches. Git is also valuable for reproducible research (Haman & Miller, 2022).

6.2 GitHub

GitHub is a web-based platform that serves as a hub for version control and collaboration in software development projects. It leverages the Git version control system, providing developers with a centralized repository for storing code, assets, and other project files. GitHub's intuitive interface and robust features make it a popular choice for developers worldwide, offering tools for code review, issue tracking, and project management. With GitHub, developers can easily collaborate on projects, contribute code changes and track the progress of their work. Its social coding features, such as forks and pull requests, foster a sense of community and enables open-source collaboration, allowing developers to share their code with the world and contribute to projects beyond their own (Bryan, 2018).

7. Application development

In this practical part of the thesis, the focus will be on the process of creating the simulation. This includes the implementation of the ideas and the significant challenges faced during the development.

7.1 Simulation rules

In this simulation, the goal is to model the behaviour of buyers and sellers in a marketplace to understand how prices change over time. The marketplace comprises sellers who set prices for goods and buyers who purchase goods based on their price preferences and budgets. Each day in the simulation represents a trading period where these economic agents interact based on predefined rules. The subsequent adjustments of prices by both buyers and sellers aim to represent the natural market forces that drive prices toward equilibrium.

7.1.1 Seller rules

Sellers adjust their prices based on their success in selling goods each day. If a seller successfully sells their goods, it indicates strong demand at the current price. Consequently, the seller increases their price by a fixed integer amount for example by 1 for the next day. This adjustment tests the market's tolerance for a higher price and allows the seller to maximize profit. Conversely, if a seller does not sell their goods, it suggests that the price was too high for the buyers. To attract buyers, the seller decreases their price by a fixed integer amount for example by 1 for the next day. This reduction makes the goods more competitively priced and increases the likelihood of sales. These incremental adjustments help sellers find an optimal price point that balances supply and demand.

7.1.2 Buyer rules

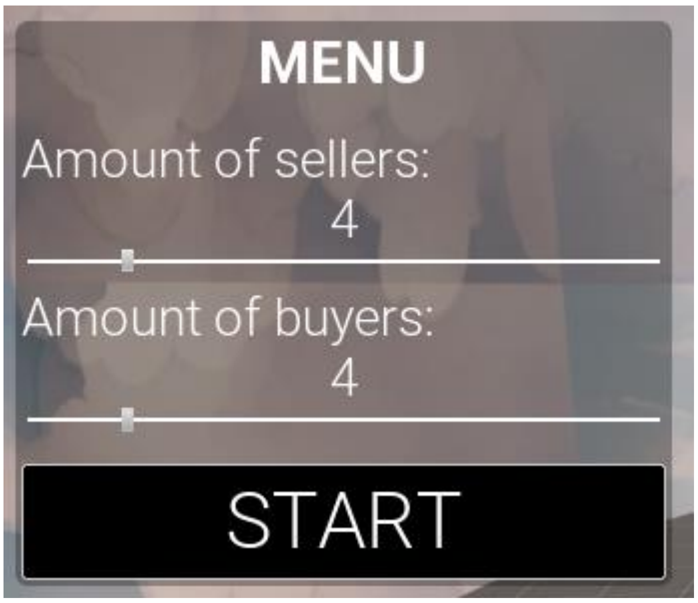
Buyers also adjust their preferred prices based on their purchasing experiences. When a buyer successfully purchases bread, it signifies that the current price was acceptable and affordable. As a result, the buyer increases their preferred price by a fixed integer amount for example by 1 for the next day. This adjustment reflects the buyer's willingness to potentially pay a bit more, acknowledging that they value the bread and are flexible with their spending. On the other hand, if a buyer does not purchase bread, it indicates that the prices were too high. Therefore, the buyer decreases their preferred price by a fixed integer amount for example by 1 for the next day. This decrease shows the buyer's intention to find more affordable options in the future. These adjustments help buyers refine their price expectations based on market conditions.

The iterative process of price adjustments by both sellers and buyers drives the market toward equilibrium. Sellers learn to set prices that reflect actual demand, while buyers adjust their willingness to pay based on their purchasing power and market prices. Over time, this dynamic interaction helps balance supply and demand, leading to a stable market price that satisfies both parties. By simulating these rules, the application hopes to show economic behaviours and provide insights into how markets naturally regulate prices through the actions and reactions of buyers and sellers. The fixed integer adjustments ensure simplicity and clarity in the simulation, making it easier to observe and analyse the path to market equilibrium.

7.2 Menu

In the application, users are initially presented with an empty map accompanied by a menu located in the upper left corner, as seen in figure 6. This menu allows the user to select the number of sellers and buyers for the simulation, as well as to initiate the simulation by clicking the start button.

Figure 6: Initial Menu

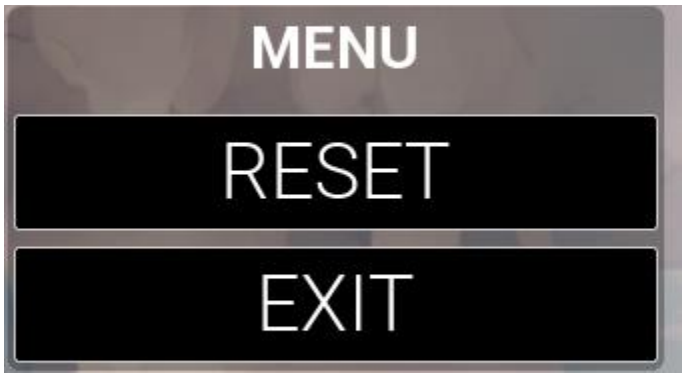


Source: Author (2024), screenshot from Unreal Engine

The menu is constructed using a canvas panel, which allows the arrangement of widgets in specific locations and allows them to be anchored relative to other elements within the canvas. This canvas panel serves as an abstract space for the menu. To further create the menu a border widget is added to the canvas panel to contain a child widget. The label “MENU” is displayed using a simple static text widget.

Upon clicking the “START” button the simulation begins, and a new widget appears, as seen in figure 7. Presenting two options the “RESET” button, which restarts the simulation from scratch and displays the initial menu again (see figure 7) and the “EXIT” button, which closes the game.

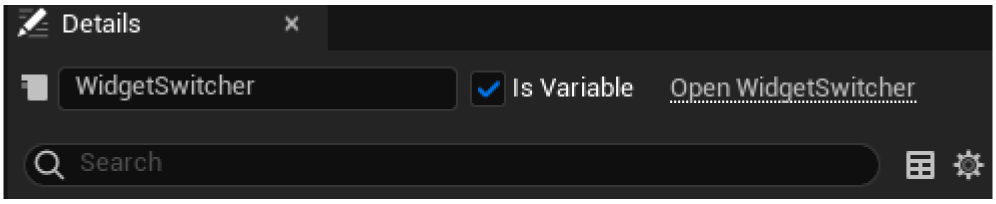
Figure 7: During Simulation Menu



Source: Author (2024), screenshot from Unreal Engine

To ensure proper functionality of widget switching and slider operations, it is crucial to enable the check boxes representing true and false values, as seen in figure 8. This step is necessary to implement the logic effectively.

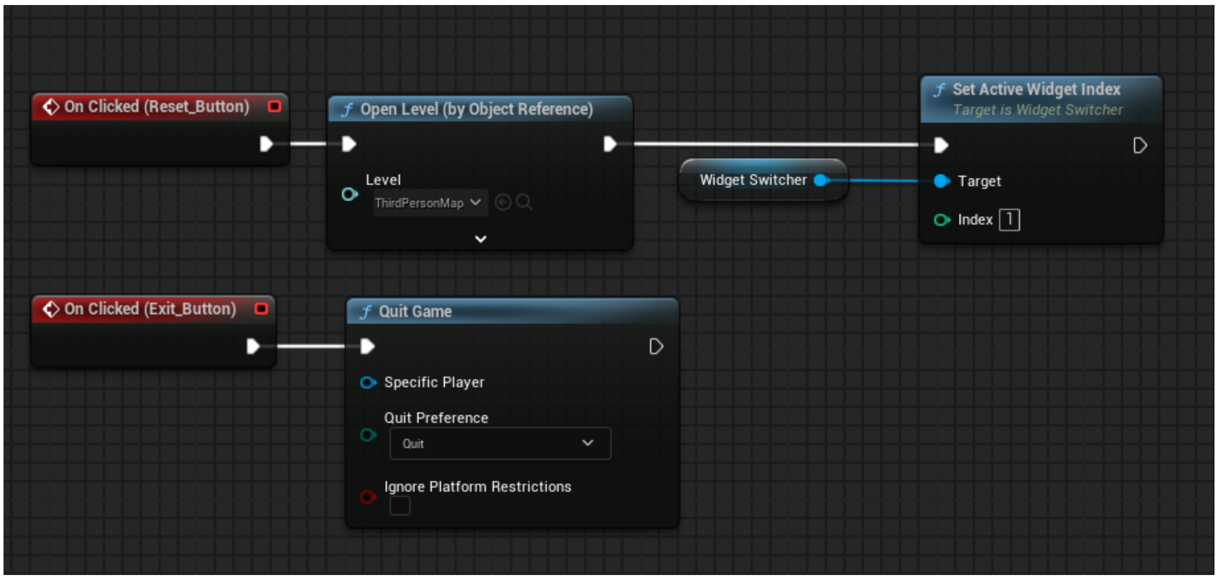
Figure 8: Is Variable Value



Source: Author (2024), screenshot from Unreal Engine Editor

The “Event Graph” is the area where all the widget logic is created, as seen in blueprint 2. To program the functionality of the reset and exit buttons, it is necessary to switch from the designer window to the graph window. By selecting the button widget in the designer, you can access its properties under the “Details” panel. Here, you can find the events section and choose the “On Clicked” event to define the button’s functionality.

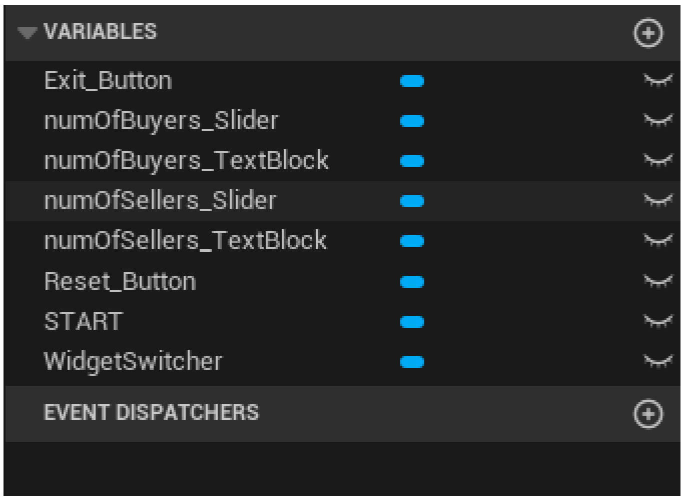
Blueprint 2: Reset and Quit buttons.



Source: Author (2024), screenshot from Unreal Engine Editor

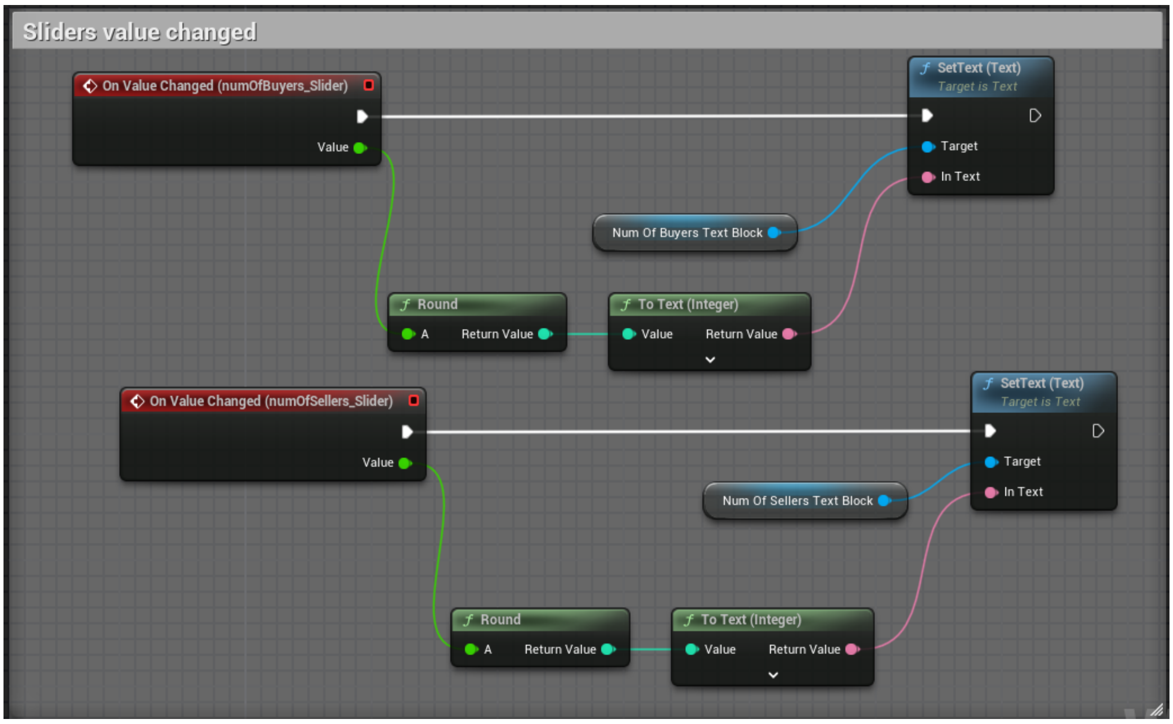
The slider widgets require the "Is Variable" to be set to true. Like the reset and exit buttons, the implementation of the slider's functionality is carried out in the Event Graph (See blueprint 3). It is important to note that for the widget switcher we want to change it to the initial menu. Figure 9 shows us all the variables that the widget blueprint holds. Like the number of buyers, sellers, their text and lastly the buttons.

Figure 9: Blueprint Widget Variables


VARIABLES
Exit_Button
numOfBuyers_Slider
numOfBuyers_TextBlock
numOfSellers_Slider
numOfSellers_TextBlock
Reset_Button
START
WidgetSwitcher
EVENT DISPATCHERS

Source: Author (2024), screenshot from Unreal Engine Editor

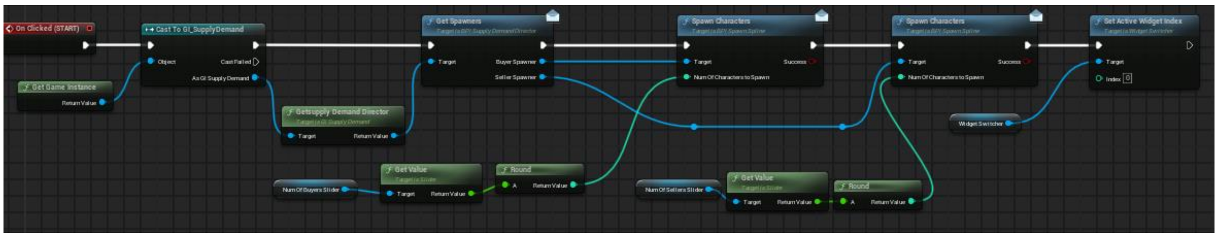
Blueprint 3: Slider Value Change



Source: Author (2024), screenshot from Unreal Engine Editor

Implementing the logic for the "START" button in the opening menu was particularly challenging, as seen in blueprint 4 or attachment 1 (larger version). Initially, I needed to create a game instance by connecting the appropriate nodes. Numerous functions were then developed to initiate the simulation. These functions include retrieving all sellers and buyers and spawning the respective characters.

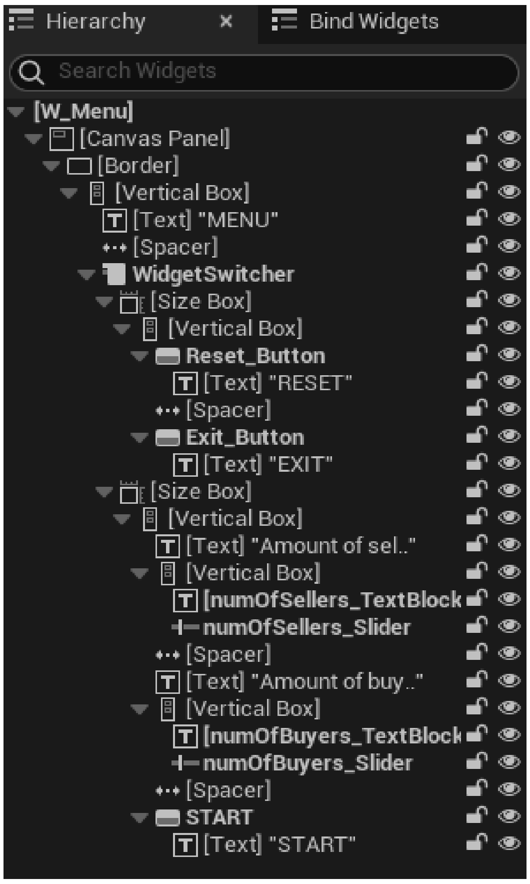
Blueprint 4: Start Button implementation.



