Kristófer Ívar Knutsen

Visual Scripting in Game Development

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

10 May 2021

Abstract

Author:	Kristófer Knutsen
Title:	Visual Scripting in Game Development
Number of Pages:	41 pages + 3 appendices
Date:	10 May 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisor:	Ulla Sederlöf, Principal Lecturer

This thesis explores how visual scripting is used in game development, both in practice and in theory. The topic starts on a broad level but is narrowed down to the two most popular game engines to date, the Unreal Engine and the Unity Engine. Next, the theoretical part covers how the selected game engines have evolved from their infancy and when visual scripting became a part of their respective platform. Finally, the visual scripting tools were used to create mini-games in each engine, and the process was compared and analyzed in the practical part.

The hypothesis that visual scripting is inferior to traditional written code was tested by research and in practice. The main emphasis was to look at the visual scripting offerings for the Unreal Engine and the Unity Engine, compare them to their scripting language counterpart and then compare them against each other.

The results of both the theoretical and the practical part show that the visual scripting tools are an excellent asset for anyone who wants to use their many benefits, such as quick iteration, fast prototyping, and increased productivity between programmers and designers. However, it also demonstrates some aspects of the game development process that should be taken into consideration while using visual scripting due to some limitations that the tool can show.

Keywords: Unreal, Unity, visual scripting, game development, Blueprint, Bolt.

Contents

List of Abbreviations

AI:	Artificial Intelligence
API:	Application Programming Interface
COM:	Component Object Model
CPU:	Central Processing Unit
FPS:	First Person Shooter
GPL:	General Public License
GRAIL:	GRAphical Input Language
IDE:	Integrated development environment
LTS:	Long Term Support
MIT:	Massachusetts Institute of Technology
NPC:	Non-Playable Character
OTEE:	Over the Edge Entertainment
PEGI:	Pan European Game Information
PS5:	PlayStation 5
UE3:	Unreal Engine 3
UE4:	Unreal Engine 4

VR: Virtual Reality

VS: Visual Scripting

1 Introduction

The video game scene is continually changing, and its development with it. Tools and software are updated continuously, which makes their selection often cumbersome. This report looks at specific game engines, what they are, how they work, and what they can produce. These reflections are essential for entry-level users selecting their tools for the first time and advanced developers expanding their horizons with different development techniques.

The goal of this project is to look at visual scripting (VS) when it comes to game development in a theoretical and practical manner. The focus is narrowed to the visual scripting option Bolt for Unity and Blueprint for Unreal Engine 4 (UE4). As a reference, this study uses Unity version 2020.3.1f1 Long Term Support (LTS) and the latest stable one, 2021.1.0f1. [1.] As for the Unreal Engine, version 4.26.1 is used for direct comparison to Unity [2]. The tools are compared as delivered per each game engine version. The plans for the tool's development are discussed but are not used in the comparison.

Other engines such as CryEngine, Godot and, GameMaker Studio 2 all have visual scripting options but, the scope for this project is limited to Unreal Engine and Unity [3–5]. Visual scripting is not only found in game development. Although outside of the scope, other sectors such as multimedia, automation, and simulation heavily rely on visually scriptable systems for their operations.

Other paid-for visual tools for Unity such as Playmaker, FlowCanvas, and NodeCanvas exist, but they are neither free nor a core feature of the game engine, so they are out of the scope of this project [6,7]. Coding concepts, conventions, and other high-level methods will be demonstrated but not explained in detail.

The following chapter covers how game development integrates and uses visual scripting. Furthermore, understanding where visual scripting comes from and how it has evolved is a good indication of where it is going; therefore, it is covered in the following part.

The chapter following the history of visual scripting covers the game engines in more depth. It goes through the background of the engine, its practical use, and its visual scripting tools. The chapter ends with a demonstration of products made with each engine, highlighting certain capabilities of each engine. Lastly, the final chapters are dedicated to a real-world practical example of visual scripting being used in both engines and ending with reflections and conclusion of the study.

2 Research Background: Visual Scripting in Game Development

Learning programming can be a daunting task. It can often take years to master a programming language and to be able to use it in a professional environment. That can often be highly intimidating for people and put them off from ever looking at game development. Not everyone is looking to be a game programmer, but they still want to make games. Game artists and designers would want to change the game logic and mechanics without going through a programmer each time. That is where visual scripting comes in.

As the name suggests, visual scripting is programming without having to write code. Instead, the user constructs node-based graphs where each node corresponds to a code block and then linked to other nodes, as demonstrated in Figure 1. That creates a flow graph that can be easily read and altered by users who do not have extensive programming knowledge.

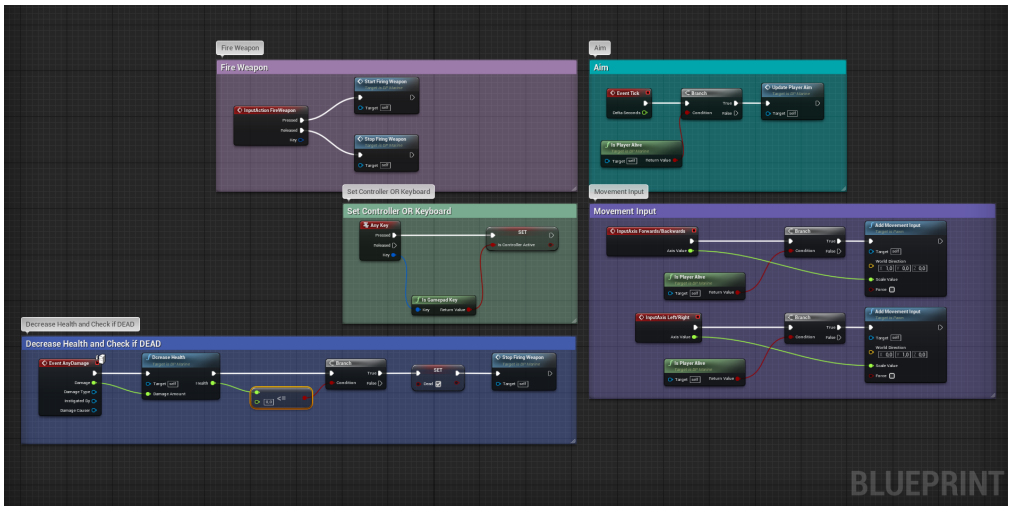


Figure 1. A Blueprint from Unreal Engine [8]. Screenshot by author.

From a logical perspective, visual scripting can be thought of as a tool, not unlike a hammer. If the said tool can reduce time, effort and allow other users to access a previously inaccessible platform due to lack of knowledge, it would logically make sense to utilize it. That is exactly why it was created for game development. Quick iteration, fast prototyping, and increased collaboration

between people make pretty great arguments for giving visual scripting a second look. [9.]

The purpose of this study is to investigate how visual scripting has evolved throughout history in different areas and see how the gaming sector has implemented that practice. One objective will be to look at visual scripting without prejudice or preference to other coding conventions and focus on the tool itself and how it can help users with different tasks.

Other aspects of game development that currently use node-based solutions, like materials, shaders, and animations, will be touched on briefly to showcase the widespread usage of this technology. However, it will not be focused on like the visual scripting for programming purposes will be.

The main question is how visual scripting works in game development for the Unreal Engine and the Unity game engine, both in practice and theory. Those topics will be explained and expanded on in the following chapters.

3 History of Visual Scripting

Visual scripting is constantly gaining popularity, but it is hardly anything new. Visual scripting has been around almost as long as the graphical interfaces have been able to support it. Dating back to 1968, GRAIL (GRAphical Input Language) was a flowchart-based language that used stylus input from the user to create operations and flow control. Although GRAIL's creation was over 50 years ago, it still bears some resemblance to the flow structures of today's visual scripting tools, seen in Figure 2. [10.]

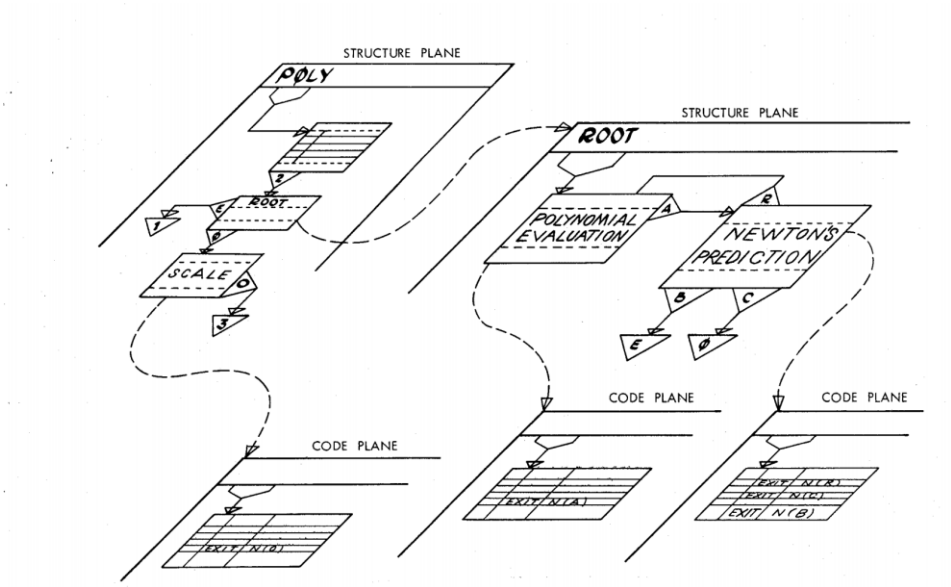


Figure 2. Flow structure from GRAIL Input language [10].

In March of 1991, Microsoft launched Visual Basic, a drag-and-drop Component Object Model (COM). This software was a game-changer for many people and opened the way to application programming in Windows to a broader audience. Visual Basic had numerous updates throughout the years, adding support for web development in 1998. In 2008, Microsoft ended its extended support for Visual Basic, although the base development environment still runs in all 32-bit Windows systems. [11.]

In May of 2007, MIT Media Lab released its first version of Scratch, a free web-based visual programming language aimed at young people from 8 to 16. Scratch features a colorful drag-and-drop environment where the user creates the flow of operations by vertically stacking code blocks. Fourteen years later, in March of 2021, the 500.000.000th project was shared, marking an astounding success for the platform and visual scripting in general [12].