Source: Author (2024), screenshot from Unreal Engine Editor

All components for the menu were created within the Widget Blueprint. The parent-child relationships of the widgets can be viewed in the hierarchy section of the Widget Blueprint as seen in figure 10.

Figure 10: Menu Widget Hierarchy



```
Hierarchy    Bind Widgets
Search Widgets
[W_Menu]
  [Canvas Panel]
    [Border]
      [Vertical Box]
          [Text] "MENU"
          [Spacer]
        WidgetSwitcher
          [Size Box]
            [Vertical Box]
              Reset_Button
                [Text] "RESET"
              [Spacer]
              Exit_Button
                [Text] "EXIT"
          [Size Box]
            [Vertical Box]
              [Text] "Amount of sel.."
              [Vertical Box]
                [numOfSellers_TextBlock
                numOfSellers_Slider
              [Spacer]
              [Text] "Amount of buy.."
              [Vertical Box]
                [numOfBuyers_TextBlock
                numOfBuyers_Slider
              [Spacer]
              START
                [Text] "START"
```

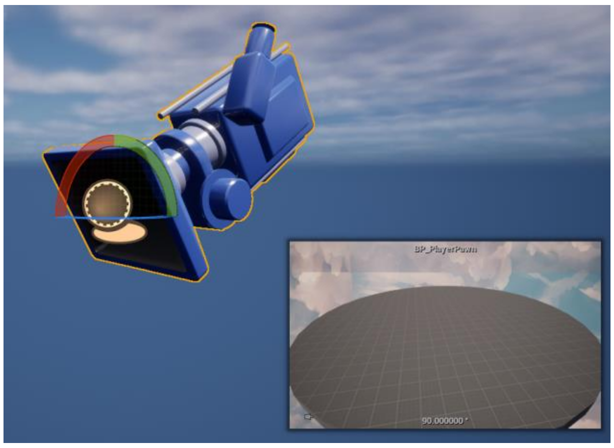
Source: Author (2024), screenshot from Unreal Engine Editor

7.3 Main character

The main character in my simulation differs from traditional characters that moves using WSAD keys. Instead, I chose a static camera as the main character, as seen in figure 11.

The primary reason for this decision is that the focus of the simulation is on observing the interactions between supply and demand elements rather than navigating through the game environment. By using a static camera, the player can have a clear and consistent view of the entire simulation process, making it easier to monitor changes without the distraction of moving a character around.

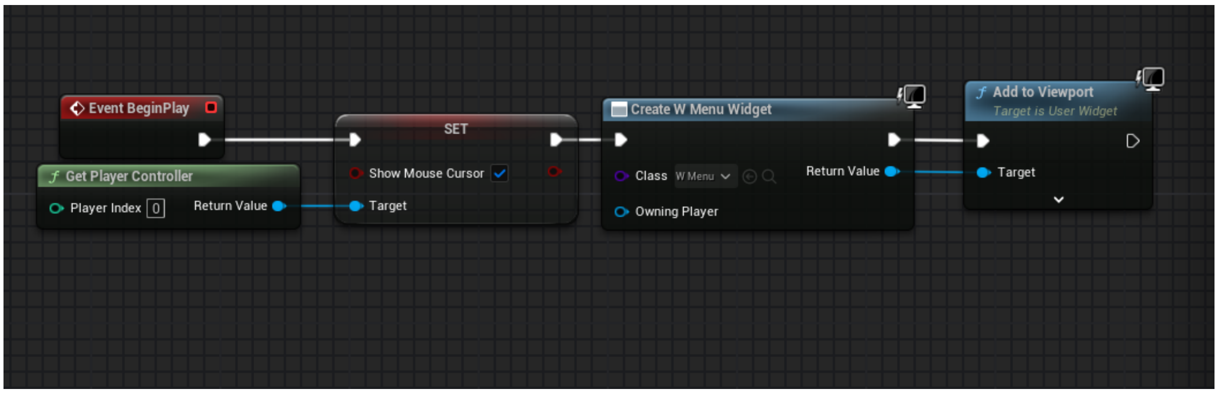
Figure 11: Main Character Camera



Source: Author (2024), screenshot from Unreal Engine

To accomplish this, a Blueprint class was created, inheriting from the Pawn class. Within the Viewport, a Camera component was added. The Event Graph, as seen in Blueprint 5. Contains three custom events: "Event ActorBeginOverlap", "Event Tick" and "Event BeginPlay". If nodes are not executed, they will be automatically disabled and will not be called. Only the "Event BeginPlay" node needed activation. To display the mouse cursor, we first obtained the player controller. Subsequently, we created a node called "Create W Menu Widget" and selected the widget blueprint "W_Menu" as the class. To make sure the widget is visible on the screen, a node "Add to Viewport" was added, and all nodes were connected accordingly.

Blueprint 5: Event Begin Play for Player Pawn

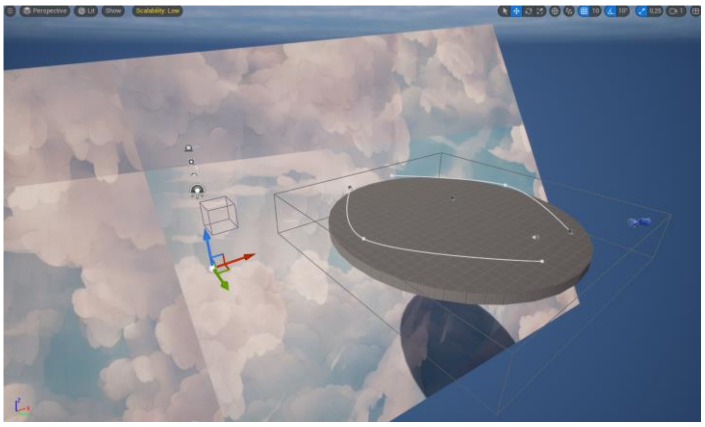


Source: Author (2024), screenshot from Unreal Engine Editor

7.4 Virtual world

To run the simulation, a map was needed. A simple round circle was made using four quarter cylinder meshes. Instead of a plain blue background, a large cube was used. The cube was scaled up to display a picture as the background. It was done to create an illusion of a prettier environment, which can be seen in figure 12.

Figure 12: The virtual world.



Source: Author (2024), screenshot from Unreal Engine

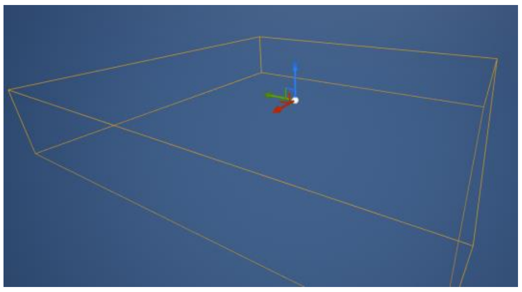
The texture was created using an AI image generator available for free on the website Canva.com. By entering the prompt "clouds" and choosing the "hand painted" as the graphics option the resulting texture was generated. This texture was then downloaded as a PNG file and imported into Unreal Engine by dragging and dropping it into the Content Drawer.

7.5 Characters

Unreal Engine offer two mannequins, male and female, that are freely available to all users, providing a versatile and highly detailed character model for various projects. This mannequin is a pre-rigged, humanoid model that can be easily integrated into any Unreal Engine project. It comes fully equipped with a comprehensive set of animations, including walking, running, and jumping. In my simulation game developed within Unreal Engine, I utilized the freely available mannequin as the primary character model for my buyers and sellers.

It is important to place the "NavMeshBoundsVolume" first because it defines where the non-player characters can move in a game level, as seen in figure 13. When placed in a level, it creates an area that tells the engine where to generate Navigation Mesh. It's a data structure used to make pathfinding easy for non-player characters.

Figure 13: Navigation Mesh

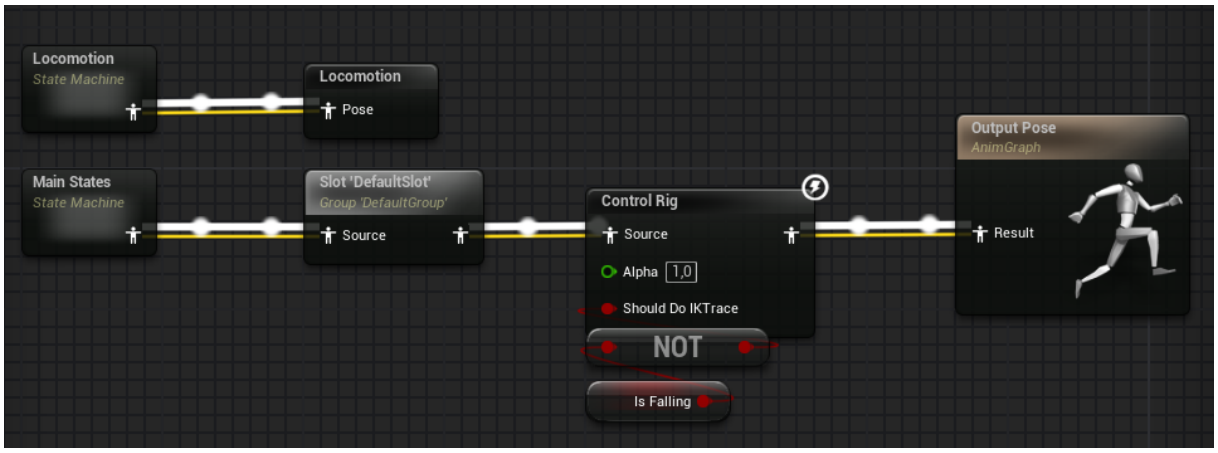


Source: Author (2024), screenshot from Unreal Engine

7.5.1 Animation

For the characters to walk, run or jump a "Animation Blueprint" class is needed. The Animation Blueprint class in Unreal Engine is a special type of blueprint used to control the animations of a character. It allows developers to create complex animation behaviours without writing code by using a visual scripting system. Within an Animation Blueprint, you can define how different animations blend, how the character transitions between movements like walking and running, as seen in figure 14.

Figure 14: Anim Graph example.

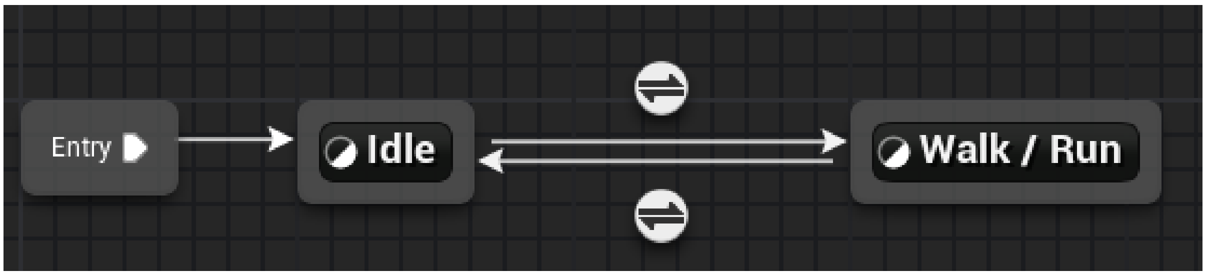


Source: Author (2024), screenshot from Unreal Engine Editor

The AnimGraph is a part of the Animation Blueprint, which focuses on making characters move. With the AnimGraph, developers can visually link different animation parts to control how a character acts and switches between movements.

The main thing in the AnimGraph is the State Machine. This helps set different animation states, like standing, walking, and running. Each state can have its own animation, as seen in figure 15.

Figure 15: State Machine example.

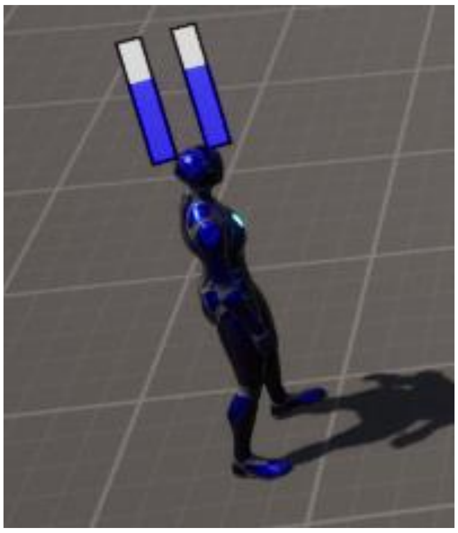


Source: Author (2024), screenshot from Unreal Engine Editor

7.5.2 Seller

In my simulation game, I made a seller character to show the supply. This seller acts as a virtual shopkeeper, selling goods to buyers or to those who want them. Every morning, it appears on the stage and waits for someone to come and buy things, as seen in figure 16. The seller doesn't move from its spot.

Figure 16: Seller character.



Source: Author (2024), screenshot from Unreal Engine

The character class uses the SKM_Quinn skeletal mesh, while the animation blueprint class utilized is ABP_Quinn_C. My Blueprint class, Seller, is a child of a more general blueprint class named BP_SupplyDemandCharacter. Additionally, it implements an interface called BPI_Seller.

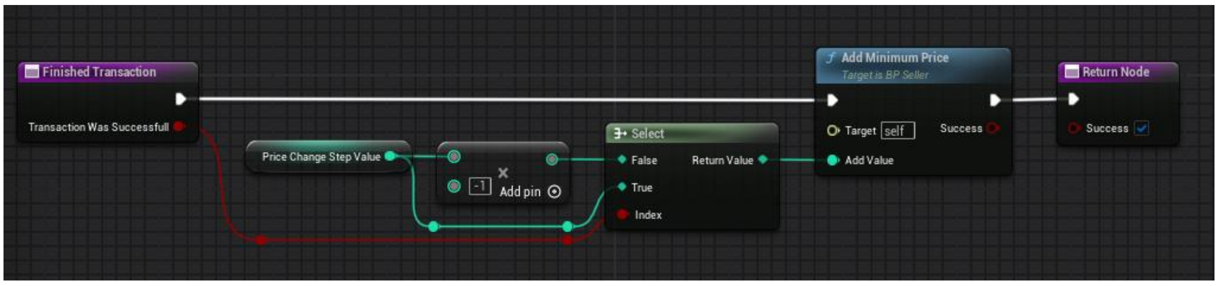
The Seller class inherits from the Character class and has seven implemented functions:

- **Finished Transaction:** Takes a Boolean as input and returns a Boolean.
- **Get Minimum Price:** Outputs an integer.

- **Set New Minimum Price:** Takes an integer as input and returns a Boolean.
- **Get Current Buyer Reference:** Outputs a reference to an Actor object.
- **Set New Buyer Reference:** Takes an actor object reference as an input and returns a Boolean.
- **Add Minimum Price:** Takes an integer as an input and returns a Boolean.
- **Setup Character:** Inherited from its parent class.

Blueprint 6 starts with "Finished Transaction" event, checking if the transaction was successful using a branch node. If successful, it takes a "Price Change Step Value," processes it through a "Select" node, and passes the result to the "Add Minimum Price" function to adjust the price. Finally, it ends with a "Return Node" indicating the success of the process. If the transaction is not successful, the price change does not occur.

Blueprint 6: Finished Transaction



Source: Author (2024), screenshot from Unreal Engine Editor

7.5.3 Buyer

Similarly, I designed a buyer character to represent demand, as seen in figure 17. This buyer behaves like a virtual shopper, purchasing goods from sellers. Each morning, it heads out to search for a seller to buy a product from. After the successful transaction or unsuccessful it heads back to its starting position waiting for the next day to come.

Figure 17: Buyer character.

Source: Author (2024), screenshot from Unreal Engine

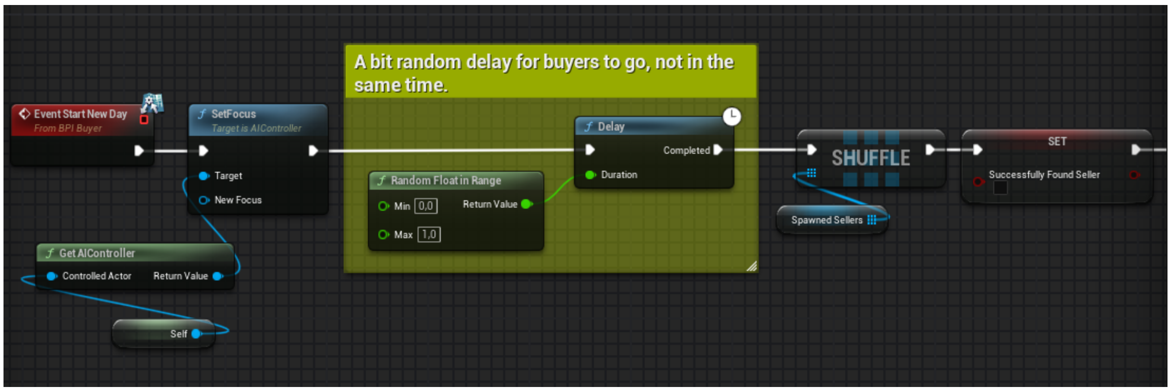
The character class also uses the SKM_Quinn skeletal mesh and the animation blueprint class utilized is ABP_Quinn_C too. My Blueprint class, Buyer is also a child of the blueprint class BP_SupplyDemandCharacter. Additionally, it implements an interface called BPI_Buyer.

The Buyer class inherits from the Character class and has three implemented functions:

- **Start New Day:** Implemented in the Event Graph
- **Add Maximum Price:** Takes an integer as an input and outputs a Boolean.
- **Set Maximum Price:** Takes an integer as input and returns a Boolean.

After testing, I decided to add a delay for the buyers, as seen in blueprint 7. This makes the simulation look better because it prevents all the buyers from moving at the same time and bumping into each other on their way to a seller. Makes the experience smoother.

Blueprint 7: Buyer movement delay.



A bit random delay for buyers to go, not in the same time.

Source: Author (2024), screenshot from Unreal Engine Editor

The biggest obstacle, where I spend numerous weeks in this project was to create the simulation where the buyer would be able to go randomly to a new single seller each day. For each seller to have a unique buyer, as can be seen in blueprint 8.

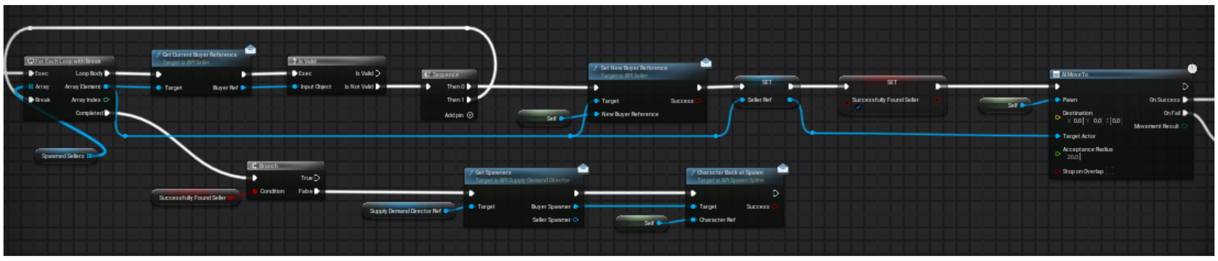
It starts by identifying the current buyer that needs to find a seller. This is done using a function called “Get Current Buyer Reference”, which ensures that the buyer is valid and can participate in the next steps.

Next, the blueprint loops through a list of potential sellers. For each seller in the list, it checks if the seller is valid. This means the seller must exist and be capable of interacting with the buyer. If a seller is not valid, the blueprint skips to the next seller in the list.

Once a valid seller is found, the blueprint assigns this seller to the current buyer. It sets the buyer’s reference to this seller, indicating that the buyer now knows which seller to go to. This step is crucial as it links the buyer with a specific seller.

Finally, as seen in the blueprint 8 or attachment 3, at the end it uses a function to move the buyer towards the seller’s location. This makes the buyer character physically move in the simulation to interact with the assigned seller. The blueprint marks this process as successful once the buyer starts moving towards the seller.

Blueprint 8: Buyer finding Seller.



Source: Author (2024), screenshot from Unreal Engine Editor

7.5.4 Transaction

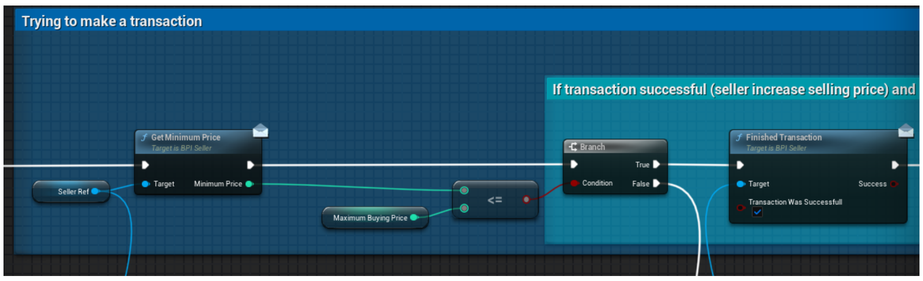
The transaction process between buyers and sellers in the simulation works like this. The process begins with a short delay to ensure actions are carried out in a less awkward fashion. This delay acts like a waiting period, making sure that each buyer takes turns when trying to make a purchase.

The next step involves the buyer attempting to make a transaction with the seller. The buyer first checks with the seller to find out the minimum price the seller is willing to accept, as seen in blueprint 9 or attachment 4. This is like asking, “What’s the lowest price you will sell it for?" The buyer then compares this minimum price to the maximum amount they are willing to pay, as seen as the Maximum Buying Price variable.

If the buyer's maximum price is equal to or greater than the seller's minimum price, the transaction is successful. This means the buyer can afford the purchase, and the seller agrees to sell at that price. If the transaction is successful, the seller is notified, and the buyer prepares to pay slightly more for future transactions. This also makes the buyer increase their desired price, making it easier for future purchases.

Finally, the buyer's maximum price is updated by adding a specific value to it. This makes that the buyer is willing to pay a slightly higher price for the same goods in the next transaction.

Blueprint 9: Transaction.



Source: Author (2024), screenshot from Unreal Engine Editor

7.6 Supply and Demand

In the simulation game, I use bar graphs to show how much people want to buy and sell things in the market. Imagine a central square where people have goods they want to sell (sellers), and other people want to buy goods (buyers). The bar graphs help us understand the changes that are happening.

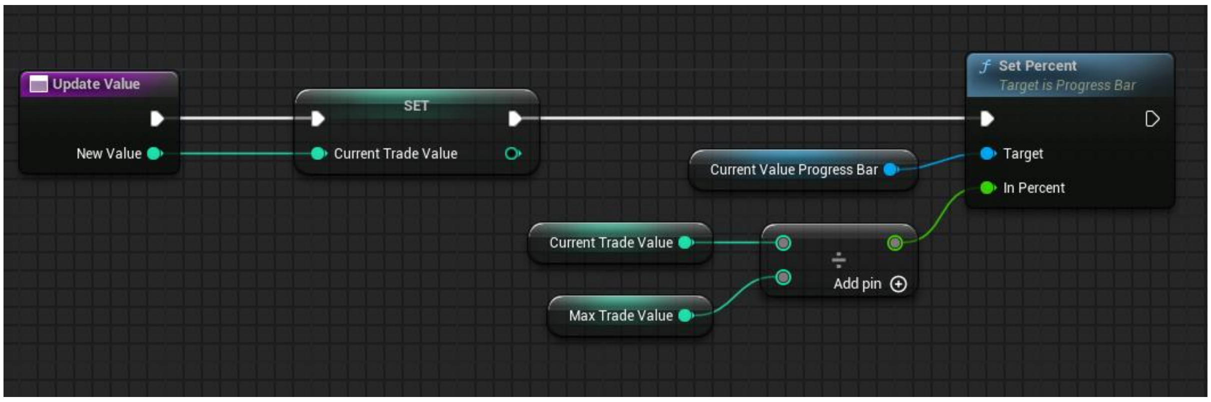
Each buyer and seller in the game has two bars next to each other. For buyers, the first bar shows the most money they are willing to pay for a toy (maximum price), and the second bar shows the price they are offering right now (current buying price). For sellers, the first bar shows the least money they want for their item (minimum price), and the second bar shows the price they are asking for right now (current selling price). The first bar is slightly lighter in colour to further help the player see a difference between the two.

In the game, these bar graphs help players understand how the market is working. By looking at how tall the bars are, players can see if prices are going up or down based on how much people want to buy and sell. The goal of these bars is to be a simple visual representation to help user understand the dynamics of supply and demand within the simulated market.

When a transaction is successful, the blueprint follows the “Success” path. First, it updates the system to indicate that the transaction was successful, notifying the player. Then, it adjusts the prices based on this successful transaction, updating the seller’s records and future prices for both buyers and sellers. The final step in this path makes all necessary updates are completed, making sure everyone knows about the successful transaction and adjusts accordingly.

On the other hand, if a transaction fails, the blueprint follows the “Fail” path. It starts by updating the system to indicate that the transaction failed, informing the player of the failure. The blueprint then takes steps to manage the failed transaction, which involves displaying a sad emoji. The final steps ensure all necessary updates are completed for the failure, including informing the buyer and seller and adjusting their expectations, as seen in blueprint 10.

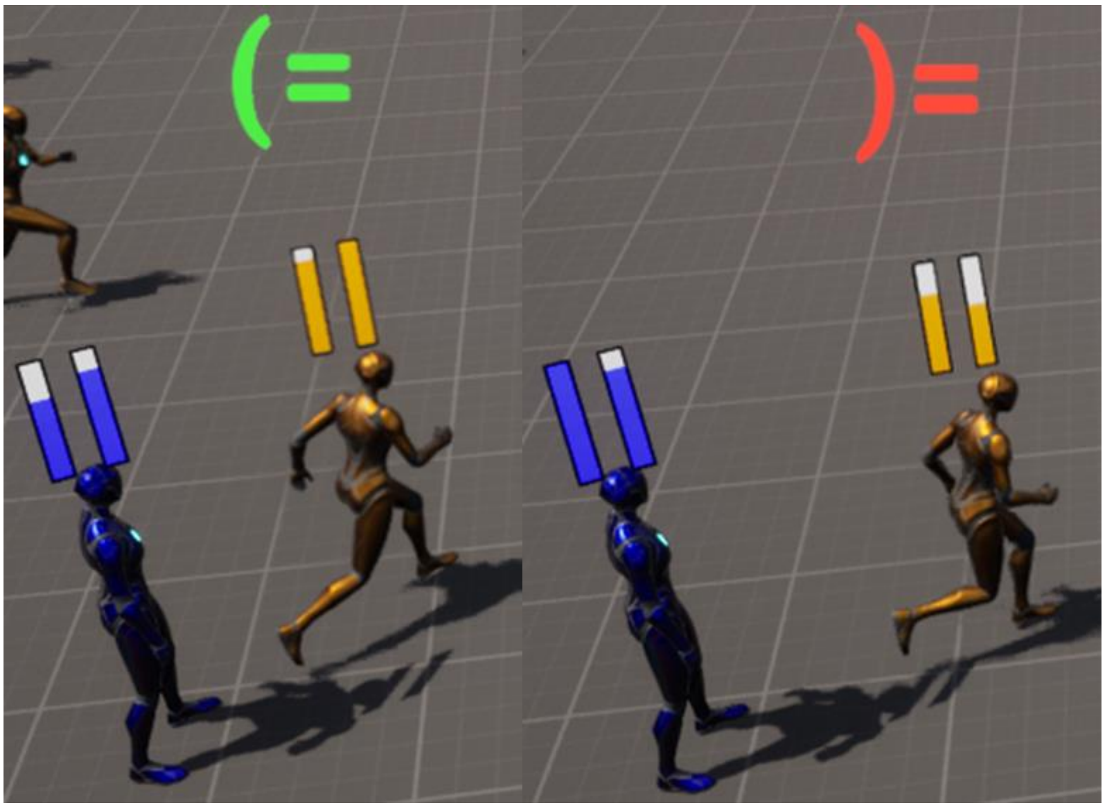
Blueprint 10: Update Value inside Widget Blueprint



Source: Author (2024), screenshot from Unreal Engine Editor

In this game, I have added a special feature to enhance the user experience during the simulation. When a transaction is successful, a bright green smiley face emoji pops up above the buyer for a few seconds. Indicating a positive outcome. However, if the transaction is unsuccessful, a red sad smiley face emoji appears instead, expressing disappointment, as seen in figure 18.

Figure 18: Successful and Unsuccessful visual representation.

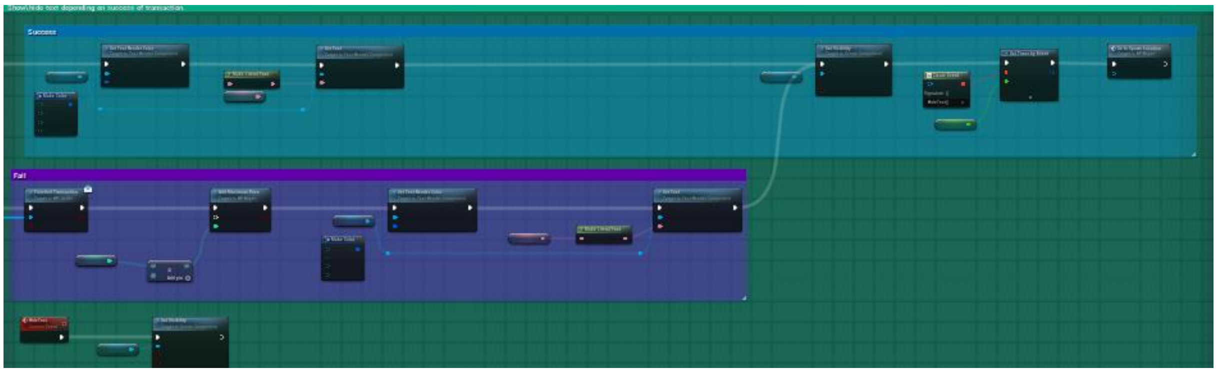


Source: Author (2024), screenshot from Unreal Engine

The goal of this visual feedback system was to add a playful and funny touch to the transaction process but also provide clear and intuitive communication about the success or failure of each transaction, making the experience more engaging and user-friendly.

In the following blueprint 11 can be seen the blueprint visual code to achieve this feature. A node was created and connected to display respective characters upon their execution. The attachment 5 shows it better, the successful upper path and in attachment 6 is the lower unsuccessful path.

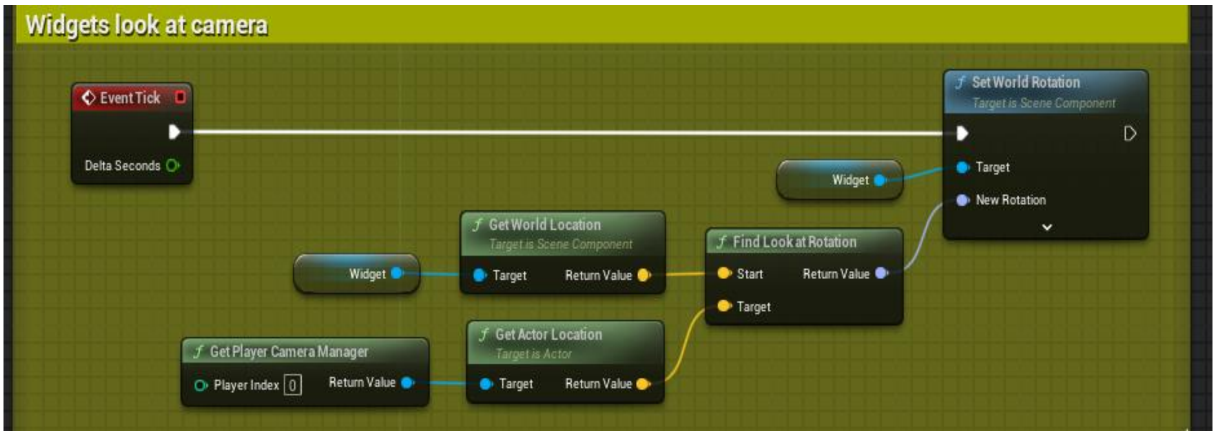
Blueprint 11: Success and Fail emoji.



Source: Author (2024), screenshot from Unreal Engine

One problem I had during the development was that the widgets were always out of place during the simulation. Turning away, disappearing and not being very clear. To enhance visual cohesion, I have coded the widgets in such a way that they always face towards the camera, as seen in blueprint 12. This means that no matter where the buyers and sellers are within the simulation, the widgets maintain their orientation towards the camera, preventing them from appearing out of place or randomly oriented. This makes the widgets become clear visual feedback for the user.