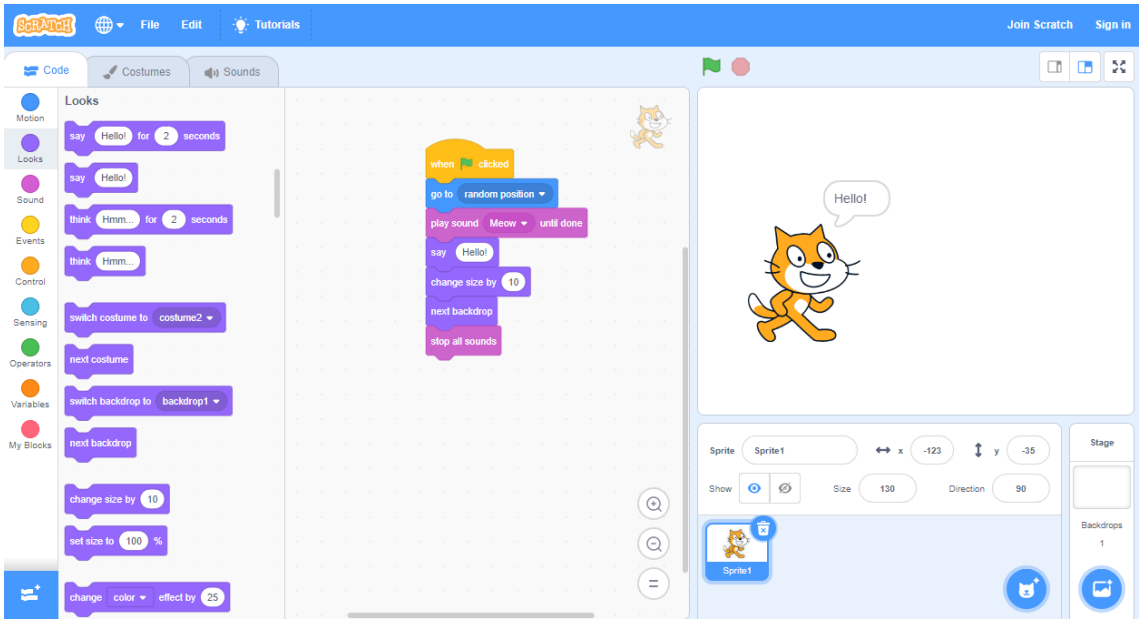


Figure 3. Scratch visual programming [13]. Screenshot by author.

A broad look over the gaming industry reveals a pattern that indicates where the tool development is going and what the consumer expects regarding the working environment. 3D modeling programs, game engines, and other platforms such as BabylonJS use node-based editors for their material, shaders, and particle creation. [14]. Seeing how popular the node-based scripting is on other platforms, it is not difficult to imagine why the game industry has widely adopted that to its code creation.

4 Game Engines

An Integrated Development Environment (IDE) specifically crafted to make video games is probably the simplest way to describe a game engine. Although the explanation is simple, the engine itself is an intricate work of engineering. A barebone game engine only consists of two elements: a script editor for code writing and a compiler to run the code. Although this is technically enough to create an engine, this will not accomplish any incredible feats for game making because more tools are needed to create a good game making experience. [15.]

Modern game engines far exceed the notion of just being a compiler and a script editor. Taking the Unreal Engine, for example, it is a complete real-time 3D creation platform. It is used not only for games but architecture, films, simulations, transportation, and so much more. The engine handles physics calculations, rendering, particle creation, and complete platform support, whether PC, console, or in extended reality. [8.] These features, alongside numerous others, make this not just a game engine but a comprehensive platform for creators all over the industry.

Not all game engines are equal in their capability nor availability. Some game engines like Godot and OGRE are open source under the Massachusetts Institute of Technology (MIT) license, available to use by anyone for free. [16,17]. Other engines like the featured Unity and Unreal Engine are free for use; however, they come with a subscription model for multiple users, premium version, profit percentage cut, or royalties. [18,19]. Valve, 4A games, and id Software have created their proprietary engines that are only used in-house, circumventing all licensing and fee issues with other engines.

4.1 Licencing

Picking a game engine and assets to create a project can involve many factors. One of those factors has to do with licensing. A software license comes in

various forms, and each license caters to a specific user type. Depending on the nature of the project, whether it is open or closed source, careful consideration must be made regarding licensing.

MIT

The MIT license is one of the most liberal licenses you can use. It gives many permissions, such as commercial use, distribution, modification, and private use. Anything with an MIT license can be used freely by anyone in any project; however, there is no warranty or liability given with an MIT product. [20.]

GPL

The foundation of the General Public Licence (GPL) is freedom. This license ensures that any product produced with GPL software is free to use and free to distribute. The usage of a GPL product is free; however, it binds the user to keep it free. Any changes, usages, or redistributions to the product must remain free. The act of ensuring the product is open source is referred to as copyleft, the guarantee of user freedom. [21.]

Apache 2.0

The Apache 2.0 license is similar to the MIT license in many ways. It is very liberal, but it has some features that make it more intricate than the MIT one. The usage of the product is free, publicly and privately. It can be sublicensed, patented, and re-distributed at will. However, any modifications to the product by the user must be documented and made available with the product. [22.]

4.2 Unity Engine

4.2.1 History

Unity Technologies was founded in Denmark in 2004 by Nicholas Francis, Joachim Ante, and David Helgason [23]. Unity is the world's leading platform in creating real-time 3D content. In 2020, over 50% of all games, regardless of platform, were made with Unity. [24.]

Unity Technologies got its name in 2007. Before that, it was called Over the Edge Entertainment (OTEE) and was a Mac-only concept. The idea was to create an engine to make more advanced games because game making was always the goal. After blowing past their self-imposed deadlines, out came a game-making tool. [23.]

In June of 2005, Unity released version 1.0. The first version only supported the Mac operation system. Windows and web browsers did not get build support until version 1.1. Two years later, during the Unite Developer Conference in 2007, version 2.0 was released. Although version 1.1 came with Windows build support, it did not support the editor itself. It was not until 2009, when version 2.5 came out, where the first Windows support for the editor was released. [25 pp 8.]

In 2017, Unity changed its version naming convention. Version 5.6.7 was the last one released with the old convention. The first version of the new convention was 2017.1.0. That still follows the semantic versioning strategy, which means that the first number represents a major change, the second one is a minor change, and the last represents a patch. That works well for Unity since they release one major version change a year, so it corresponds to the current year. [26.]

Unity has created a roadmap for the future regarding what is in development and what is improving. Those include a vast number of things, including 3D world-building, multiplayer networking, and augmented and virtual reality. One

of the aspects that are on the roadmap is the gameplay and user interface design. That category includes all the features in development for visual scripting. A concurrent theme throughout the roadmap is increasing performance, accessibility, and overall optimization for tools and the engine itself. [27.]

4.2.2 Practical Use

Unity is an excellent platform for many game developers to start their game development journey. Many features make the Unity platform stand out as a viable choice for project selection. One of the most prominent features that make the platform so alluring is that it is free for individuals, as long as the revenue or funding does not exceed $100,000 per fiscal year [28]. The subscription plans for teams range quite a bit depending on the size of the team and what features are requested, but all plans, individual or teams, are royalty-free.

The size of Unity's market share translates to a vast community that offers an abundance of help and support from a growing population of Unity developers. This extensive community and user base also attract other people to create videos, tutorials, and whole courses around the Unity platform, resulting in a positive feedback loop that only attracts more users.

The programming language is also a factor. Unity's scripting language is C#, which is an object-oriented programming language. In the effort of making Unity more beginner-friendly, an announcement in July of 2020 stated that all future Unity plans would include the visual scripting tool Bolt. Bolt allows the implementation of logic without any coding knowledge, which opens the platform to a whole other audience. [29.]

One prominent feature that Unity has is its asset store. The asset store is a marketplace where users can find tools, art, sounds, and everything in between. These products are sold both by users and Unity itself. There is also an

abundance of free assets that can be downloaded and used in any project, but not all assets have the same license, which must be considered. [30.]

4.2.3 Bolt

In July of 2017, Ludiq announced on Unity‘s forum a visual scripting tool in development called Bolt. Later that month, it was released on the Unity asset store. [31.] Finally, four years after its release, Unity and Ludiq announced, in early May of 2020, Unity‘s acquisition of Bolt. That would be the first step in permanently integrating and natively supporting visual scripting into the Unity Engine. [32.]

What makes Bolt a great tool to integrate into a project is its many benefits. The key benefits are the primary creation tool itself, the flow graph. It lets the user create and execute logic using nodes. State graphs are high on the list, creating game states such as artificial intelligence (AI) behaviours or any other structures that require state transition. [33.]

One of the more important things to keep in mind is not to limit Unity’s Engine capability to utilize the C# language and its benefits. That is taken care of using codebase compatibility. The nodes reflect the codebase, which makes them up to date and allows for full access to all available methods available in C#. Another honorable mention is the ability to do quick iteration with live editing. That allows the user to manipulate values and the environment while the game is in play mode. That translates to faster workflow and limits the time spent recompiling changes made in the project. However, this flexibility comes with a performance cost that can add up on larger projects. [34.]

For versions 2020.3 and older, the user must manually add Bolt to the working project as a tool from the asset store. Versions 2021.1 and above have Bolt built-in as a standard feature, although in the newer versions, Unity has dropped the Bolt name in favor of VisualScripting. As Figure 4 shows, the integration is different between the 2020 and 2021 versions. Unity has changed

the Flow Graph name to Script Graph, Bolt and Ludiq installation folders have been removed and now comes ready with a built-in option, generically named, VisualScripting. [34.]