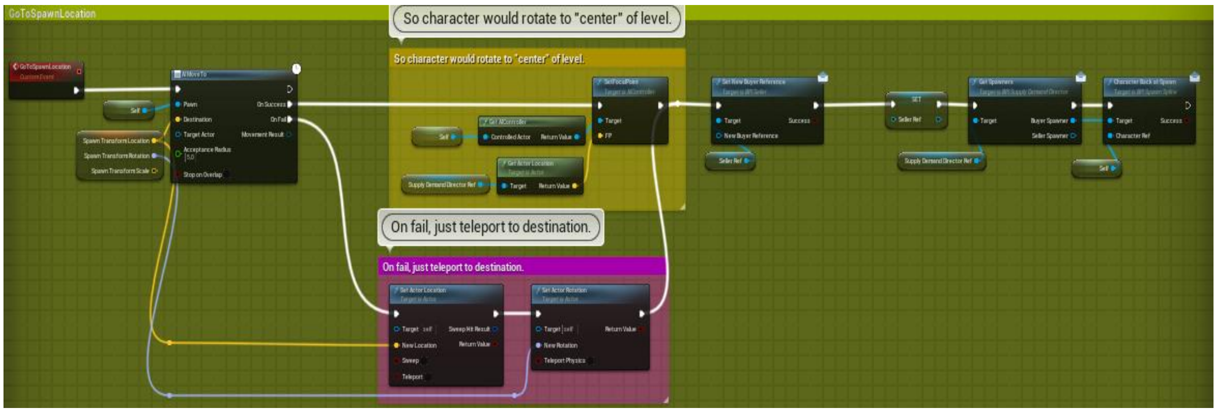
Blueprint 12: Widgets look at camera.



Source: Author (2024), screenshot from Unreal Engine Editor

At the end of the transaction a call function is executed. A custom event is called. This event works like a signal, telling the buyer to go back to where they started from, kind of like hitting the rewind button, as seen in blueprint 13 or attachment 7. So, after all the buying is done, the buyer is transported back to where they first began the simulation, and the next cycle starts again once all characters are back at spawn, or the player presses the button "RESET" or "EXIT".

Blueprint 13: Go to Spawn Location Custom Event

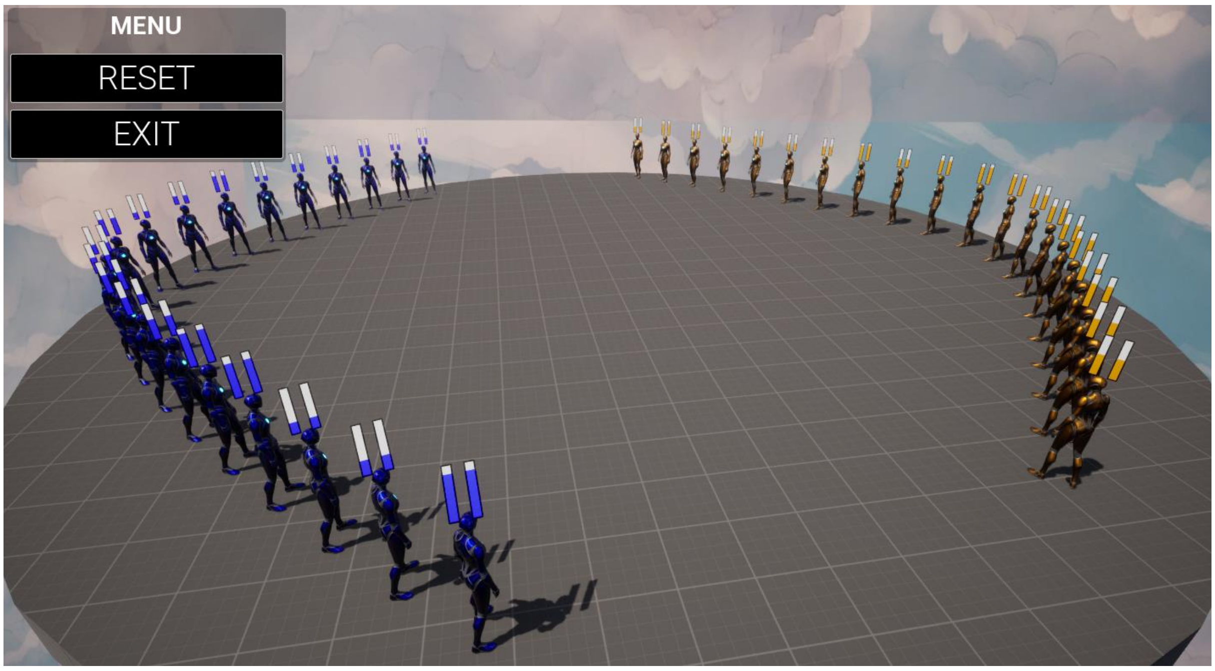


Source: Author (2024), screenshot from Unreal Engine Editor

7.7 Spline Spawner

During the development, one of the biggest problems I faced was figuring out how to spawn the buyers and sellers. After much trial and error, I devised a solution by creating a Blueprint class. This Blueprint class enabled the spawning of buyers and sellers along a spline a curved line within the game environment, as seen in figure 19. Along this spline, I implemented the logic to spawn buyers and sellers. This helped me summon the desired number of characters. Two spline blueprints were put into the map. One used for spawning buyers and the other used for spawning sellers.

Figure 19: Buyers and Sellers spawned along a spline.



Source: Author (2024), screenshot from Unreal Engine

The Spline class inherits from the Actor class and has three implemented functions:

- **Spawn Characters:** Takes an integer as an input and outputs a Boolean.
- **Character Back at Spawn:** Takes an Actor object reference as an input and outputs a Boolean.
- **Get Spawned Characters:** Outputs an array of an Actor.

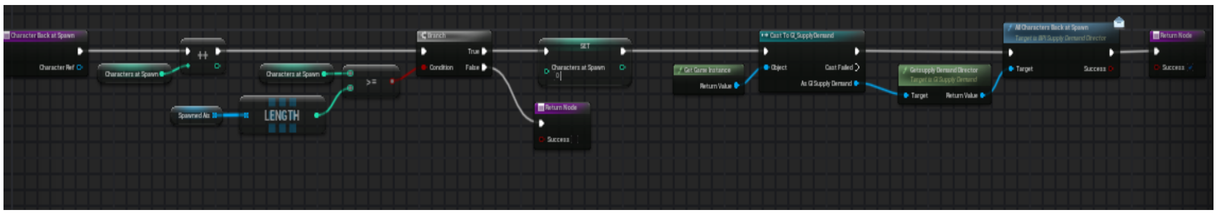
For the next day to begin it was crucial to make sure all the characters were back at spawn and to do this the function "Character Back at Spawn" was implemented, as seen in blueprint 14 or attachment 8. First, it had to wait for a message from the Buyer class. Once a character is back at their spawn point. A counter is incremented to keep track of how many characters have returned. This count is then compared to the total number of spawned characters in the game to determine if all characters are back at their starting positions.

The sequence first checks if all characters are accounted for by comparing the number of characters at the spawn point with the total number of characters in the game. If the number of characters at the spawn point is equal to or greater than the total number of spawned characters, the next node activates.

Once all characters are confirmed to be back at their spawn points, the blueprint retrieves the game instance. This instance is then cast to a specific type (GI_SupplyDemand) to access the functions required for the next steps. If the cast is successful, it proceeds to get the supply and demand director from the game instance. The director manages the loop logic inside the game.

Finally, when all characters are back at their starting positions, a message is sent to the supply and demand director to start the function "All Characters Back at Spawn". The sequence then ends, returning a successful bool.

Blueprint 14: Characters back at spawn.



Source: Author (2024), screenshot from Unreal Engine Editor

7.8 Director

The final piece of this simulation was to create a blueprint class that managed the simulation. The primary role of the director is to loop the simulation. It's placed in the virtual world and doesn't have any visual representation.

The blueprint class Director inherits from the Actor class and has two implemented functions:

- **Get Spawners:** Outputs an Actor object reference.
- **All Characters Back at Spawn:** Outputs a Boolean.

The final problem of the game was to infinitely loop the game, which the function "All Characters Back at Spawn" manages to do. The transition between the end of one day and the start of a new day within the game. It begins by checking if all characters have returned to their starting positions, known as the spawn point. This initial check makes that the game is ready to continue to the next day, as seen in blueprint 15 or attachment 2.

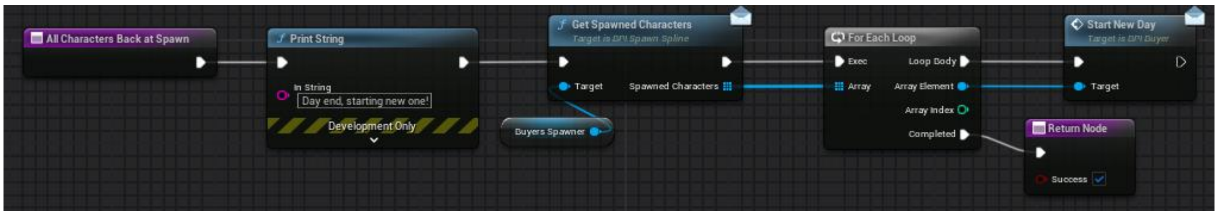
Moreover, the blueprint prints a debug message, “Day end, starting new one!” This message is important during development as it confirms to me that the day cycle logic is working correctly.

The sequence then retrieves a list of all characters that have been spawned in the game. This list is stored in an array, which is essentially a collection of all the active characters.

Following this, a loop processes each character in the array individually. This loop, called the “For Each Loop” performs actions on each character one by one. This node secures that every character is accounted. The loop structure is efficient for handling multiple characters, making sure that no character is missed in the process.

Finally, once all characters have been processed, the blueprint triggers the start of a new day. This step involves activating the call function, which sends a message to the blueprint interface buyer. By doing this, the game starts the next day cycle.

Blueprint 15: Starting a new day.



Source: Author (2024), screenshot from Unreal Engine Editor

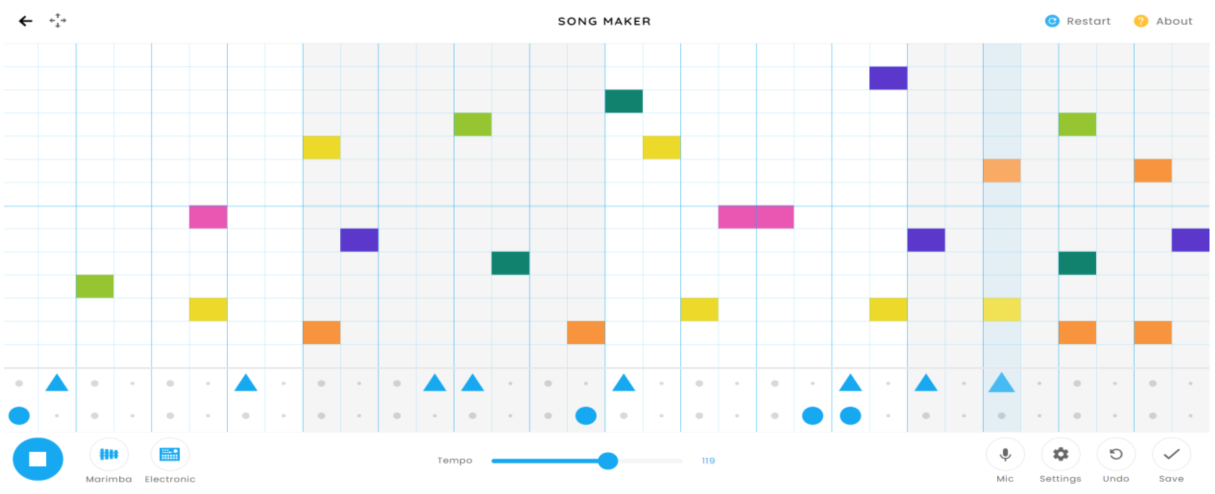
7.9 Music creation

To make the simulation game more fun, exciting and to prevent it from feeling dull or monotonous. Background music was created. The aim was to produce an upbeat track to keep it lively and excite the user. The music was designed with the capability to loop indefinitely. This was accomplished using a website called “Chrome Music Lab”.

The website allows users to interact with instruments spanning two octaves, from C to B. Available instrument options include marimba, piano, strings, woodwind, synth, electronic, blocks, kit, and conga. Users can freely adjust the tempo using a slider, with a range from 40 to 240 beats per minute. Additionally, there is a feature to activate the microphone, allowing users to add notes based on their voice input.

Furthermore, the website features a variety of interactive experiments that visually demonstrate musical concepts such as rhythm, melody, and sound waves, as seen in figure 20. Each experiment uses intuitive graphics and animations to make the experience more fun. For instance, users can create rhythmic patterns by toggling beats (pressing on them) on the grid. These tools not only create a fun time of music creation but also motivates a creative exploration, making the entire learning process very enjoyable and accessible for users of all ages and skill levels.

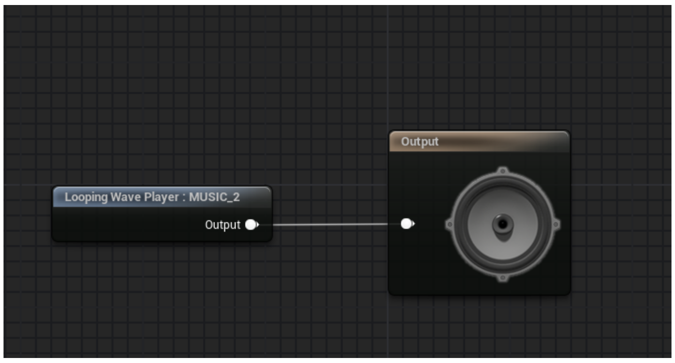
Figure 20: Chrome Music Lab website music creation example.



Source: Author (2024), screenshot from Chrome Music Lab website

The music track was exported as a WAV file and imported into Unreal Engine. Upon importing, a cue was generated by right clicking the file and selecting the appropriate option. To enable looping of the music, a looping node was created and connected to the output, as seen in figure 21. Integrating the music into the simulation required dragging and dropping the cue into the level.

Figure 21: Music Cue



Source: Author (2024), screenshot from Unreal Engine Editor

8. Results

8.1 Scenario 1

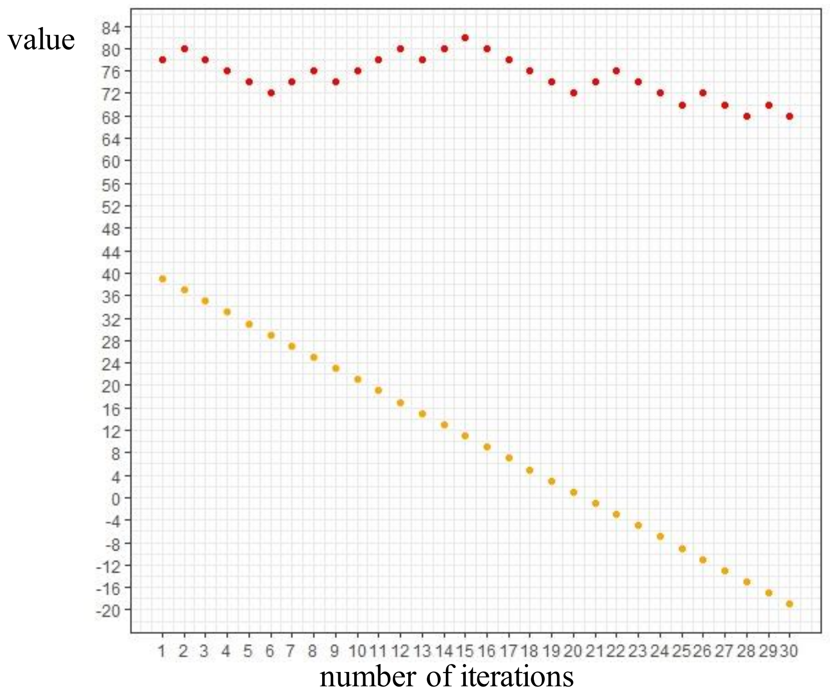
Let’s consider a finite number of buyers and sellers. All participants move independently of each other. Each seller has their own place randomly located within a predetermined space. Buyers move randomly within this space. At the beginning of the simulation, each participant is assigned a random initial value within the range (a, b), where a < b and both are real numbers. The amount of money for a buyer is denoted as i_b and for a seller as i_s.

When a buyer encounters a seller, the following situations can occur:

- $If\ i_s < i_b$, an interaction happens, and both participants increase their value by 2.
- $If\ i_s > i_b$, the seller makes no sale, and both participants decrease their value by 2.

In the initial phase of the application, it is certain that each seller will find a buyer. The application runs thirty times, recording the results of all successful encounters between buyers and sellers in each iteration. Participants gain experience from each encounter for future interactions. The goal is for the amounts of buyers and sellers to “equalize” by the end. The development of the amounts for the first pair (buyer0-seller0) is illustrated in graph 1. In following graphs, the “x” axis will represent the iteration cycle, and the “y” axis will represent the value they are willing to buy or sell for in short value represents money.

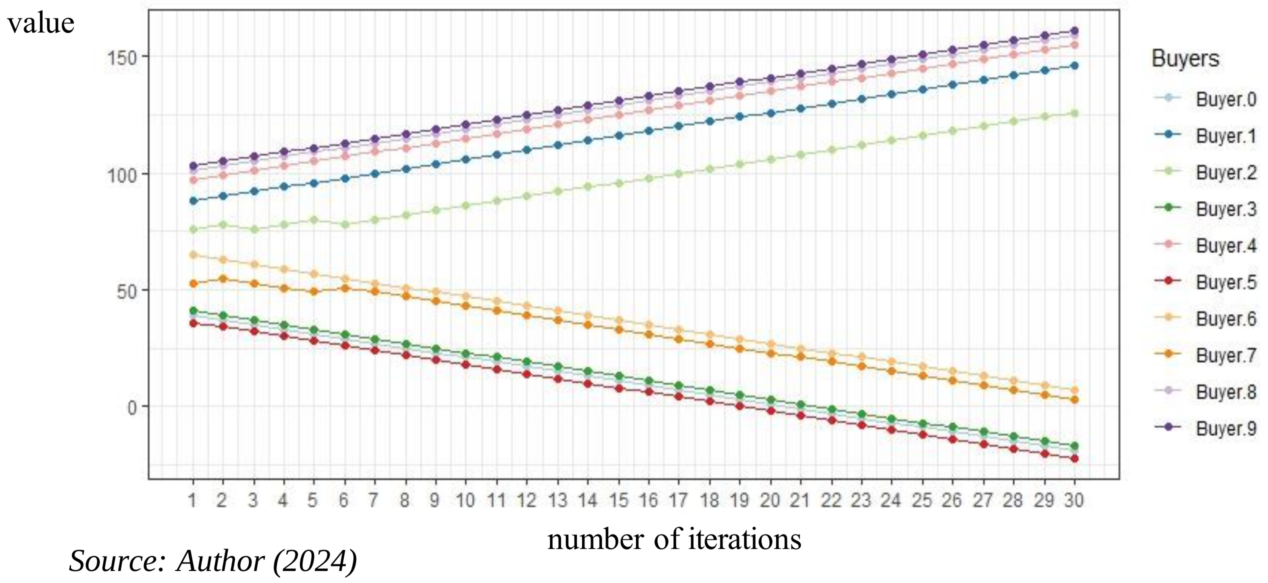
Graph 1: Buyer0 and seller0 development



Source: Author (2024)

The development number of all 10 buyers is summarized in the following graph 2. The colour of each curve corresponds to the specific buyer according to the legend. It can be seen from this graph 2 that half of the buyers increase their profit, and the other half of the participants decrease their profit. Considering the set goal, it is reasonable to believe that the initial conditions in the application may not have been set in the most appropriate way.

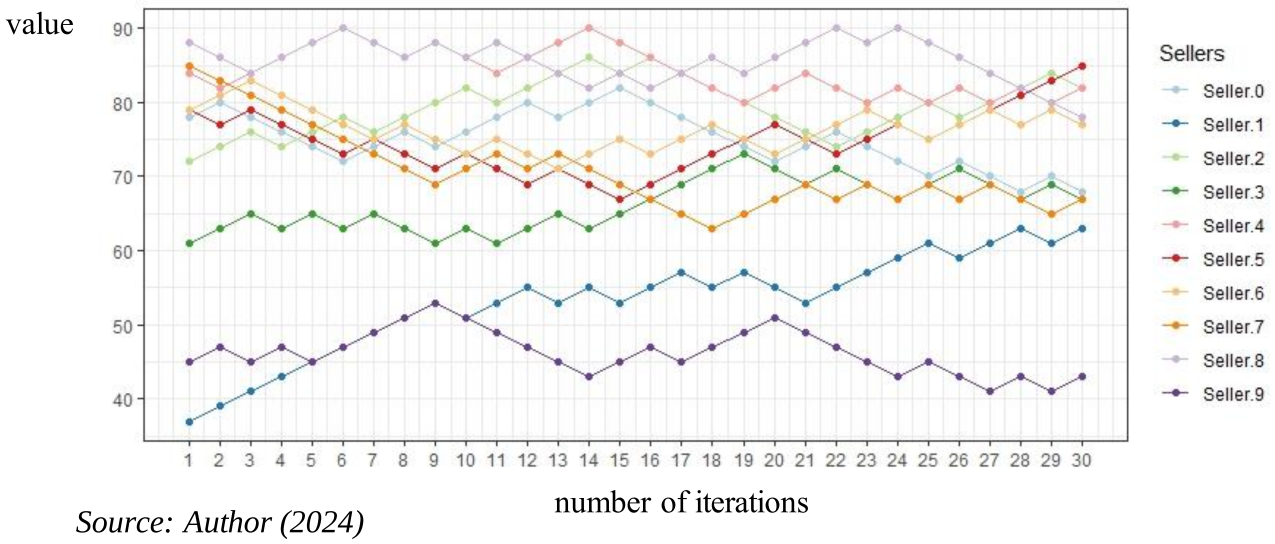
Graph 2: The development of all 10 buyers



Source: Author (2024)

The development of amount of all 10 sellers is seen in graph 3. The color of each curve corresponds to the amount of a particular seller. The colors of the two graphs 2 and 3 correspond to the respective “buyer-seller” “pairs”, the first pair of which is shown above. It can be seen from the last graph 3 that there is a rather variable dynamic in the amount of sellers amounts, for example, Seller1 ends up with a significantly higher amount than he started with.

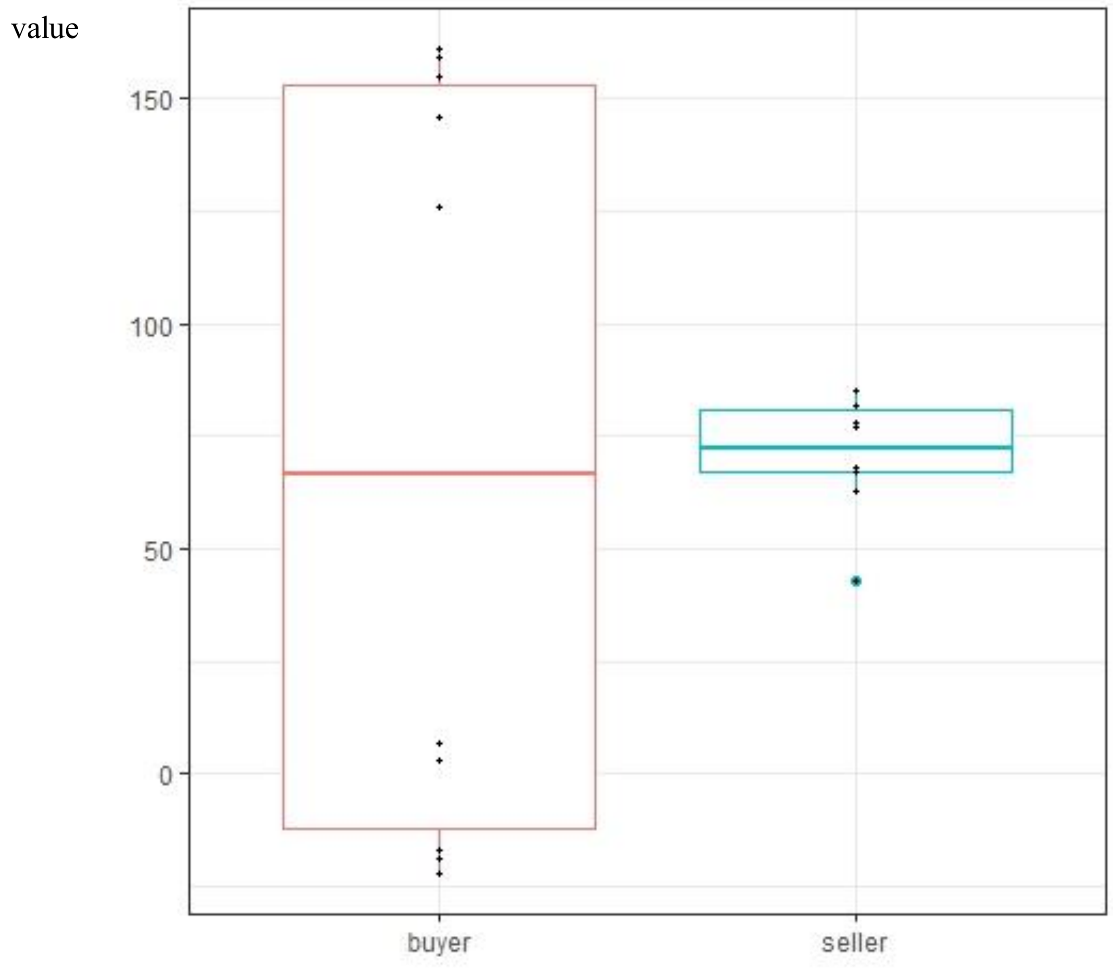
Graph 3: Development of all 10 sellers



Source: Author (2024)

In the following graph 4, two box graphs are shown. Each boxplot corresponds to one of the two monitored groups of participants. Buyers belong to the red boxplot in the left part of graph 4 and sellers belong to the turquoise boxplot in the right part of graph 4. The black points inside the boxplots are the final values of each participant after the 30th cycle of the application.

Graph 4: Boxplots of buyer and seller

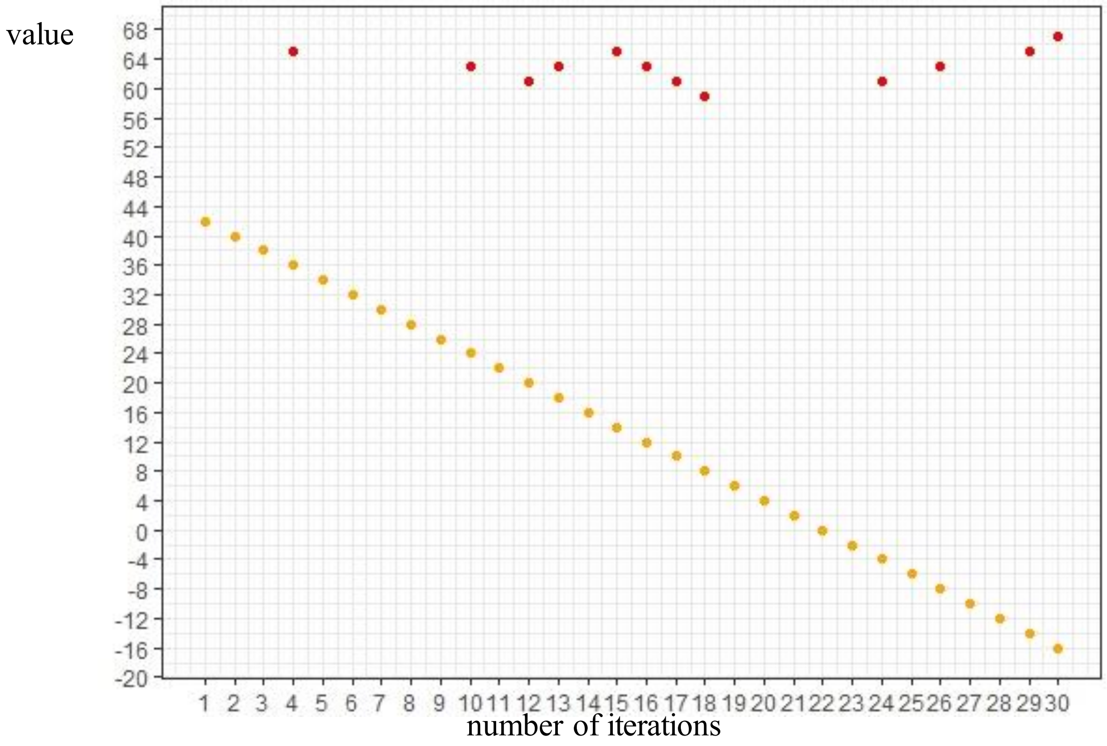


Source: Author (2024)

8.2 Scenario 2

The second simulated scenario was done in a similar manner with 30 iterations and 10 sellers but this time the number of buyers were 5. In the following graph 5 we see the development of buyer0 and seller0 for the second scenario where the number of sellers decreased to 5 compared to the first scenario where the number of buyers and sellers were equally 10.

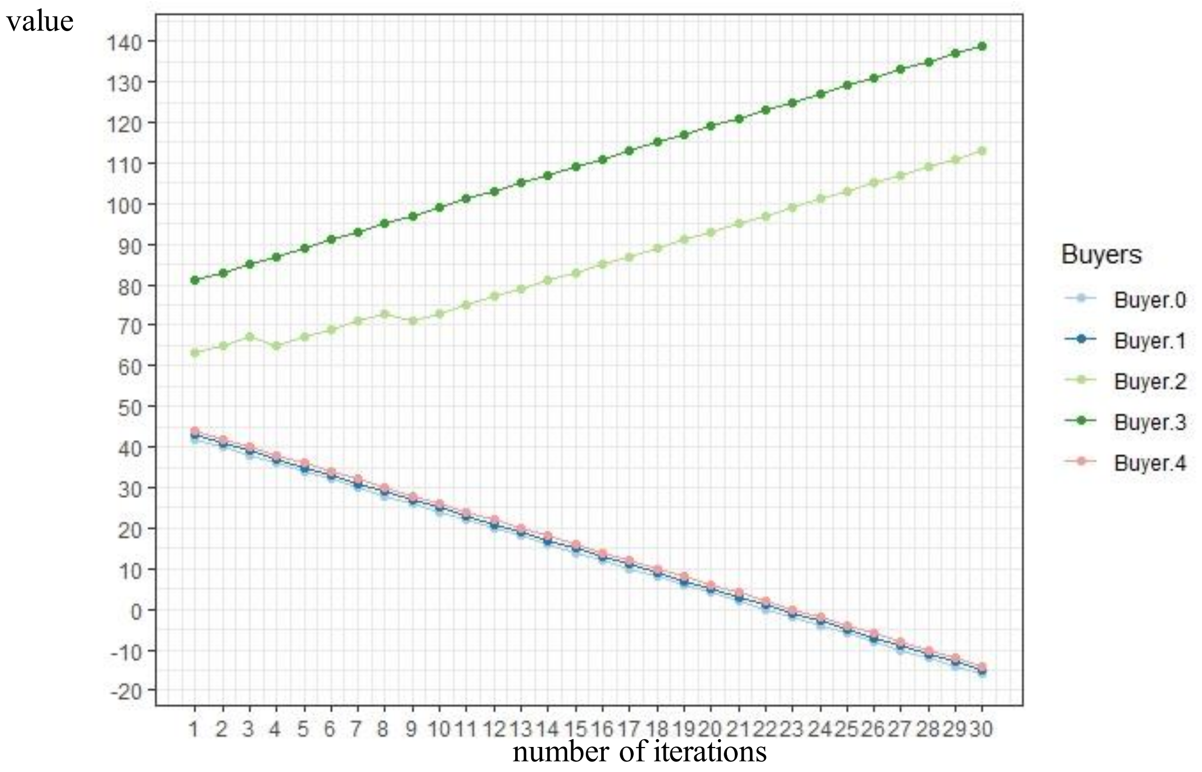
Graph 5: Buyer0 and seller0 development in second scenario



Source: Author (2024)

The following graph 6 shows the development of all 5 buyers in the second scenario. The colour of each curve corresponds to the specific buyer according to the legend.