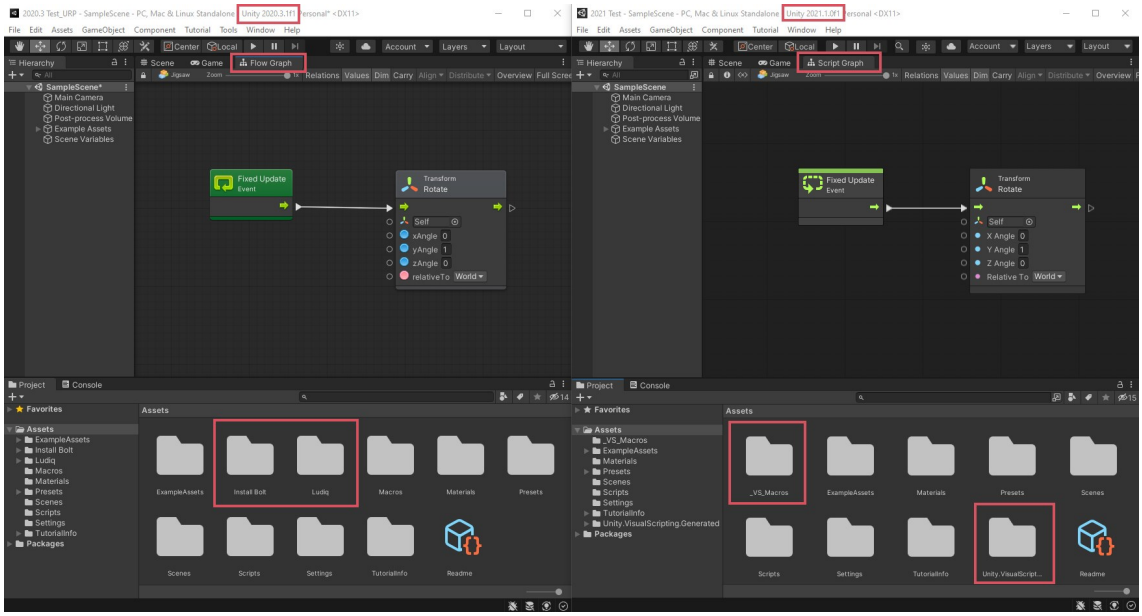


Figure 4. Version 2020.3.1f1 on the left and 2021.1.0f1 on the right [24]. Screenshot by author.

During Unity's acquisition of Bolt, Ludiq was already developing Bolt 2. The second version offered many new features and upgrades and was already beta testing on users. According to Unity's roadmap, they are not releasing Bolt 2 as a tool. As mentioned above, Bolt is now integrated into Unity as visual scripting, and it will be developed as such. It has many features in development, but they are focused on standardizing and consolidating all the visual scripting features to be compatible with all aspects of their engine. [35.]

4.2.4 Games made with Unity

Subnautica

Subnautica is a massive open-world survival game that takes place in an ocean on an alien planet. As Figure 5 demonstrates, the game is both beautiful and

massive. The game's objective is to gather resources from the ocean floor and craft tools, vehicles, and other equipment to survive. [36.]

The game shows the power of the Unity engine and its capacity to create big open-world games. For total immersion, Subnautica is also available in virtual reality (VR), which Unity fully supports. [37.]

Figure 5. Subnautica gameplay screenshot [38].

Escape from Tarkov

When it comes to visual graphics, Escape from Tarkov pushes Unity to the limit. High fidelity graphics combined with hardcore and realistic first-person shooter make this game a great addition to the long list of Unity games. Figure 6 shows just how capable Unity Engine is in producing hyper-realistic graphics.

Figure 6. Escape from Tarkov screenshot [39].

Hearthstone

Hearthstone is a free-to-play strategy card game developed by Blizzard Entertainment, one of the world's biggest gaming companies. The game builds on the characters and lore of their biggest game, World of Warcraft. As shown in Figure 7, the outlook of the game is very distinctive. [40.]

The simplicity of the game, graphics, and cross-platform gameplay make Unity a perfect fit. However, a surprising one since Blizzard often builds their custom game engines for its game development.

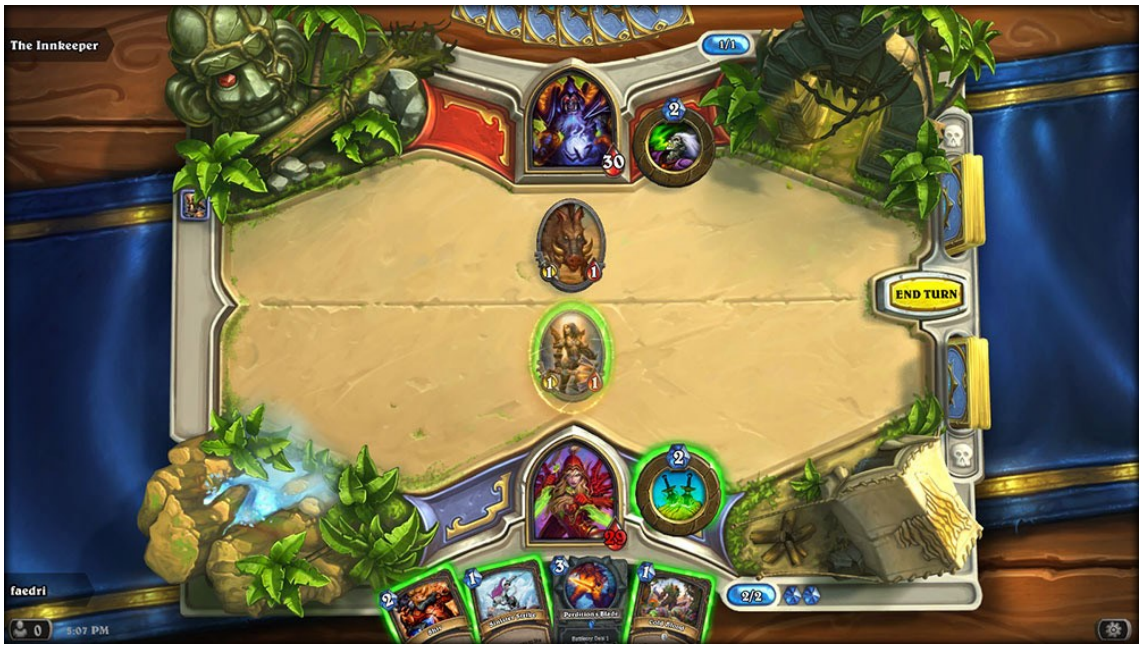


Figure 7. Hearthstone gameplay screenshot [41].

4.3 Unreal Engine

4.3.1 History

The creation of the Unreal Engine started like many other inventions start, with one person in a room trying something different and new. The twenty-one-year-old Tim Sweeney had made his first game called ZZT. ZZT was a simple puzzle game and nothing revolutionary, even by the gaming standards of 1991. However, Sweeney's object-oriented approach to the ZZT's programming sparked the fire's flame that became the Unreal Engine. [42.]

In 1998 the first version of the engine was released alongside a first-person shooter called Unreal, available for Pc and Mac. The game was a big hit and paved the way for the following title, Unreal Tournament. This was a big step in the engine development since it made online multiplayer features available and PlayStation 2 support. [43.]

The second debut of the Unreal Engine was in 2002. This next-generation engine brought significant improvements such as Xbox console compatibility,

particle system, a physics engine, and static mesh tools. Faster internet speeds increased the demand for online multiplayer games. With the launch of the titles Unreal II and Unreal Tournament 2004, many improvements were made in the capabilities of online networking, and with the Xbox compatibility, the 3rd person camera was developed. This significantly opened the engine to a whole new kind of game creation. [43.]

Unreal Engine 3 (UE3) was released in 2006. With this update, the platform support was broadened with support for the PlayStation 3 and Xbox360. A new physics engine was created for UE3; however, one of the most significant updates from the previous version was the introduction to Kismet. Kismet was a visual scripting tool that enabled designers and other users to create game logic and mechanics without writing code. This was a huge step in opening the platform to more users and increasing collaboration between designers, artists, and programmers. [43.]

It was not until eight years later that UE4 was finally released. One of the major changes from UE3 was that the visual scripting tool Kismet would be replaced by a new one called Blueprint. This builds on the idea of what Kismet was and takes it to the next level. With increased flexibility, function, and features, Blueprint allows quick level creation, fast prototyping, and quick iteration. [43.]

In June of 2020, Epic released a demo video showcasing the new features of the unreleased Unreal Engine 5. The demo reveals two new core technologies that will be available with the new engine. Those are Nanite, a virtualized geometry technology that allows for ultra-high polygon assets to be used in a scene without any performance loss. The other one is Lumen, a dynamic global illumination technique that allows for an immediate light change in the scene with infinite light bounces at a vast scale. According to Epic, Unreal Engine 5 will be available in preview in early 2021, although no timeline update has been given since the demo was released in 2020. [44.]

4.3.2 Practical Use

When thinking of the Unreal Engine, the mind automatically wanders to games and game development, but it has so much more to offer. Animations, films, architecture, and hyper-realistic renders are merely a fraction of what the Unreal Engine offers. [8.]

The hyper-successful tv-series The Mandalorian is a perfect example of the non-gaming utilization of the Unreal Engine. The Mandalorian studio sets were surrounded by massive LED walls, which displayed a real-time environment that could be adjusted and manipulated on the fly, seen in Figure 8. This technology enables previously unimaginable creative flexibility, which translates into faster workflow and minimal location shoots. [45.]

Figure 8. The Mandalorian studio set [45].

It is hard to believe that such a powerful tool is free for everyone to use. In June of 2020, an announcement stating that games with under $1 million in gross revenue will remain royalty-free, retroactive to 1 January 2020. After the first $1 million, the royalties increase to 5%. [44.]

As previously stated, the birth of the Unreal Engine started with an object-oriented coding practice. That has not changed. Using C++, Unreal Engine is one of the most powerful game engines to date, with Unreal Engine 5 releasing in 2021 [44].

4.3.3 Blueprint

In the early days of the Unreal Engine, the only way it could be used was with Unreal's proprietary scripting language called UnrealScript. That was the case until 2012 when Tim Sweeney, the founder, and CEO of Epic Games, announced that their scripting language would be dropped for C++, a more flexible and robust scripting language. [46.]

It was only in Unreal Engine 3 when visual scripting became available. The tool was called Kismet and opened the game development accessibility to designers and artists that did not have programming knowledge. That was a great tool, but it had its problems. It was not optimal for reusable scripting behavior, and the visual style was old-fashioned, as seen below. [46.]

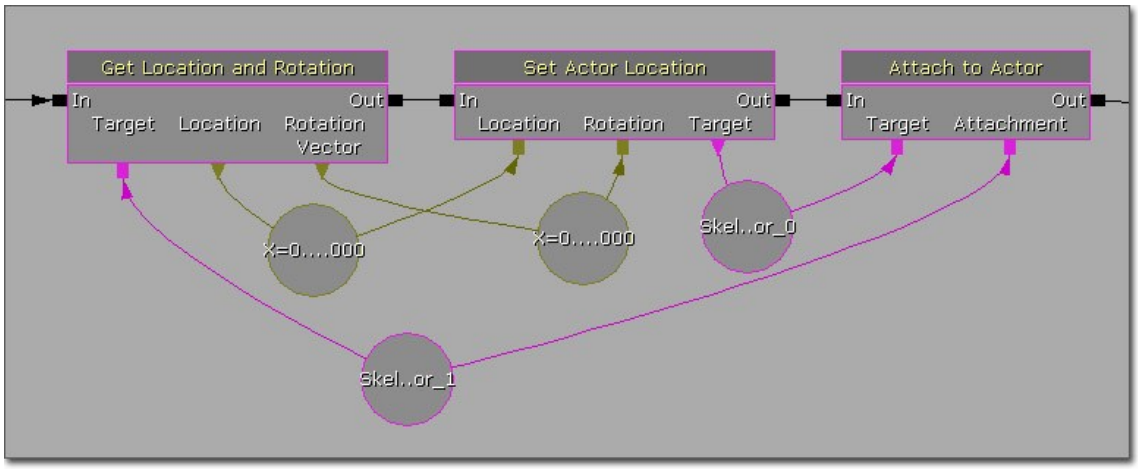


Figure 9. The VS tool Kismet in Unreal Engine 3 [47].