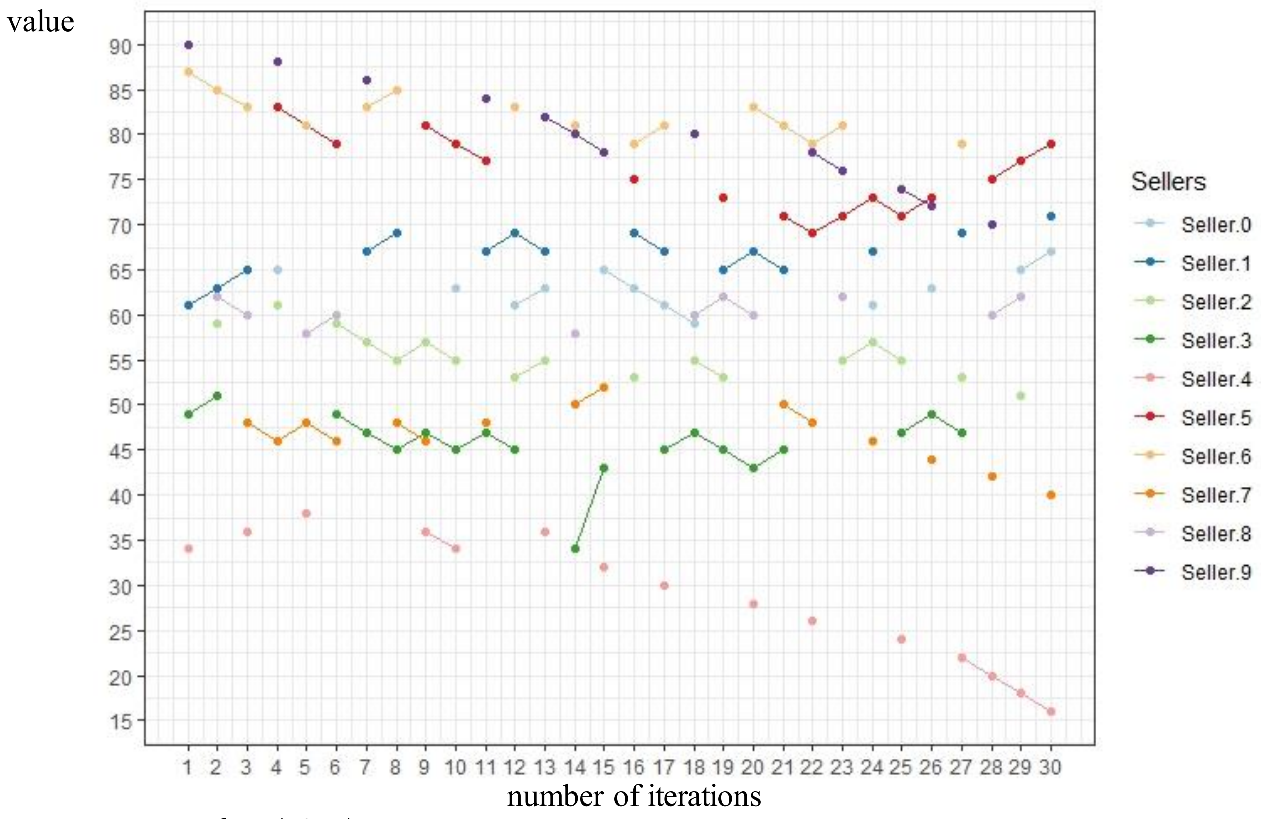
Graph 6: The development of all 5 buyers in second scenario



Source: Author (2024)

The following graph 7 shows that, each curve in the graph represents a specific seller, and the points in the graph show the interactions made. The curve for each seller is piecewise continuous, with breaks indicating cycles where no interaction occurred between the buyer and seller. Points are connected by lines when contacts happen in successive cycles otherwise, they are discrete.

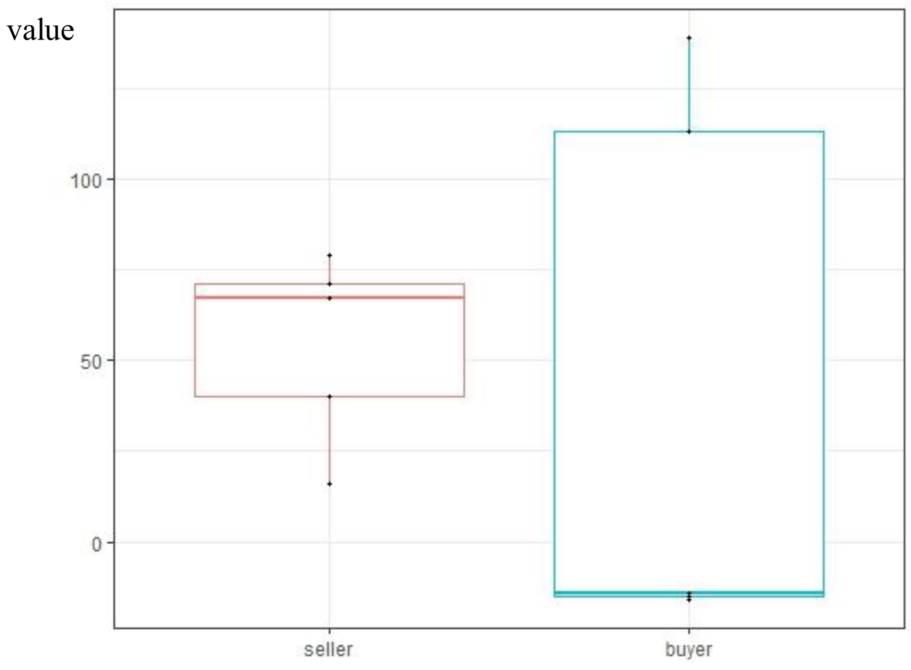
Graph 7: The development of 10 sellers in second scenario



Source: Author (2024)

Graph 8 shows two box plots, each representing one of two groups of participants. The red box plot on the left represents buyers, and the turquoise box plot on the right represents sellers. The black dots inside the box plots show the final amounts after 30 cycles. These final amounts correspond to two independent random samples. The horizontal line in the middle of each box plot indicates the average value for that group. The asymmetry of the box plots suggests that the samples come from an asymmetric probability distribution, meaning the average value does not match the median. In the 30th cycle, there were five successful buyer interactions and 7 successful seller interactions. The size of the box plots reflects the variance of the amounts in each group, indicating different variances for buyers and sellers.

Graph 8: Boxplots of buyer and seller second scenario

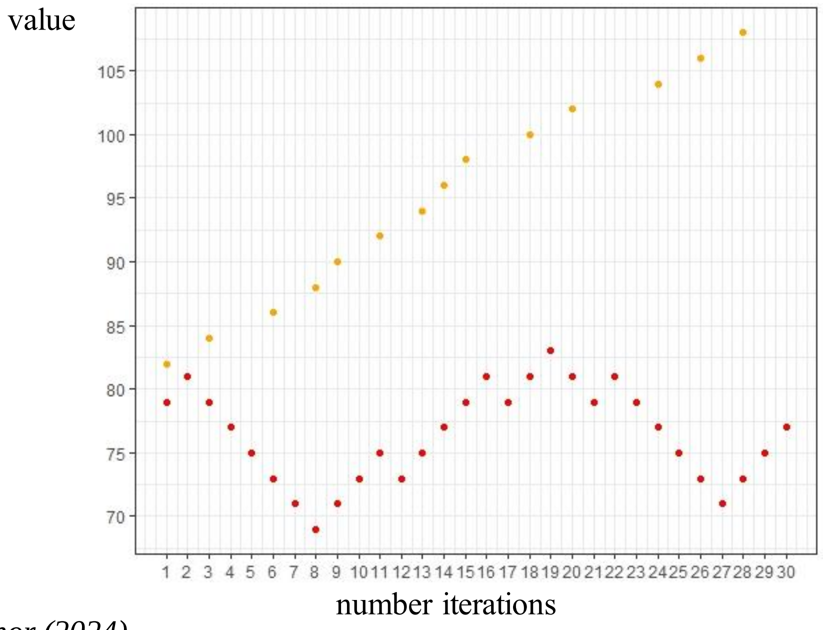


Source: Author (2024)

8.3 Scenario 3

This scenario is like the second scenario but with a key difference. The number of sellers is five and the number of buyers equals 10. This reduced certainty lowers the buyer's chance to gain experience from interaction, though it is still certain that each seller will meet some buyer. Looking at the first pair (buyer0 and seller0), we get the following graph 9.

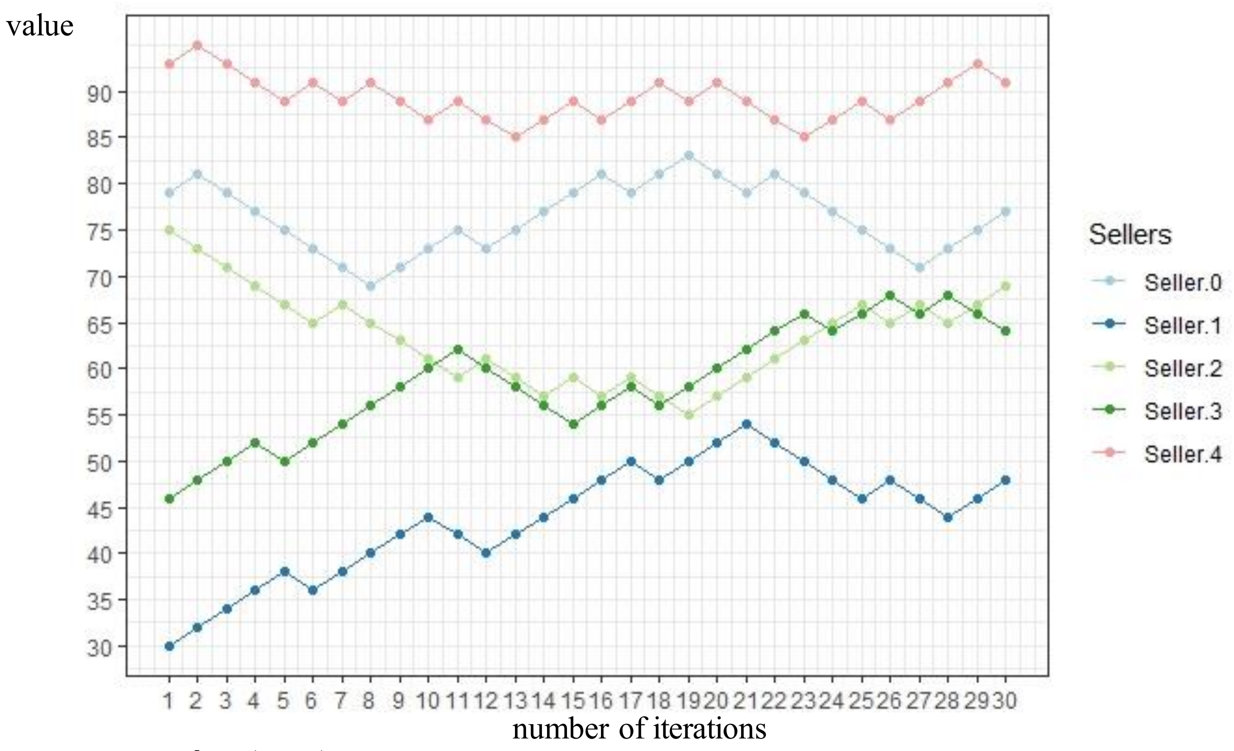
Graph 9: Buyer0 and seller0 development in third scenario



Source: Author (2024)

The following graph 10 shows the development of 5 sellers in the third scenario. The colour of each curve corresponds to the specific seller according to the legend.

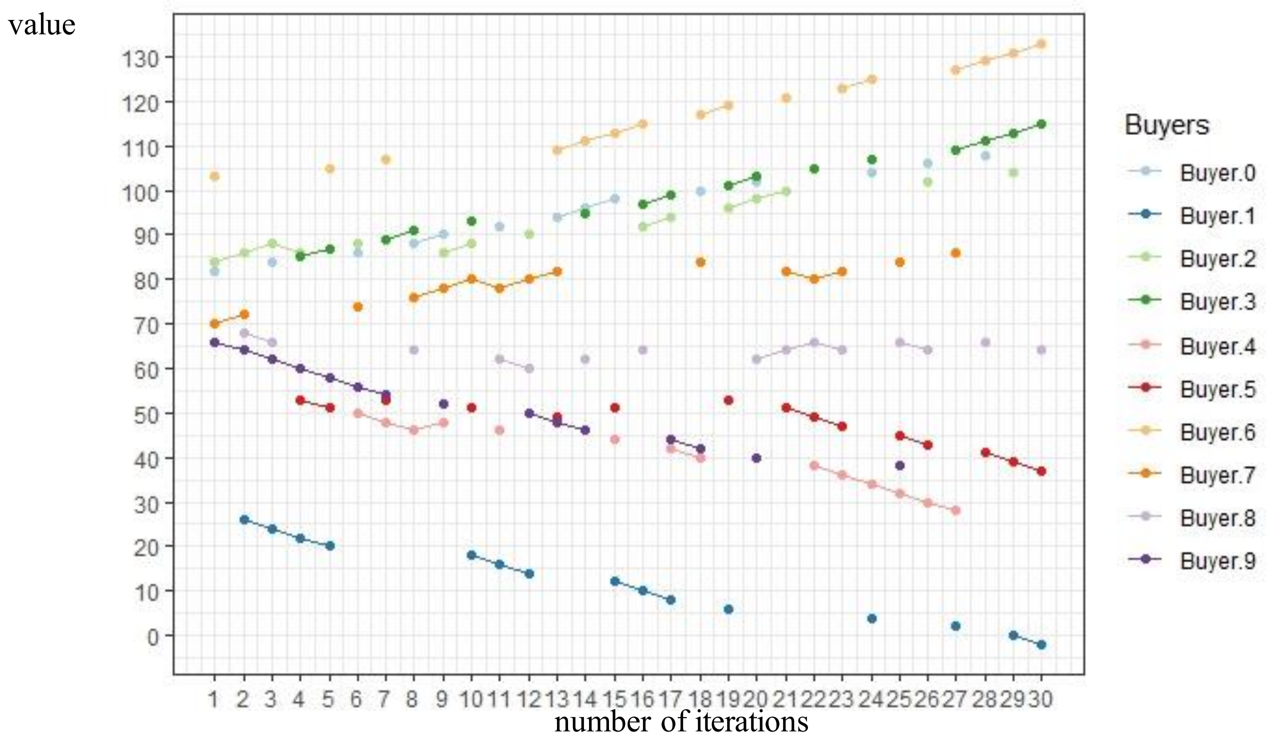
Graph 10: The development of 5 sellers in third scenario



Source: Author (2024)

In graph 11, each curve represents a specific buyer. The points on the graph represent actual interactions made. The curve for each buyer is piecewise continuous because not every cycle sees contact between the seller and buyer. Lines connect two or more points in the graph when interactions happen in consecutive cycles. Otherwise, the function is represented by discrete points in that section. From graph 11, it is evident that 5 buyers increase their earnings over thirty cycles. For instance, Buyer6 starts with the value 103 and ends with 133. Additionally, it shows that 4 buyers decrease their earnings over cycles, for example, Buyer1 begins with the value 8 and ends with -2 after thirty cycles. The colours in the last two graphs (10 and 11) correspond to specific "pairs" with the first pair detailed earlier in graph 9.

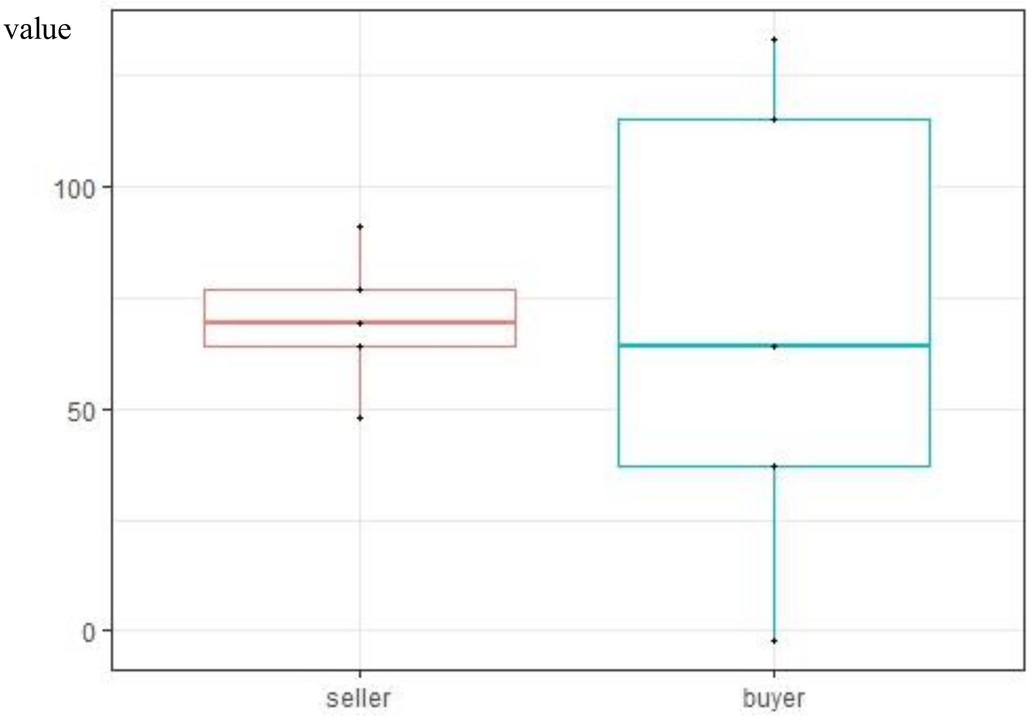
Graph 11: The development of 10 buyers in third scenario



Source: Author (2024)

In graph 12, there are two box plots. Each boxplot represents one of two groups of participants being observed. Buyers are represented by the red boxplot on the left side and sellers are represented by the turquoise boxplot on the right side. The black dots inside the boxplots show the final amounts after 30 cycles of the application running. These final amounts come from two independent random samples. The horizontal line in the middle of each boxplot represents the average value for that group. Notably, in this Scenario 3 scenario, 5 sellers and 7 buyers had successful interactions during the 30 cycles. The size of the boxplots reflects the variance in the amounts for each group of participants. This scenario illustrates a situation where demand exceeds supply, affecting the balance between buyers and sellers.