All that Kismet lacked was made up for by the new update in Unreal Engine 4 called Blueprint. Blueprint could create reusable components and quickly became the interface between designers and programmers. What made

Blueprint special was that it was made with the whole engine in mind. All tools that could use node graphs, such as animations and materials, were unified using the same interface and style that Blueprint has. [46.]

A brief overview of a large blueprint quickly shows that the flow pattern can easily grow into an overly complex entity, commonly referred to as a spaghetti monster, as seen below.

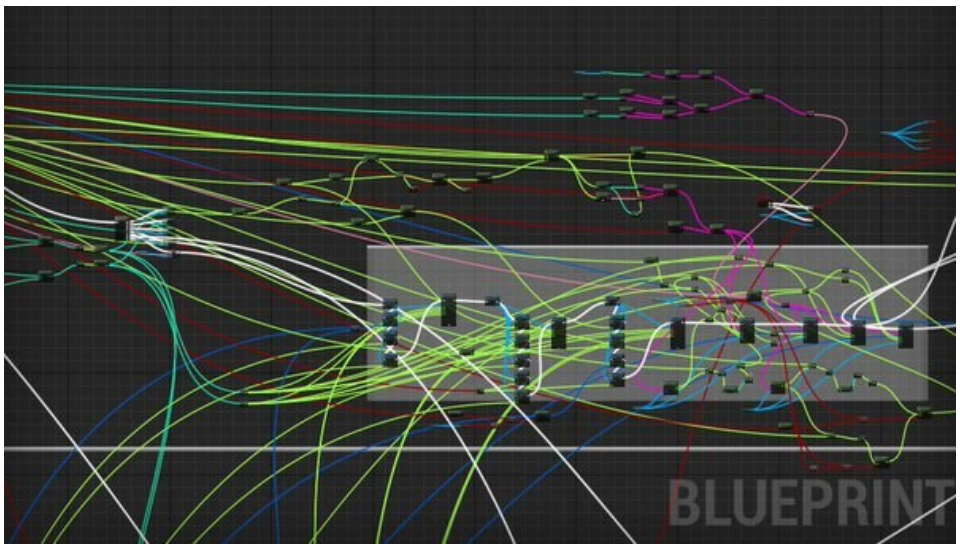


Figure 10. An ill-structured Bluepirnt [48].

Unreal has many tools to help the user create more readable graphs. For example, when a graph's complexity is nearing the user's comprehension, multiple nodes can be collapsed to form a function. That can help de-clutter the graph significantly and is a good practice. Grouping nodes together into colorful sections with explicit comments describing what those nodes do together is also a great feature to have, previously shown in Figure 1. [49.]

4.3.4 Games made with Unreal Engine

Fortnite

In recent years, a trend in game development has manifested, battle-royal games. Multiple players spawn together with the objective is to eliminate each other and be the last one standing. Fortnite is one of the biggest games in that category, with registered accounts exceeding 350million, according to Epic Games [50].

The art style of Fortnite can be described as cartoonish, as Figure 11 demonstrates. The Pan European Game Information (PEGI) rating of the game is twelve, so it attracts mainly the younger crowd. That reflects in the marketing of the game as well.

The success of Fortnite has primarily the Unreal Engine to thank. The vast landscape, customization of characters, and massive multiplayer interaction make Fortnite a perfect example of what Unreal offers concerning game development.

Figure 11. Fortnite screenshot [51].

Borderlands 3

When it comes to having a unique art style, the Borderlands franchise does not fail to deliver. Borderlands 3, the fourth game in the franchise, is powered by Unreal Engine 4. In 2020 Borderlands 3 won three Webby's people choice awards, best art direction, best game design, and best user experience [52]. Figure 12 shows just how intricate and unique the art style is. Combine that with a great storyline and fantastic gameplay; the outcome is a stunning game.

Figure 12. Borderlands 3 Screenshot [53].

Returnal

Returnal is Housemarque's biggest title to date. The game is a PlayStation 5 (PS5) exclusive and features procedural world generation, roguelike gameplay, exploration, and hell-fuelled combat. Released in April 2021, the title pushes the boundaries of the PS5 hardware and the Unreal Engine. According to the former lead game programmer of Returnal, Ari Arnbjörnsson, Blueprint was

heavily used in its development, which allowed for quick iteration and greatly improved cooperation between programmers, designers, and artists [54]. Looking at screenshots, trailers, and gameplay footage gives excellent credibility for the use of Blueprint and visual scripting in general.

Figure 13. Returnal gameplay screenshot [55].

5 Practical Part – Visual Scripting Implementation

5.1 Unity Engine Visual Scripting Implementation

The Unity part of the implementation was done as a part of a development course on Udemy.com, called Bolt Visual Scripting in Unity by Wilmer Lin. That is a highly rated course that goes through all the basics of Bolt. [56.] The main objective is to create a First Person Shooter (FPS) controller without any code. The FPS controller gets all the keyboard and mouse input and translates it into movement. It also handles how the player weapon works and interacts with the environment and enemies. This FPS controller has all the features needed to be implemented into a working game.

Like mentioned previously in chapter 4, Bolt uses flow machines to create the flow of operations and logic. That makes it easy to visualize the order of execution and for debugging purposes. This project uses 14 flow macros and super units. Alongside the flow machine, the game uses a state machine for game management. The state machine is quite simple. It has three states, and in each state, visual scripts are executed. The states are; when the game starts, when the game is running and when the game is over, shown in Figure 14.

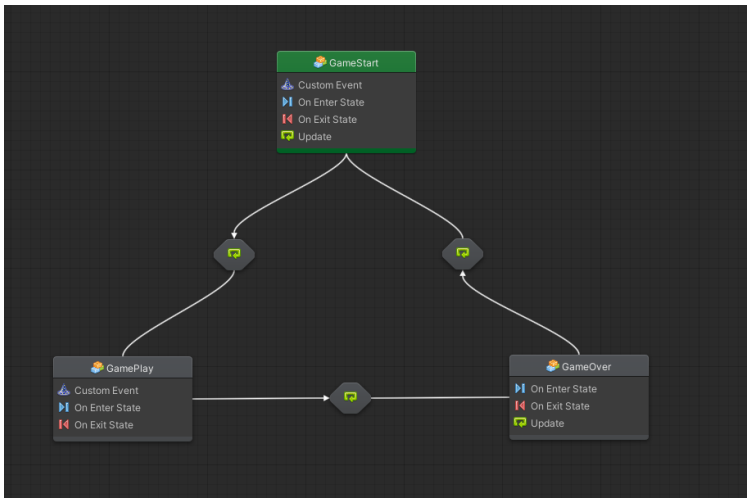


Figure 14. The state machine for the game manager [57]. Screenshot by author.

In the first state of the game manager called GameStart, game variables are reset. The game is paused by stopping the in-game time, game targets are initialized, and the manager waits for the player to hit any key to trigger the transition into the next state.

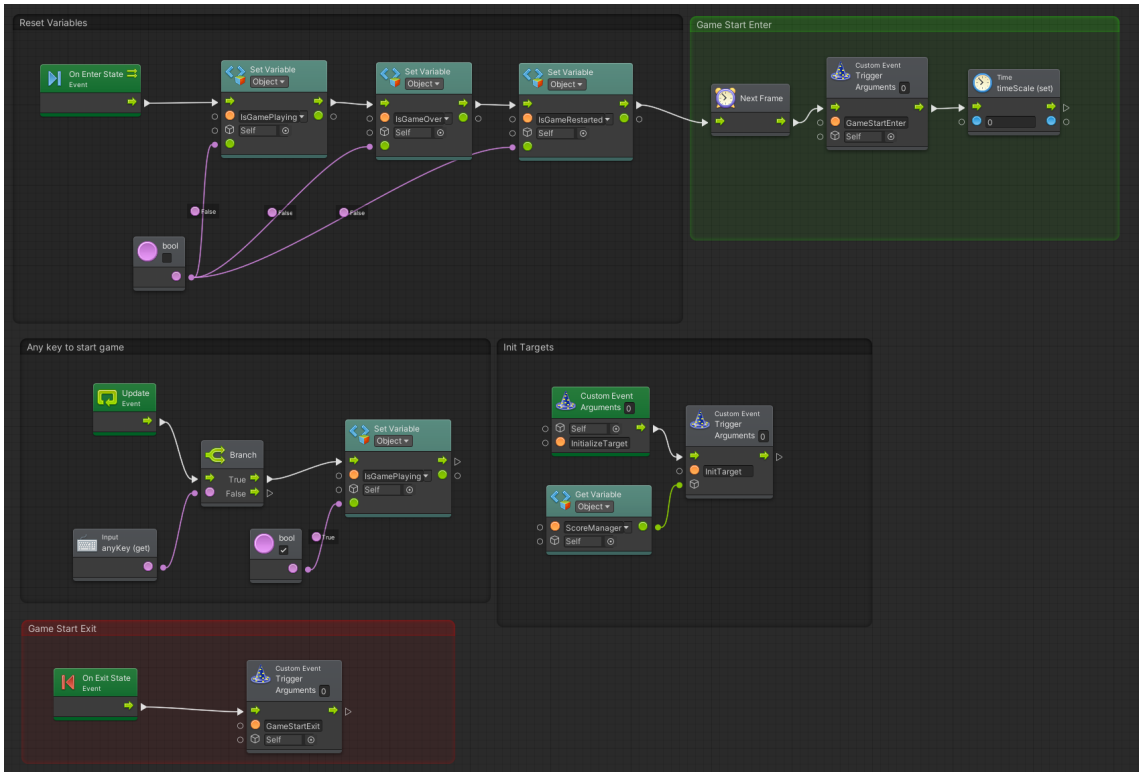


Figure 15. The flowgraph of the start state of the game manager [57]. Screenshot by author.

When the input from the player triggers the next state, the game has been set up, and it is ready to play, so it enters the main state, GamePlay. In the GamePlay state, the game manager communicates with the score manager and the enemy target using custom events. For example, when the enemy target is destroyed, it triggers a custom event that lets the game manager know that one target has been destroyed. When the game manager receives that signal, it triggers another custom event to tell the score manager to add one to the current score of defeated enemies. The score manager also checks if the number of defeated enemies is equal to the number of enemies in the current

level. If the number is equal, it signals the game manager that the game is over and the player has defeated all targets, which is the winning condition.

By using custom events in this pattern, it greatly helps with code encapsulation. All objects only contact the game manager, and the game manager delegates that information as a middleman. Other managers and objects have no direct contact with each other.

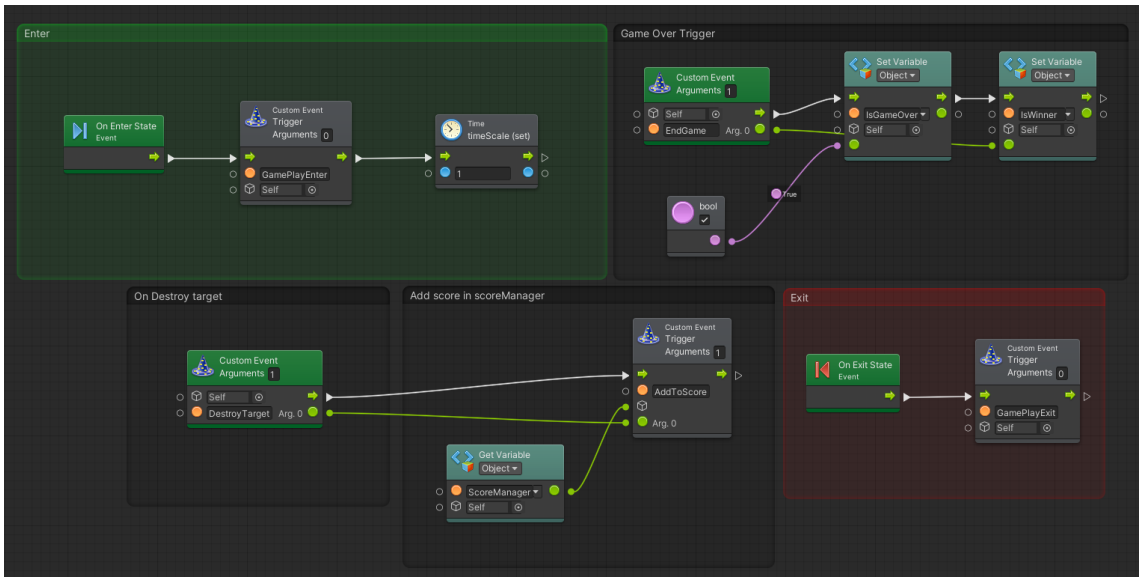


Figure 16. GamePlay state [57]. Screenshot by author.

5.2 Unreal Engine Visual Scripting Implementation

When creating a UE4 project, the user is prompted by a variety of template options. The templates include the controllers for different types of games, giving the user a great starting point for any project. After a template or a blank project is chosen, another option is given to the user, C++ or Blueprint. That allows for the project to be set up for C++ programming or visual scripting in Blueprint. Choosing either option does not exclude the other. Blueprints can still be created in a C++ project, and C++ scripts can be created in a Blueprint project. These options only set up the project folder structure, and if a template is selected, it creates the base character in the project format, C++ or Blueprint.

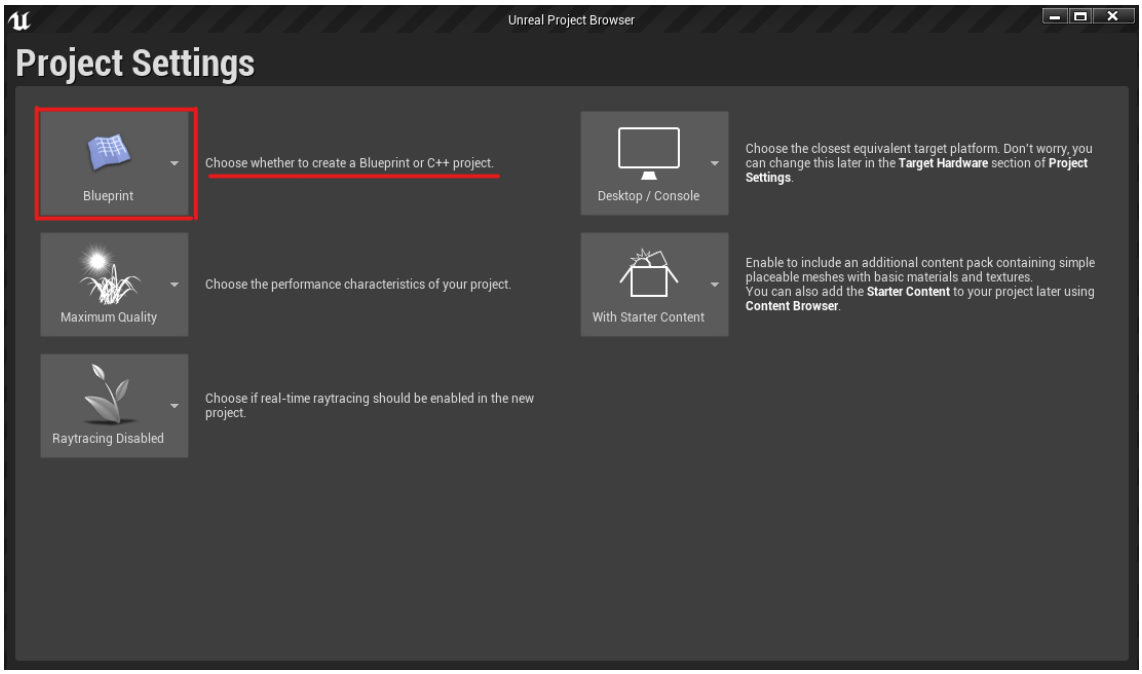


Figure 17. Project settings for new project creation in UE4 [8]. Screenshot by author.

The UE4 practical part started as an empty blueprint project with no template selected. Similarly to the Unity part, the project was done as a part of an online course on Udemy.com. The course name is Unreal Engine Blueprint Game Developer and is created by GameDev.tv in collaboration with the UE4 creator, Epic Games. [58.]

The course is split up into three sections, where a small game is created in each one, increasing in complexity as the user progresses. The game created in section one is called Marble Run. The objective is to roll a ball through a maze using the mouse axis input to tilt the maze in the direction the player wants the ball to move, then gravity and physics take care of the ball movement. The game is mechanically simple, so all the game logic is found in a single-level blueprint.

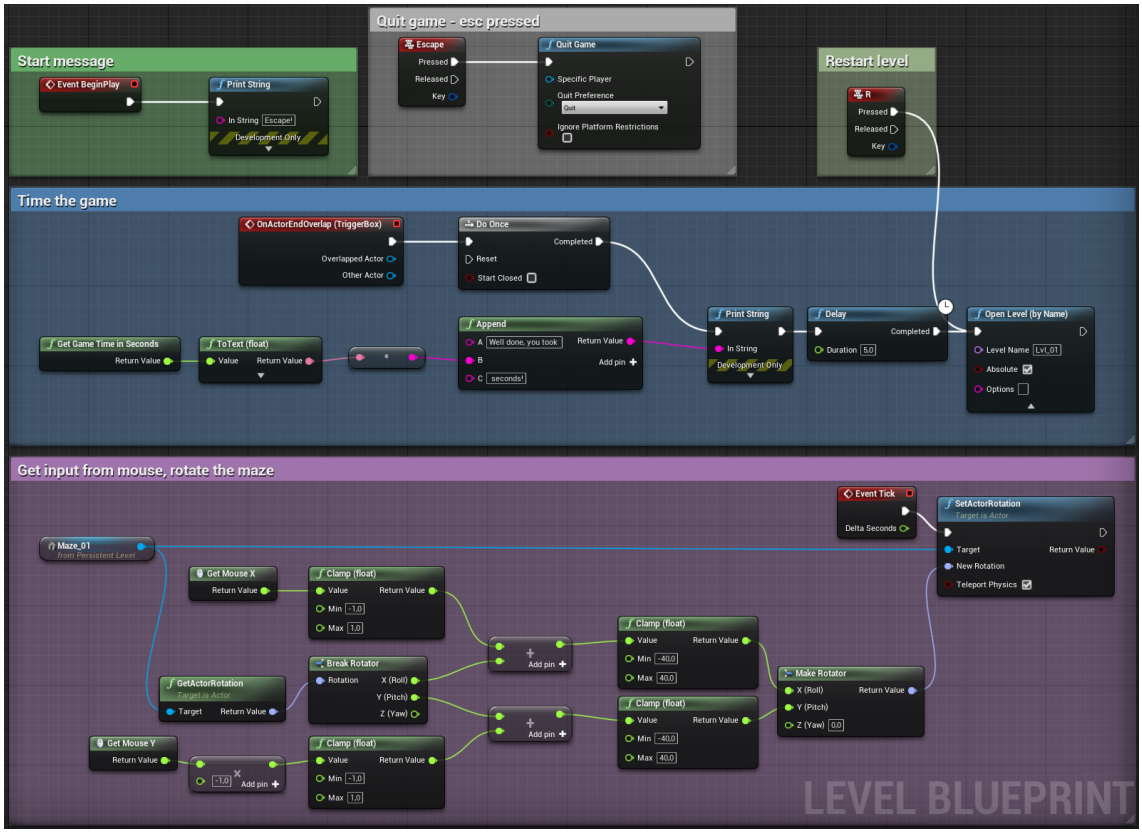


Figure 18. Marble Run only blueprint [8]. Screenshot by author.

The game starts with a printed message, telling the player to escape the maze. Next, the mouse's input is used to rotate the level on the x and y-axis while being clamped at 40 degrees in each direction so the player cannot overturn the maze, making the ball fall out. Finally, when the player has cleared the maze, a message is printed showing the game elapsed time. The blueprint also enables the player to quit the game using the escape key or restart the level by pressing the r key.