Graph 12: Boxplots of buyer and seller in third scenario



Source: Author (2024)

9. Discussion

This part is focused on discussing the flaws of the simulation game. What could have been done better what could be improved. The experiences I gained throughout the entire project.

9.1 Game critique

In the current implementation of the simulation game, the rules for adjusting prices are applied only when a buyer interacts with a seller. If no interaction occurs between a buyer and a seller, the price adjustment rules are not triggered. This presents a flaw in the simulation, as it doesn't consider for the experiences of sellers who do not find a buyer to interact with. This oversight fails to simulate the reality of market dynamics where unfulfilled demand and unsold supply should also influence future pricing decisions.

When there are a lot of buyers, sellers, or characters in general, it becomes overcrowded which can lead to awkward cases, where they bump into each other. The map wasn't optimized for a horde of roaming characters.

One could argue that the base models offered by the game engine is not very visually appealing. Custom created models for the characters could make the experience better. Giving each character a more distinguished look. The map could have been better designed too. Creating unique three-dimensional models to sell the user of an actual market. For example, creating a wooden stall with apples, oranges, and other fruits. All of this would contribute to selling the illusion of a marketplace to the player.

There are a lot of free models created by the unreal engine community inside their marketplace that is accessible via their Epic Games launcher. Alternatively, if using marketplace models isn't wanted. Software like Blender, which is free can be used to create three-dimensional models.

9.2 Personal growth

My growth during the last few months have been massive. I went into this project with zero experience with game engines or game creation in general. It was my first-time using Blueprints and C++. I have gained a significantly deeper understanding of everything that is involved in game creation. No wonder, people always say it takes a gigantic team of people specializing in many different fields to bring a game to life, not to mention that it also consumes a lot of time.

10. Conclusion

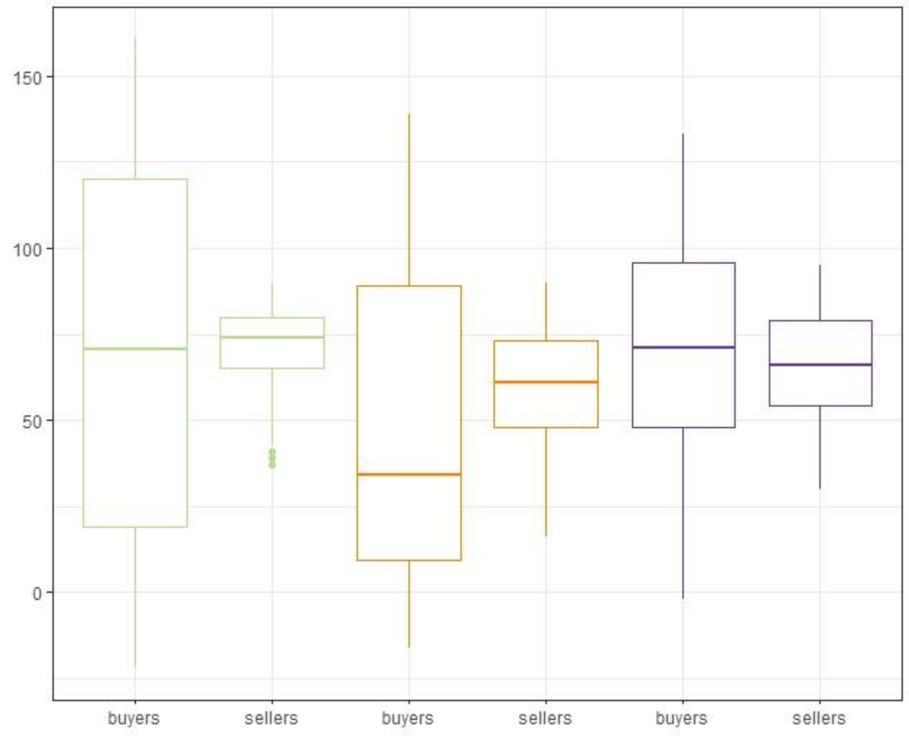
The objective of this thesis was to develop an application capable of simulating supply and demand in real time. To achieve this goal, a video game engine was used. Unreal Engine has powerful tools to bring projects to life in a three-dimensional environment.

The beginning of the thesis was focused on understanding the fundamental concepts of supply and demand in microeconomics. Building upon this understanding, the thesis then dived into the history of video games, recognizing their important role in both entertainment and education.

Moreover, the thesis proceeded to dive into the utilization of game engines, with a particular emphasis on Unreal Engine, as a powerful software for the creation of an immersive and interactive environment.

In the practical part of the thesis, I created a simulation using Unreal Engine with blueprints. Blueprints is a visual scripting system that allows you to create game logic without writing code. I set up different objects and characters and made them interact with each other based on the rules I set at the beginning of the project.

Graph 13: Boxplots of all 3 scenarios



Source: Author (2024)

In this graph 13, there are six boxplots shown. Each boxplot represents one of the six groups of participants being studied. The colour of the boxplots corresponds to a specific scenario. The final amounts in the groups come from independent random samples. The horizontal lines inside each boxplot show the average value for that sample.

I concluded that, it's important to examine whether the average values of these random samples are statistically different or not. From graph 13, we can see that the average values are generally similar, except for the orange boxplot, which represents the buyers. It is likely that the average values of all six random samples are similar.

And lastly, the focus was on discussing the flaws, problems and highlighting both the challenges faced and the practical skills I obtained during the creation of the project. I believe the application can be used in a real-life scenario for entertainment purposes to show others what can be done inside the game engine Unreal Engine. It could also motivate beginners in game development to see what is possible inside this game engine.

I. Summary and keywords

Students in economics face the challenge of comprehending the fundamental theory of microeconomics about the relationship between supply and demand in a market setting. To address this challenge this work uses Unreal Engine as it is a powerful and versatile tool.

This bachelor thesis is focused on replicating the economic model of supply and demand in a free market simulating it in real time using the game engine Unreal Engine 5 utilizing the engine capabilities to create an engaging immersive world.

The game is set to have options to freely change the number of buyers, sellers. The resulting game's purpose is to help students have a better understanding of the economic concept through dynamic simulations inside a three-dimensional world.

Key words: supply and demand, unreal engine, simulation, game development

II. List of used sources

Alzubi, T., Fernández, R., Flores, J., Duran, M., & Cotos, J. M. (2018). *Improving the Working Memory During Early Childhood Education Through the Use of an Interactive Gesture Game-Based Learning Approach*. IEEE Access, 6, 53998-54009. https://doi.org/10.1109/ACCESS.2018.2870575

Arias, M. (2014). *Using video games in education*. Journal of Mason Graduate Research, 1(2), 49-69. https://doi.org/10.13021/G8jmgr.v1i2.416

Bryan, J. (2018). Excuse Me, Do You Have a Moment to Talk About Version Control? *The American Statistician, 72*(1), 20–27. http://www.jstor.org/stable/45118524

Clayton, W. L. (1946). *The Importance of International Economic Relations to World Peace.* Proceedings of the Academy of Political Science, 22(1), 96–107. https://doi.org/10.2307/1172925

Dealessandri, M., & Calvin, A. (2023a, November 23). *What is the best game engine? Is Unity right for you?* GamesIndustry.biz. Retrieved May 24, 2024. Retrieved from https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you

Dealessandri, M., & Calvin, A. (2023b, November 21). *What is the best game engine? Is Unreal Engine right for you?* GamesIndustry.biz. Retrieved May 24, 2024. Retrieved from https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unreal-engine-4-the-right-game-engine-for-you#section-3

Dealessandri, M., & Calvin, A. (2024, February 12). *What is the best game engine? Is Godot right for you?* GamesIndustry.biz. Retrieved May 24, 2024. Retrieved from https://www.gamesindustry.biz/what-is-the-best-game-engine-is-godot-right-for-you

Dorn, D. S. (1989). *Simulation Games: One More Tool on the Pedagogical Shelf. Teaching Sociology*, 17(1), 1–18. https://doi.org/10.2307/1317920

Epic Games. (2024a). *Unreal Engine End User License Agreement*. Retrieved May 25, 2024. Retrieved from https://www.unrealengine.com/en-US/license

Epic Games. (2024b). *Unreal Engine programming and Scripting*. Retrieved May 27. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-programming-and-scripting?application_version=5.2

Geraci, R. M., Recine, N., & Fox, S. (2016). *Grotesque Gaming: The Monstrous in Online Worlds.* Preternature: Critical and Historical Studies on the Preternatural, 5(2), 213–236. https://doi.org/10.5325/preternature.5.2.0213

Griffiths, M. D. (2002). *The educational benefits of videogames*. Education and health, 20(3), 47-51.

Haman, J. T., & Miller, C. G. (2022). *Introduction to Git.* Institute for Defense Analyses. http://www.jstor.org/stable/resrep40583

History.com Editors. (2022, October 17). *Video game history*. HISTORY. A&E Television Networks. Retrieved May 20, 2024, from https://www.history.com/topics/inventions/history-of-video-games

Lahtinen, S. J. T. (2016). *Utilization of Game Engine in Simulation Visualization* (Master's thesis).

Pyndick, R. S., & Rubinfeld, D. L. (2018). *Microeconomics 9th Edition*. Pearson.

Solomon, S. L. (1980). *Building Modelers: Teaching the Art of Simulation.* Interfaces, 10(2), 65–72. http://www.jstor.org/stable/25059885

Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Pearson Education, Inc.

Unity Technologies. (2024) *Unity Visual Scripting*. Retrieved May 25, 2024. Retrieved from https://unity.com/features/unity-visual-scripting

VANAS. (2023, February 1). *Top 5 video game software engines*. MyFame. Retrieved May 22, 2024. Retrieved from https://www.myfame.org/blog/top-5-video-game-software-engines

Värtinen, S., Hämäläinen, P., & Guckelsberger, C. (2024). *Generating role-playing game quests with GPT language models*. IEEE Transactions on Games, 16(1), 127-139. doi:10.1109/TG.2022.3228480.

Whelan, J., Msefer, K., & Chung, C. V. (2001). *Economic supply & demand* (Vol. 520). Cambridge, MA: MIT.

Wolf, M. J. P. (2011). MYST AND THE ADVENTURE GAME GENRE. *In Myst and Riven: The World of the D'ni* (pp. 7–21). University of Michigan Press. https://doi.org/10.2307/j.ctv65sx38.6