The second section brings more mechanics and complexity to the game creation. Here the player takes control of a sphere using keyboard inputs. The objective is to move around the level, picking up all the collectible crystals and avoiding hazardous traps along the way. Since the player is controlling a sphere, the movement controls are different from other shapes. Instead of adding a force to a rigid body, torque is added to the object as a change in angular acceleration, ignoring the object's mass, like Figure 19 demonstrates. That results in smooth movement on the x and y-plane.

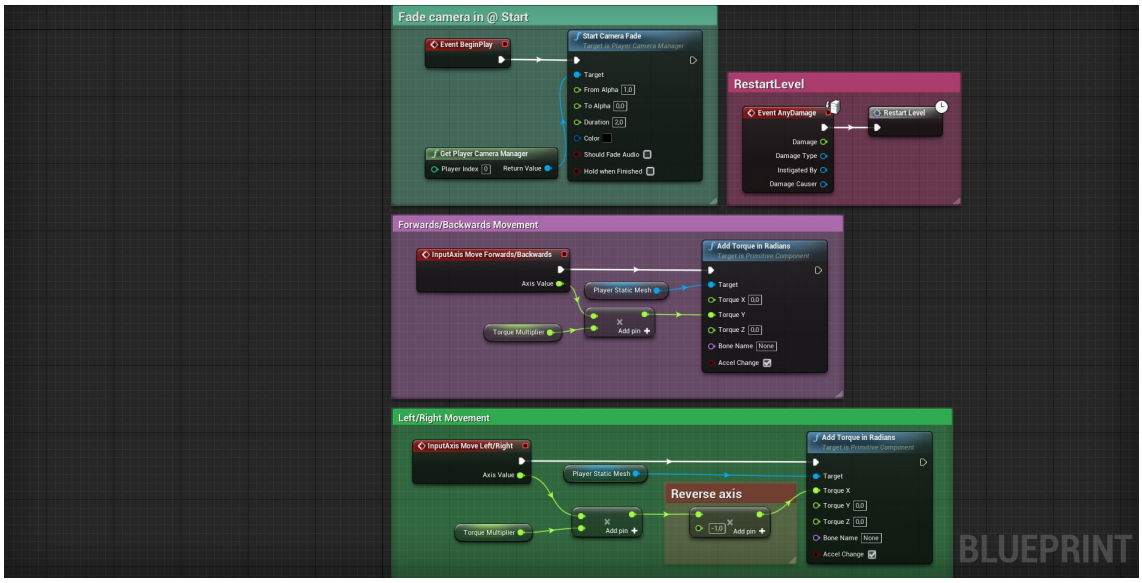


Figure 19. Player movement blueprint [8]. Screenshot by author.

The handling of the player input is different from section to section. In section one, the input is fetched straight from the mouse axis. Section two and three have a custom input mapping. That translates to more flexibility when it comes to the input setup. The input mapping can be changed or added to without disturbing the blueprint itself. Figure 20 shows the section three game input mapping. Input has been added for both keyboard and game controller, giving the user the flexibility of choosing his preferred method of playing.

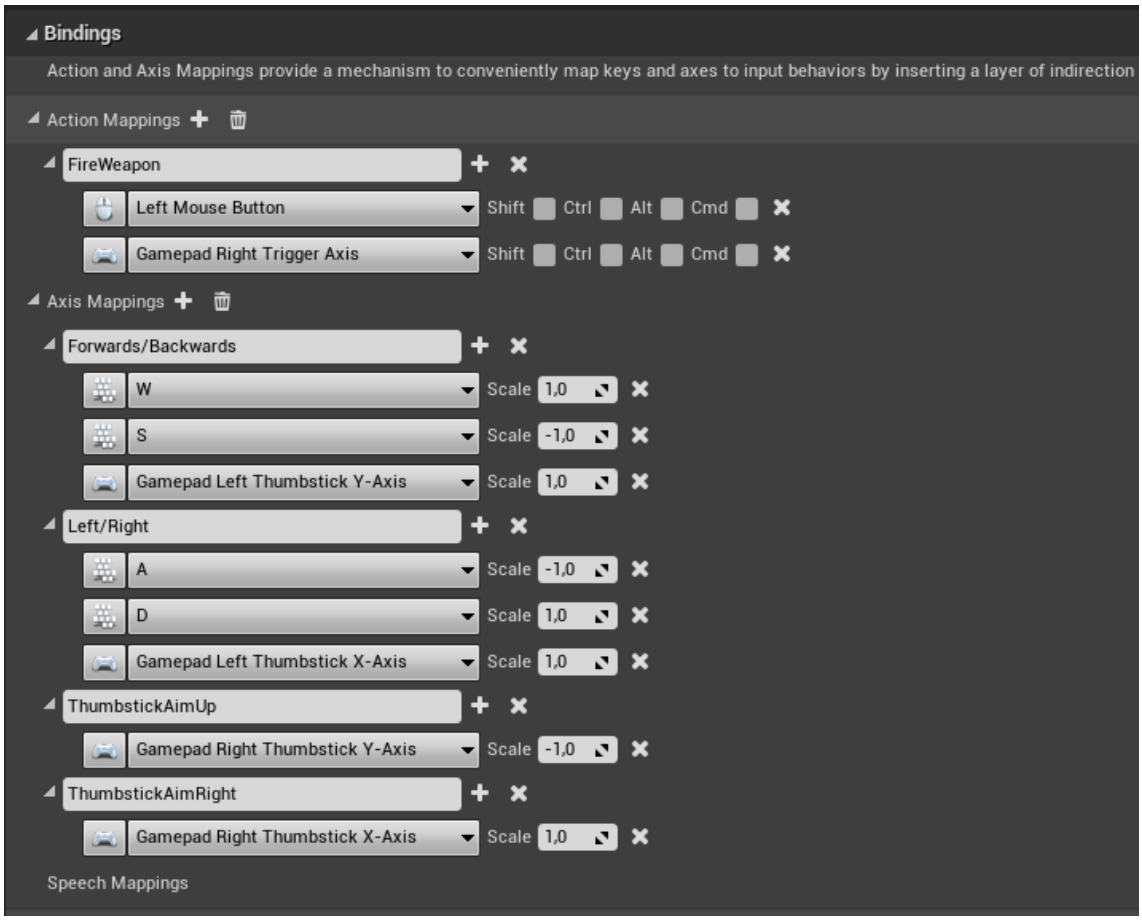


Figure 20. Custom input mappings [8]. Screenshot by author.

The game made in section three builds on the foundations that previous sections are based on. The game can be described as a top-down wave-based shooter. The player uses a keyboard and mouse, or a game controller, to control a character that shoots aliens spawning around him in waves. The game utilizes realistic assets, so both the player and the non-playable character (NPC) have animations. The animation controls for the NPC are created using a state machine that calls an animation montage when it is alive and calls a death animation when the player has killed it.

Since the player has complete control of the character, the animation for it is more complex. The animation is handled with locomotion-based blending, a particular asset that allows for animation blending using two input values. That translates to a smooth animation change between, for example, forward running

animation and side running animation. Additionally, there is also a state machine for the main character that handles whether the character is idle (no player input), moving, or dead.

5.3 Technical Comparison

Many questions arise when comparing the visual scripting languages to their programming language counterpart. One of the more frequent questions is whether the visual scripting is slower to run than the written code. Asking that question with the Unreal Engine in mind, the short answer is yes; Blueprint is slower than C++. However, that is not the whole story nor enough information to dismiss using visual scripting.

When an Unreal project is built, C++ code is compiled to machine code, a list of processing instructions that run directly on the Central Processing Unit (CPU). The Blueprint function gets compiled but not to machine code like the C++ code. Instead, the Blueprint functions are run through a script compiler that executes the engine's script virtual machine at runtime. That translates to some overhead for the C++ code since it can compile directly to code for the CPU. Here is where context kicks in. The C++ code is faster, but is the performance gap significant enough to choose it over Blueprint? The answer to that is project dependant. The built-in profiler, shown below, shows everything that is

executed in the project and how many resources it takes. If a Blueprint or code is running slow, it can be identified there and optimized. [59.]

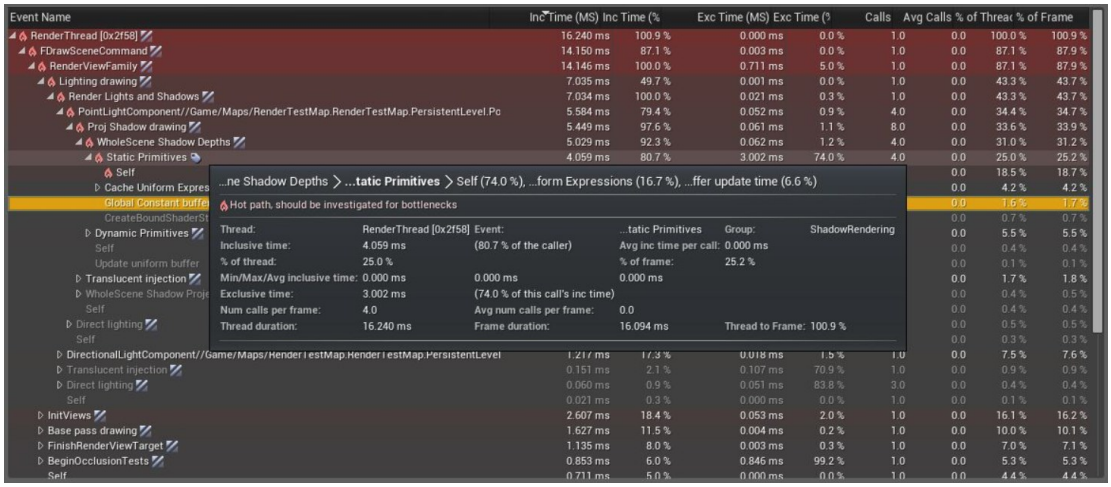


Figure 21. The profiler in Unreal Engine 4 [59].

Comparing the visual scripting in Unity versus its C# code, one can see clear signs that it was not developed as a core feature of the engine but a third-party tool. Analyzing the previous question, whether visual scripting is slower than C# code, has the same short answer, yes. Still, the long answer is slightly different due to how the VS was created in Unity. VS nodes have C# code under the hood, but the execution is not the same. VS used a technology called reflection, which means that the nodes reflect C# code based on Unity's scripting Application Programming Interface (API). This method works but carries a speed penalty. [9.]

Reflection is slow compared to regular compiled C# code. One reason for that is that the reflection technology was not designed with high-performance speeds in mind. [60.] That starts to show when projects get larger, and more objects are calling node graphs. Here, the same logic goes as for the previous one regarding the Unreal Engine. If the performance hit of the node graph is significant enough, then perhaps something needs to be restructured. One significant upside to using the VS over pure code is that the node graphs are note compiled. That means that the user can manipulate graphs at runtime, and

compilation times are next to none. According to Unity's roadmap, a high-performance runtime interpreter which would increase graph execution performance significantly is in development. That would be a notable upgrade to the platform, but there is no release date for that feature. [35.]

A quick overview of Unity's VS and Unreal's Blueprint can deduce that they are somewhat similar. Connected nodes on a graph that create logic. However, when given a closer look, differences start to appear. One of the most significant differences between them is the scope of the integration. It is evident that Blueprint was developed as a core feature of the Unreal Engine. Materials, animations, and many other systems use Blueprint as the manipulation tool, which gives a great user experience. That is what Unity lacks at the moment. However, they have indicated that the tool development is heading in a similar direction, with better overall experience and consistency over Unity's features.

Currently, Unity has no possible communications between graphs, except using custom events. There is no search function for those events nor anything in the graphs, and variables cannot be organized or grouped. All these features exist in the Unreal Engine. Many of those features are not a problem for many users, but when the project grows, it gets increasingly harder to refactor, optimize, and have a clear overview of the project's logic.

I did not run into any problems regarding the slow performance or any previous problems mentioned before. However, the scope of my projects was not at the mark where the visual scripting became a limitation. I liked more working in Blueprint, mainly because it was a clearer overall experience, and the organization tools available are better than in Unity. Yet, Unity's VS did present a fun new workflow for me, and it will be intriguing to follow the future development and integration of that tool.

6 How does Visual Scripting fit in Game Development?

As mentioned in previous chapters, visual scripting allows for quick iteration, extendibility, and increased efficiency. Using a tool such as visual scripting in a large-scale environment, where costs are high and deadlines are strict, is an easy decision. If designers had to ask a programmer to create each feature they wanted to be implemented, it would be a disaster. Visual scripting allows those artists and designers to implement features without being halted by going through another person. That translates to time saved and a better end product.

For smaller-scale operations down to a single indie game developer, the appeal of visual scripting is still there. Fast prototyping is one of the great benefits as well as ease of use. Artists and other non-coding developers who want to create games can do so without a dedicated programmer. Many games heavily rely on that, and others are entirely made with visual scripting. Good examples are the games Kine and Lab Rat from the studio Chump Squad. Both games were made with Blueprint only, which is very inspiring for people who want to make games but are intimidated by the coding language or want to try out game development. [61.]

Although visual scripting is very appealing in many ways, especially when it comes to not writing code, it is still not without its learning curve. Although the user is not writing code in the traditional sense, the connected nodes represent code blocks and functions. That means that the user needs to understand the engine, available functions, and the programming language since that is what the nodes are directly referring to. Understanding how to implement the flow of logic and what operations need calling to achieve the set goal is also essential.

Unity has moved a step closer to the ease of integration by letting the user decide what naming convention is used for the nodes. The user can choose between standard coding naming conventions, which mimics how the same logic is integrated with code. The other way they call human naming, which makes names and functions easier to understand if the user has no prior

programming knowledge, seen below. That gives the user more freedom, although the standard coding convention is often recommended since that is an excellent way to segue into writing traditional code and working with programmers who use the same convention.

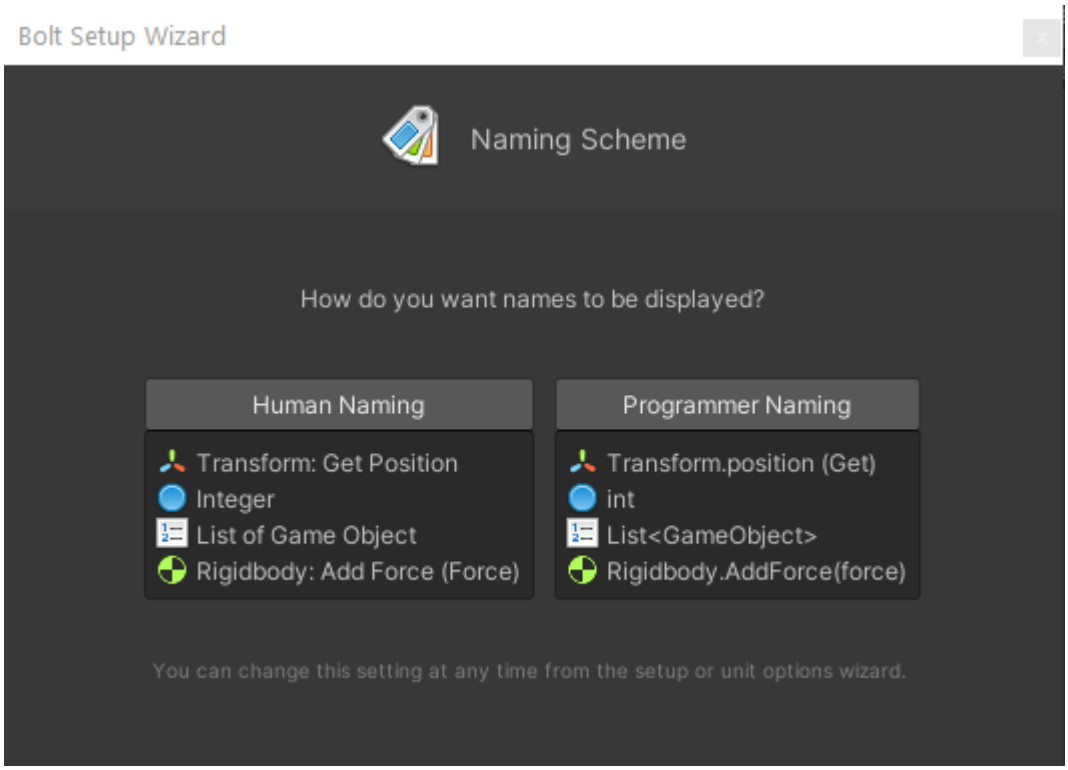


Figure 22. Unity's naming scheme setup [57]. Screenshot by author.

7 Conclusions

This thesis examined the state of visual scripting for game development, both with a practical and theoretical approach. The aim was to explore how visual scripting has been integrated into multiple industries and then narrowing the scope to the two most popular game engines available to date, the Unreal Engine and the Unity Engine.

The theoretical information behind the engines revealed some amazing development feats that have been achieved throughout the years. It was also fascinating to see how far that technology has come and, in the case of Unity's visual scripting, where it is going. Unfortunately, Unreal has no information regarding the roadmap for their Blueprint development. However, since it is tightly integrated into the engine, it will most likely evolve and adapt to any new feature that Unreal Engine publishes.

The practical part was illuminating and gave great insight into the game development capabilities of visual scripting. The hands-on experience complemented the theoretical portion, and together it painted a whole picture of the possibilities that can be achieved with this fantastic tool. Both the Unity and the Unreal projects gave an excellent fundamental understanding of the functionality and benefits of visual scripting. Although the projects did not utilize the full capabilities of the visual scripting tools, they built a solid knowledge foundation that can easily be expanded on.

Based on these conclusions and research, visual scripting is a great tool that seems to be in full development and worth giving a closer look. It is great for fast prototyping, quick iteration, and increased cooperation between people, which are great features. However, it is good to keep in mind the limitations that have been discussed. Visual scripting is slower than compiled written code; that is an architectural fact. The user might run into problems with Unity's solution concerning larger projects regarding organization and structure. Unreal's solution seems to be better integrated and more mature than Unity's to date.

The key takeaway is that visual scripting is a great tool. It can do many things, and there are other things it cannot do as well. It is excellent at productivity, prototyping, and other fast development-related things. However, it is also not as great at large-scale applications, and some management features might be lacking. That is where other tools come in with their strengths and complement visual scripting weaknesses. Furthermore, the goal set in the beginning was to look at visual scripting in game development in a theoretical and practical manner. The result looks at visual scripting from multiple angles and ultimately sees it as beneficial to numerous parties and should definitely be explored.

References

1 Get Unity - Download Archive [Internet]. Unity. [cited 2021 May 9]. Available from: https://unity3d.com/get-unity/download/archive

2 Unreal Engine 4.26 Release Notes [Internet]. [cited 2021 May 9]. Available from: https://docs.unrealengine.com/en-US/WhatsNew/Builds/ReleaseNotes/4_26/index.html

3 CRYENGINE | Introduction to Flow Graph [Internet]. CRYENGINE. [cited 2021 May 9]. Available from: https://www.cryengine.com/news/view/introduction-to-flow-graph

4 Getting started with Visual Scripting [Internet]. Godot Engine documentation. [cited 2021 May 9]. Available from: https://docs.godotengine.org/en/stable/getting_started/scripting/visual_script/getting_started.html

5 Drag And Drop Index [Internet]. [cited 2021 May 9]. Available from: https://manual.yoyogames.com/Drag_And_Drop/Drag_And_Drop_Index.htm

6 FlowCanvas | Visual Scripting | Unity Asset Store [Internet]. [cited 2021 May 9]. Available from: https://assetstore.unity.com/packages/tools/visual-scripting/flowcanvas-33903

7 NodeCanvas | Visual Scripting | Unity Asset Store [Internet]. [cited 2021 May 9]. Available from: https://assetstore.unity.com/packages/tools/visual-scripting/nodecanvas-14914

8 Unreal Engine | The most powerful real-time 3D creation platform [Internet]. Unreal Engine. [cited 2021 May 9]. Available from: https://www.unrealengine.com/en-US/

9 Technologies U. Unity engine visual scripting | Game development software without coding | Unity Bolt | Unity [Internet]. [cited 2021 May 9]. Available from: https://unity.com/products/unity-visual-scripting

10 Ellis T, Heafner JF, Sibley WL, Ellis T, Heafner JF, Sibley WL. The Grail Language and Operations. 1969.

11 Visual Basic [Internet]. Cleverism. 2016 [cited 2021 May 9]. Available from: https://www.cleverism.com/skills-and-tools/visual-basic/

12 500000000th Project of Scratch ! on Scratch [Internet]. [cited 2021 May 9]. Available from: https://scratch.mit.edu/projects/500000000

13 Scratch - Imagine, Program, Share [Internet]. [cited 2021 May 9]. Available from: https://scratch.mit.edu/

14 Babylon.js: Powerful, Beautiful, Simple, Open - Web-Based 3D At Its Best [Internet]. Babylon.js. [cited 2021 May 9]. Available from: https://www.babylonjs.com