III. List of figures

IV. List of Blueprint visual script

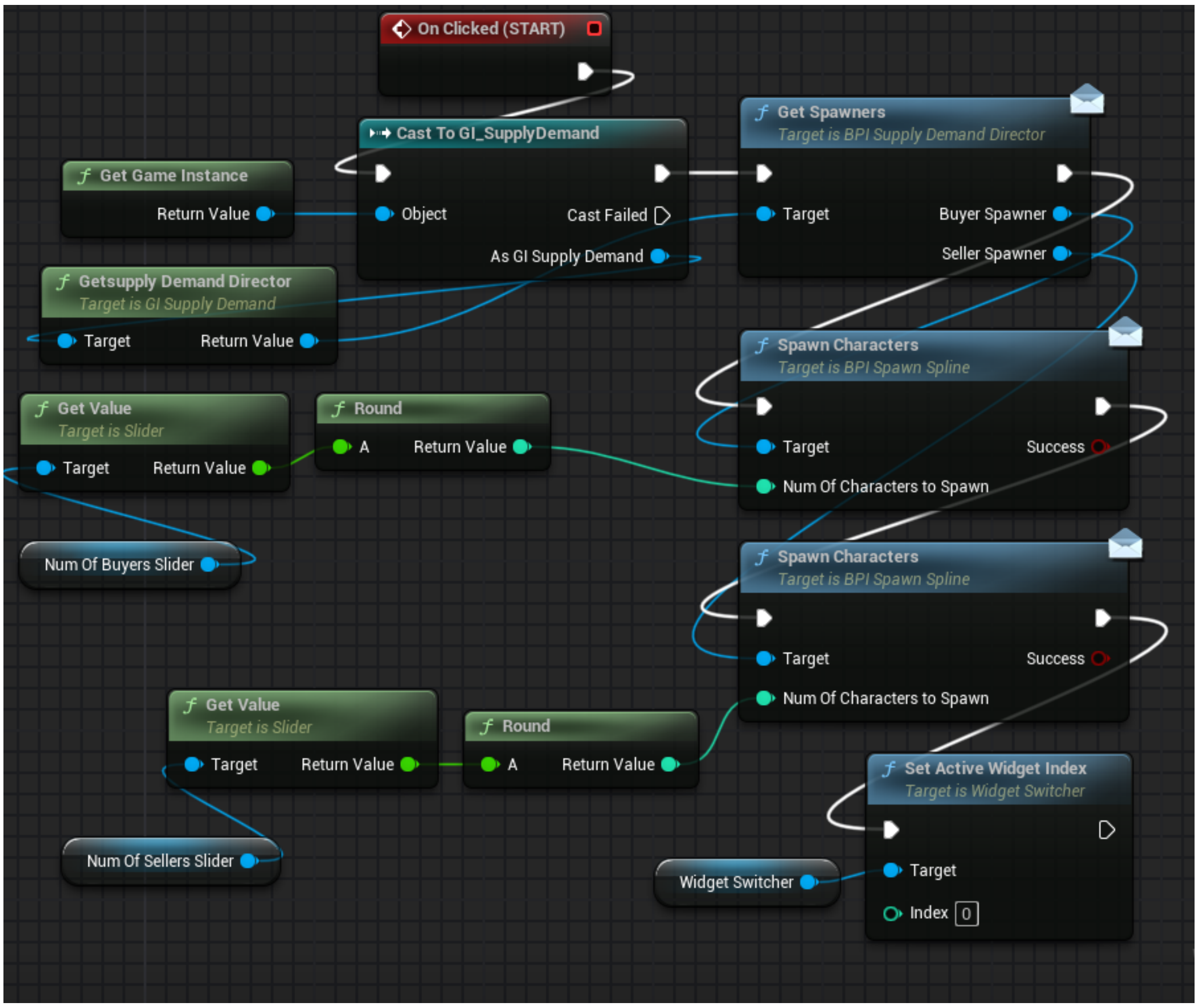
V. List of source code

VI. List of graphs

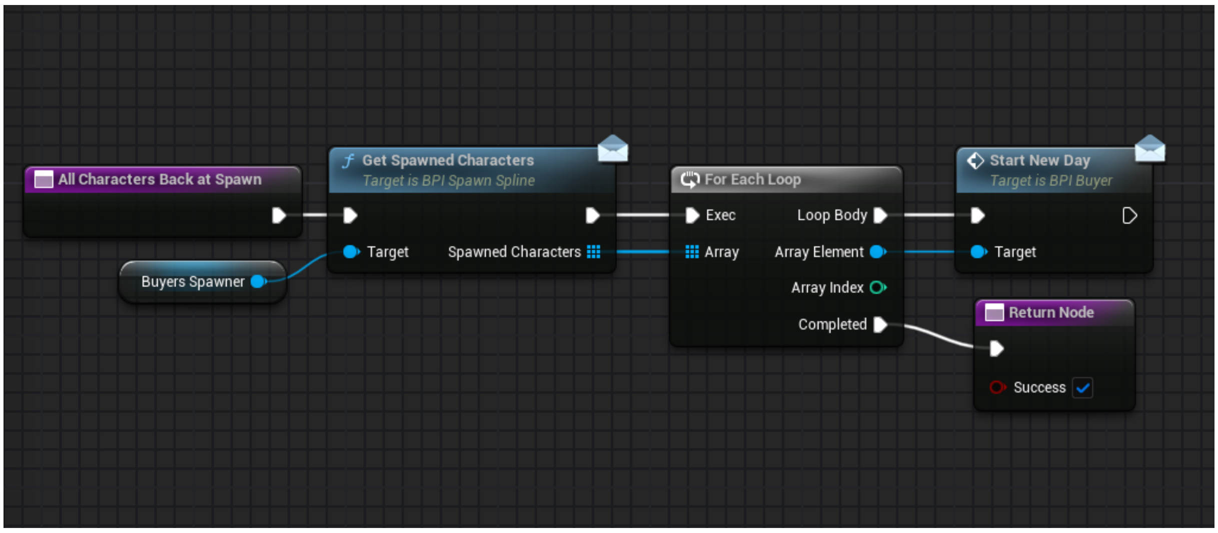
VII. List of Attachments

VIII.Attachments

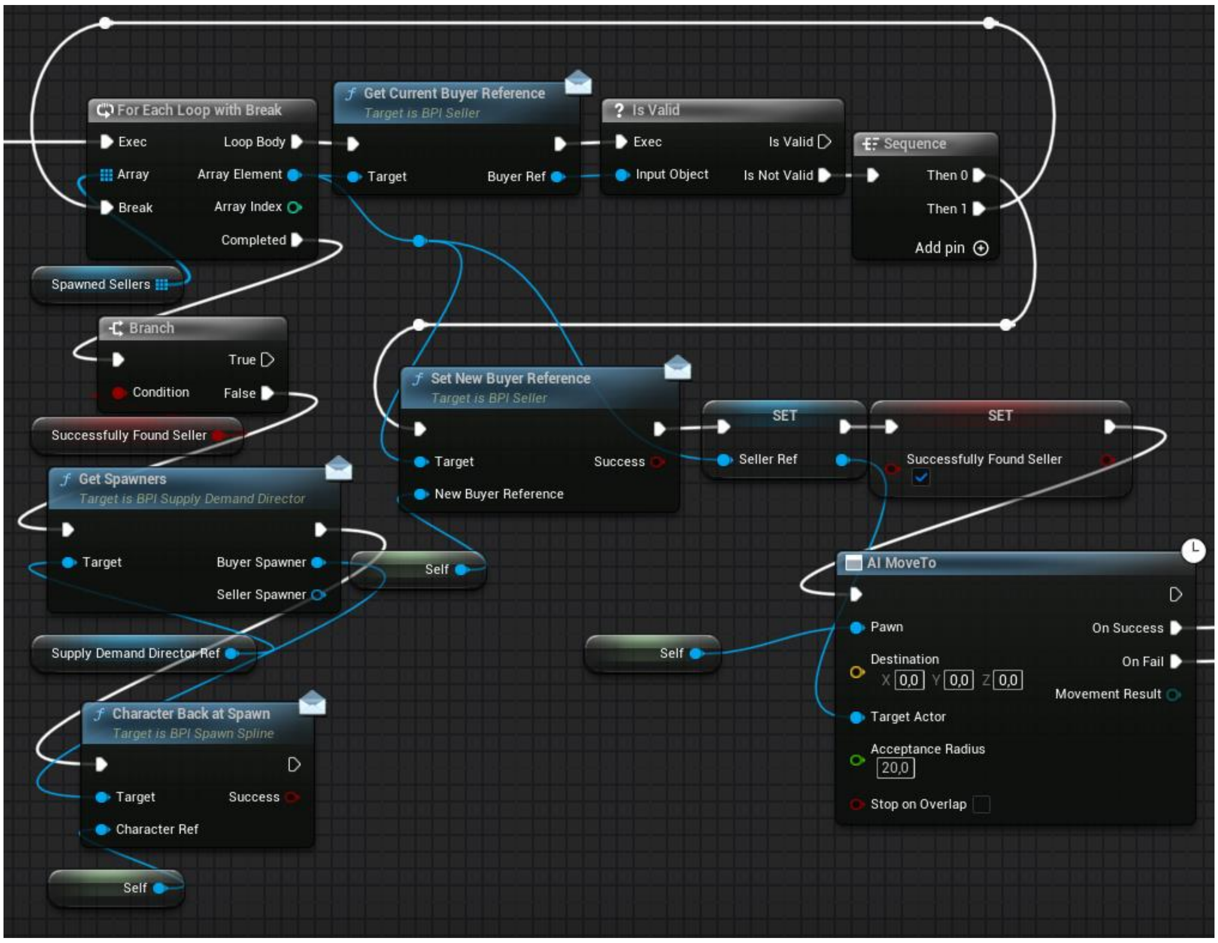
Attachment 1: Start Button Implementation (Author)



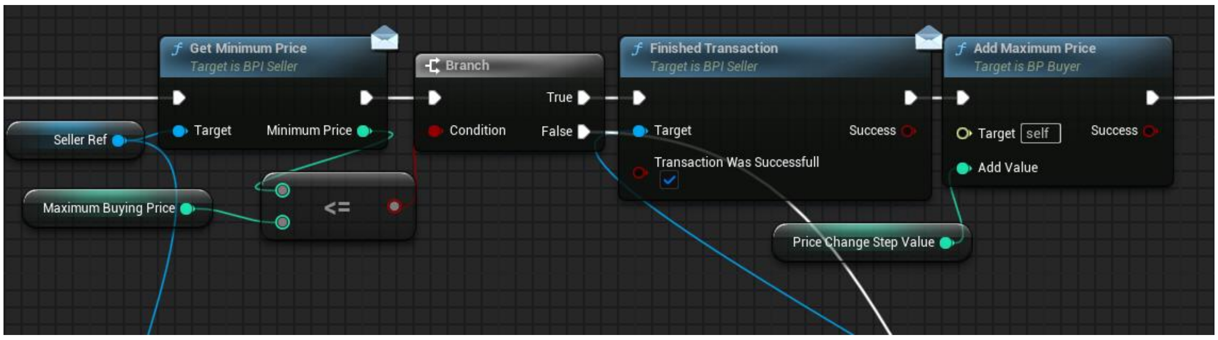
Attachment 2: Starting a new day (Author)



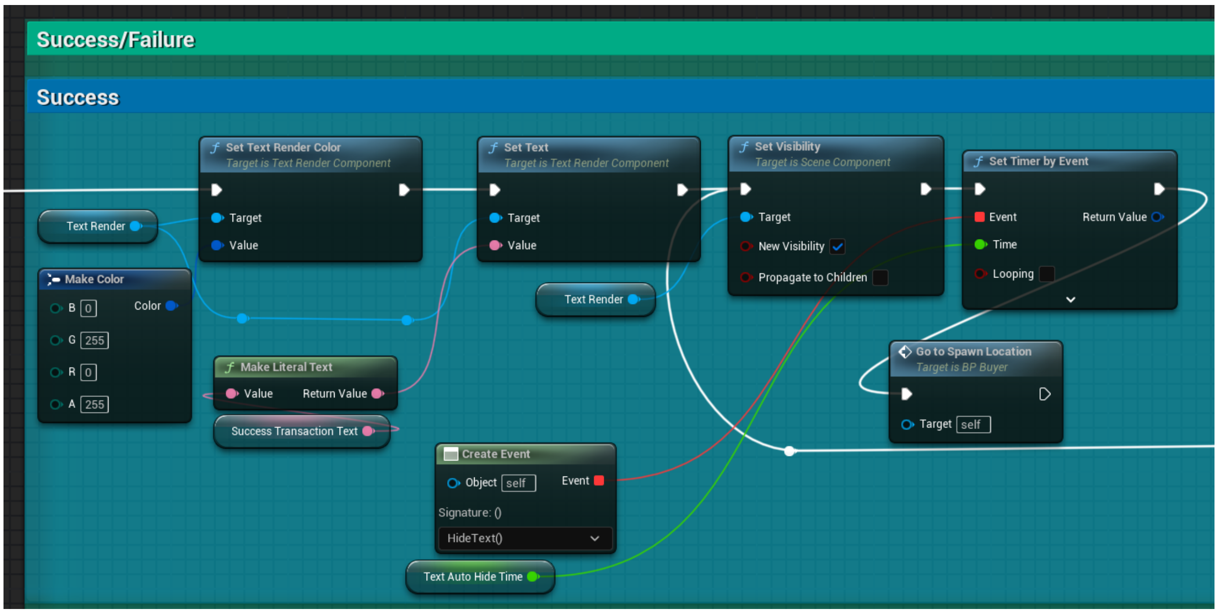
Attachment 3: Buyer finding Seller (Author)



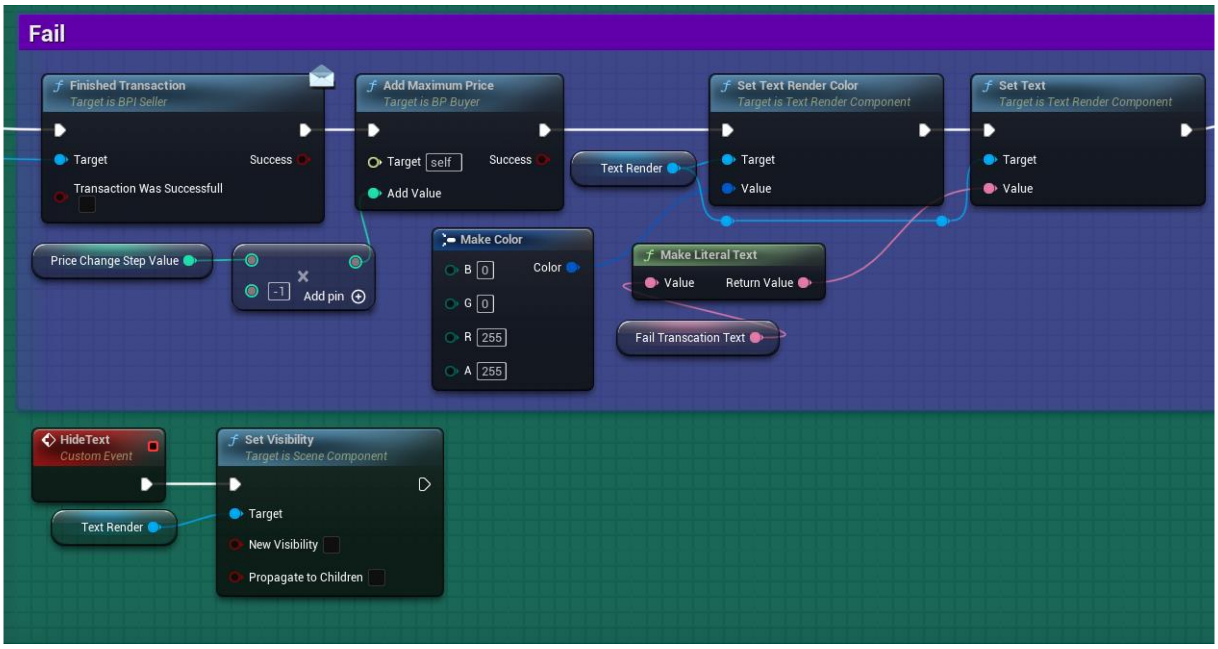
Attachment 4: Transaction (Author)



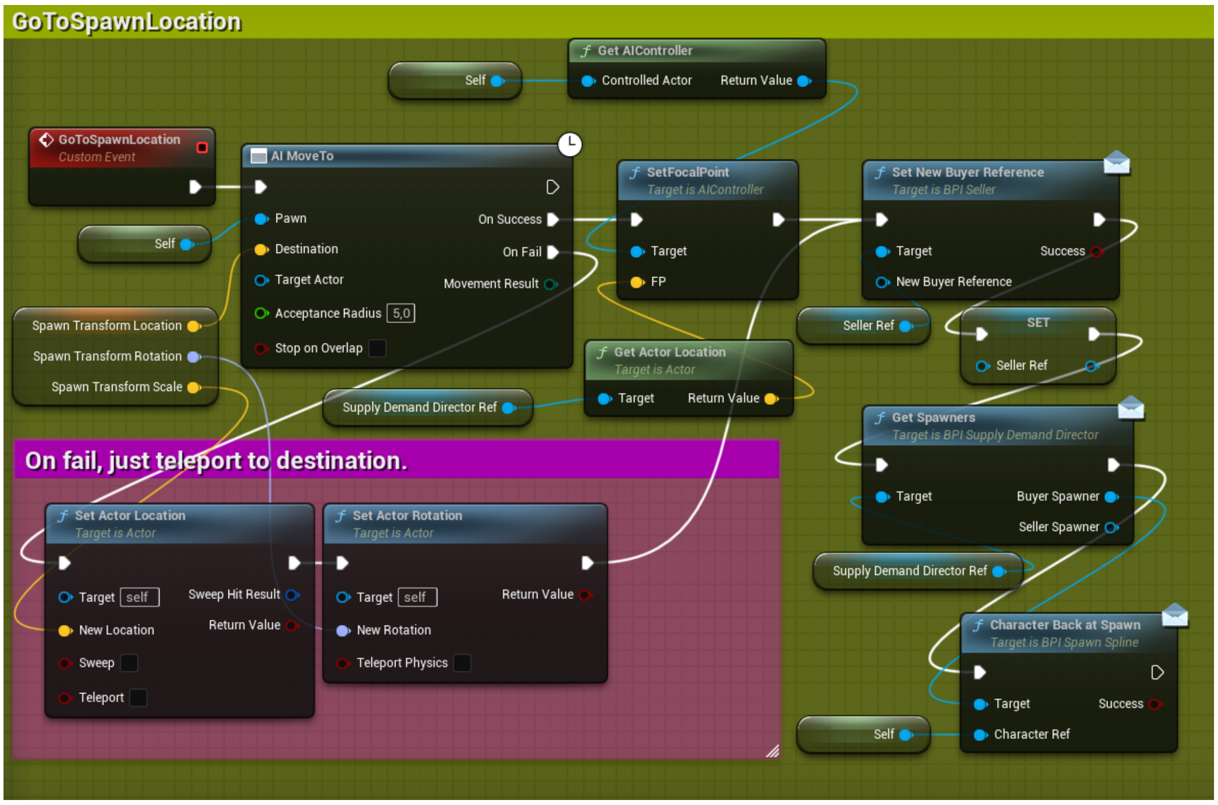
Attachment 5: Successful Emoji showcase (Author)



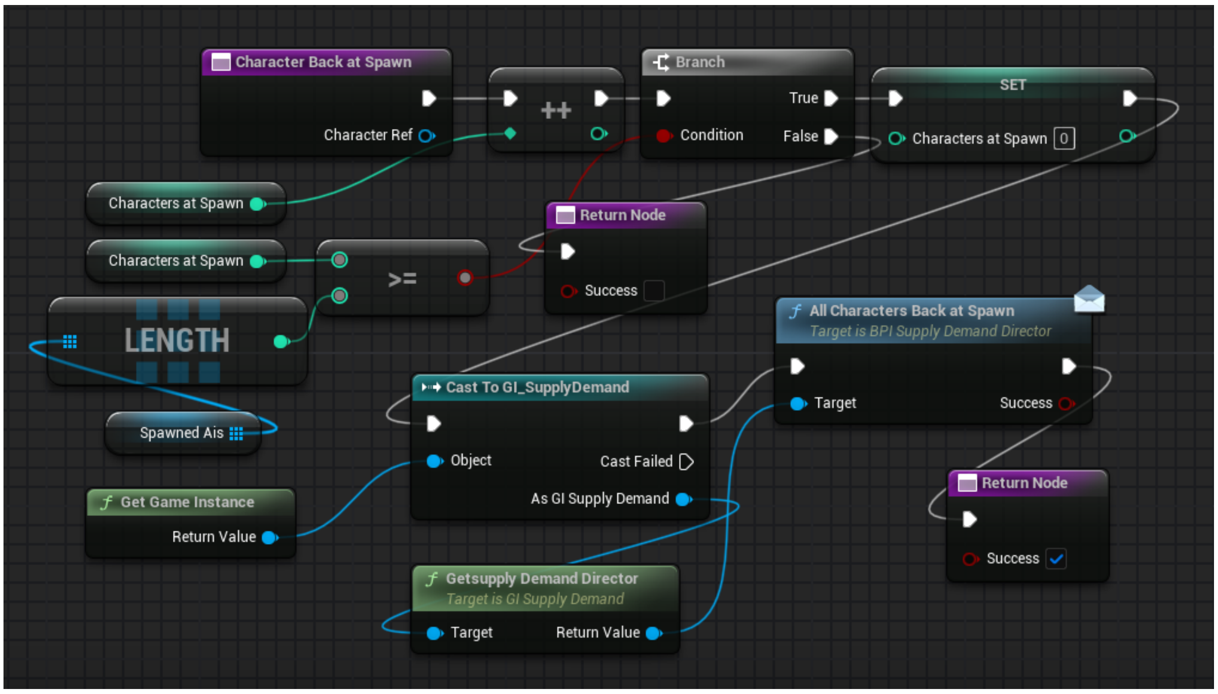
Attachment 6: Unsuccessful emoji showcase (Author)



Attachment 7: Go to Spawn Location Custom event (Author)



Attachment 8: Characters Back at Spawn (Author)


Character Back at Spawn
Character Ref
++
Branch
Condition
True
False
SET
Characters at Spawn 0
Characters at Spawn
Characters at Spawn
>=
Return Node
Success
LENGTH
Spawned Ais
All Characters Back at Spawn
Target is BPI Supply Demand Director
Target
Success
Cast To GI_SupplyDemand
Object
Cast Failed
As GI Supply Demand
Get Game Instance
Return Value
Getsupply Demand Director
Target is GI Supply Demand
Target
Return Value
Return Node
Success