15 Buttle P. The Power Behind Video Games: A Look at Game Engines [Internet]. Medium. 2020 [cited 2020 Oct 2]. Available from: https://medium.com/wetheplayers/the-power-behind-video-games-a-look-at-game-engines-2731315086e0

16 Engine G. Godot Engine - Free and open source 2D and 3D game engine [Internet]. Godot Engine. [cited 2021 May 9]. Available from: https://godotengine.org/

17 OGRE - Open Source 3D Graphics Engine | Home of a marvelous rendering engine [Internet]. [cited 2021 May 9]. Available from: https://www.ogre3d.org/

18 Technologies U. Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps. [Internet]. [cited 2021 May 9]. Available from: https://store.unity.com/

19 Download - Unreal Engine [Internet]. [cited 2021 May 9]. Available from: https://www.unrealengine.com/en-US/download

20 The MIT License | Open Source Initiative [Internet]. [cited 2021 May 9]. Available from: https://opensource.org/licenses/MIT

21 The GNU General Public License v3.0 - GNU Project - Free Software Foundation [Internet]. [cited 2021 May 9]. Available from: https://www.gnu.org/licenses/gpl-3.0.en.html

22 Apache License, Version 2.0 [Internet]. [cited 2021 May 9]. Available from: https://www.apache.org/licenses/LICENSE-2.0

23 TechCrunch [Internet]. TechCrunch. [cited 2020 Oct 2]. Available from: https://social.techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/. [Accessed 2 October 2020]

24 Technologies U. Wondering what Unity is? Find out who we are, where we've been and where we're going | Unity [Internet]. [cited 2021 May 9]. Available from: https://unity.com/our-company

25 Haas JK. A History of the Unity Game Engine. :45.

26 Technologies U. Unity - Manual: Versioning [Internet]. [cited 2021 May 9]. Available from: https://docs.unity3d.com/Manual/upm-semver.html

27 Technologies U. Unity Platform Roadmap | Unity [Internet]. [cited 2021 May 9]. Available from: https://unity.com/roadmap/unity-platform

28 Technologies U. Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps. [Internet]. [cited 2021 May 9]. Available from: https://store.unity.com/

29 What is the best game engine: is Unity right for you? [Internet]. GamesIndustry.biz. [cited 2021 May 9]. Available from: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you

30 Unity Asset Store - The Best Assets for Game Making [Internet]. [cited 2021 May 9]. Available from: https://assetstore.unity.com/

31 Bolt – Visual Scripting [Internet]. Unity Forum. [cited 2021 May 9]. Available from: https://forum.unity.com/threads/bolt-visual-scripting.482621/

32 Unity Technologies acquires Bolt [Internet]. Ludiq. [cited 2021 May 9]. Available from: https://ludiq.io/blog/unity-acquires-bolt

33 Bolt | Visual Scripting | Unity Asset Store [Internet]. [cited 2021 May 9]. Available from: https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802

34 Technologies U. Unity engine visual scripting | Game development software without coding | Unity Bolt | Unity [Internet]. [cited 2021 May 9]. Available from: https://unity.com/products/unity-visual-scripting

35 Gameplay & UI Design [Internet]. [cited 2021 May 9]. Available from: https://resources.unity.com/unity-engine-roadmap/gameplay-and-ui-design

36 Subnautica on Steam [Internet]. [cited 2021 May 9]. Available from: https://store.steampowered.com/app/264710/Subnautica/

37 Technologies U. Virtual Reality Development Software | VR Engine | Unity [Internet]. [cited 2021 May 9]. Available from: https://unity.com/unity/features/vr

38 Subnautica PlayStation Update 1.10 Released [Internet]. Subnautica. 2020 [cited 2021 May 9]. Available from: https://unknownworlds.com/subnautica/subnautica-playstation-update-1-10-released/

39 Shoreline [Internet]. Escape from Tarkov Wiki. [cited 2021 May 9]. Available from: https://escapefromtarkov.fandom.com/wiki/Shoreline

40 Hearthstone Official Game Site [Internet]. [cited 2021 May 9]. Available from: https://playhearthstone.com/en-us/new-to-hearthstone/

41 Hearthstone by Blizzard Entertainment - Mobile game | Unity Case Study [Internet]. Unity. [cited 2021 May 9]. Available from: https://unity3d.com/case-study/hearthstone

42 History of the Unreal Engine - IGN [Internet]. [cited 2021 May 9]. Available from: https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

43 Lee J. Learning Unreal Engine Game Development [Internet]. Packt Publishing; 2016. Available from: https://learning.oreilly.com/library/view/learning-unreal-engine/9781784398156/

44 A first look at Unreal Engine 5 [Internet]. Unreal Engine. [cited 2021 May 9]. Available from: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

45 Farris J. Forging new paths for filmmakers on The Mandalorian [Internet]. Unreal Engine. [cited 2021 May 9]. Available from: https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian

46 Nutt C. Epic's Tim Sweeney lays out the case for Unreal Engine 4 [Internet]. [cited 2021 May 9]. Available from: /view/news/213647/Epics_Tim_Sweeney_lays_out_the_case_for_Unreal_Engine_4.php

47 UDK | KismetExamples [Internet]. [cited 2021 May 9]. Available from: https://docs.unrealengine.com/udk/Three/KismetExamples.html

48 THE FLYING SPAGHETTI-CODE MONSTER [Internet]. [cited 2021 May 9]. Available from: https://www.undeaddev.com/the-flying-spaghetti-code-monster

49 Blueprint Best Practices [Internet]. [cited 2021 May 9]. Available from: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/BestPractices/index.html

50 Fortnite player count 2020 [Internet]. Statista. [cited 2021 May 9]. Available from: https://www.statista.com/statistics/746230/fortnite-players/

51 Fortnite – A Free-to-Play Battle Royale Game and More [Internet]. Epic Games' Fortnite. [cited 2021 May 9]. Available from: https://www.epicgames.com/fortnite/en-US/home

52 NEW Webby Gallery + Index [Internet]. NEW Webby Gallery + Index. [cited 2021 May 9]. Available from: http://winners.webbyawards.com/

53 Borderlands - Media [Internet]. [cited 2021 May 9]. Available from: https://borderlands.com/en-US/picture#picture-p=6

54 Arnbjörnsson A. Visual Scripting In Game Development. [Personal Interview, 10 March] Zoom; 2021 (Unpublished).

55 Returnal [Internet]. Housemarque. [cited 2021 May 9]. Available from: https://housemarque.com/games/returnal

56 Bolt Visual Scripting in Unity [Internet]. Udemy. [cited 2021 May 9]. Available from: https://www.udemy.com/course/bolt-visual-scripting-in-unity/

57 Technologies U. Unity Real-Time Development Platform | 3D, 2D VR & AR Engine [Internet]. [cited 2021 May 9]. Available from: https://unity.com/

58 Unreal Engine Blueprint Developer Tutorial: Learn Visual Scripting [Internet]. Udemy. [cited 2021 May 9]. Available from: https://www.udemy.com/course/unrealblueprint/

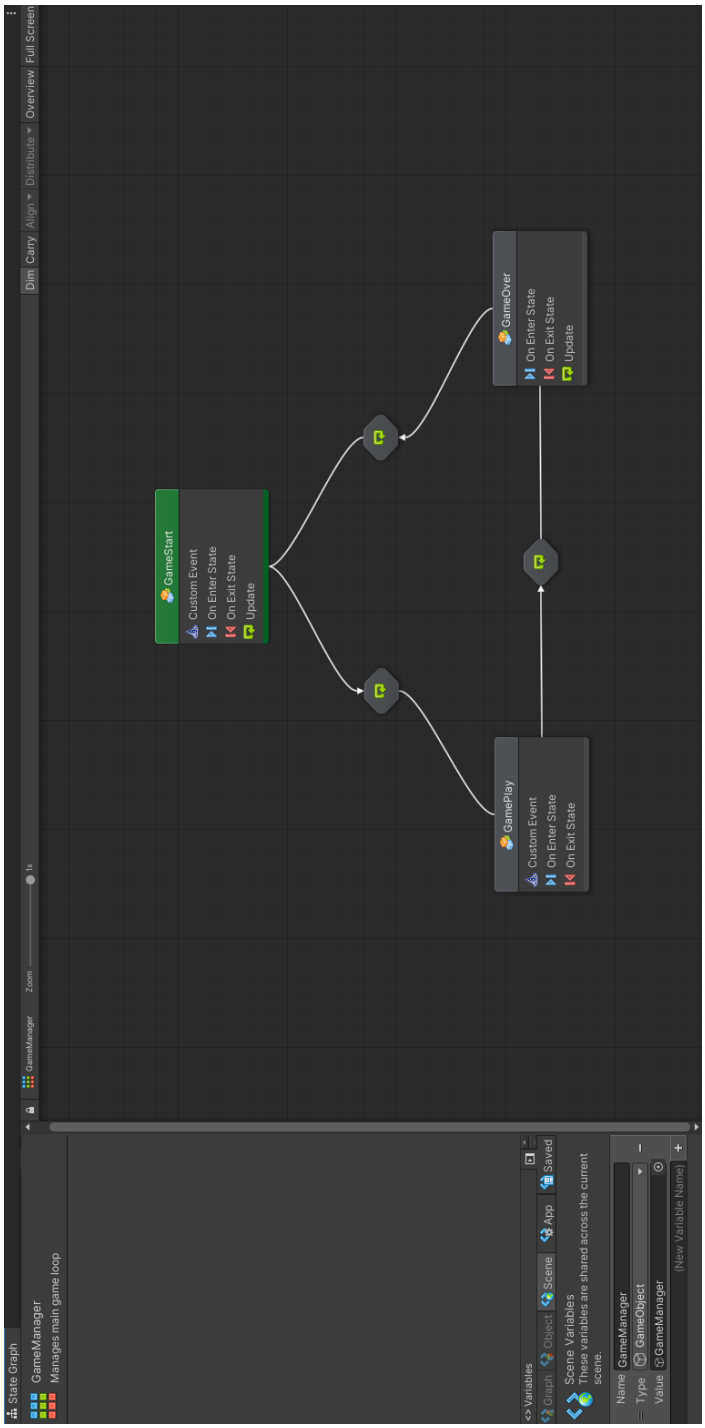
59 Profiler Tool Reference [Internet]. [cited 2021 May 9]. Available from: https://docs.unrealengine.com/en-US/TestingAndOptimization/PerformanceAndProfiling/Profiler/index.html

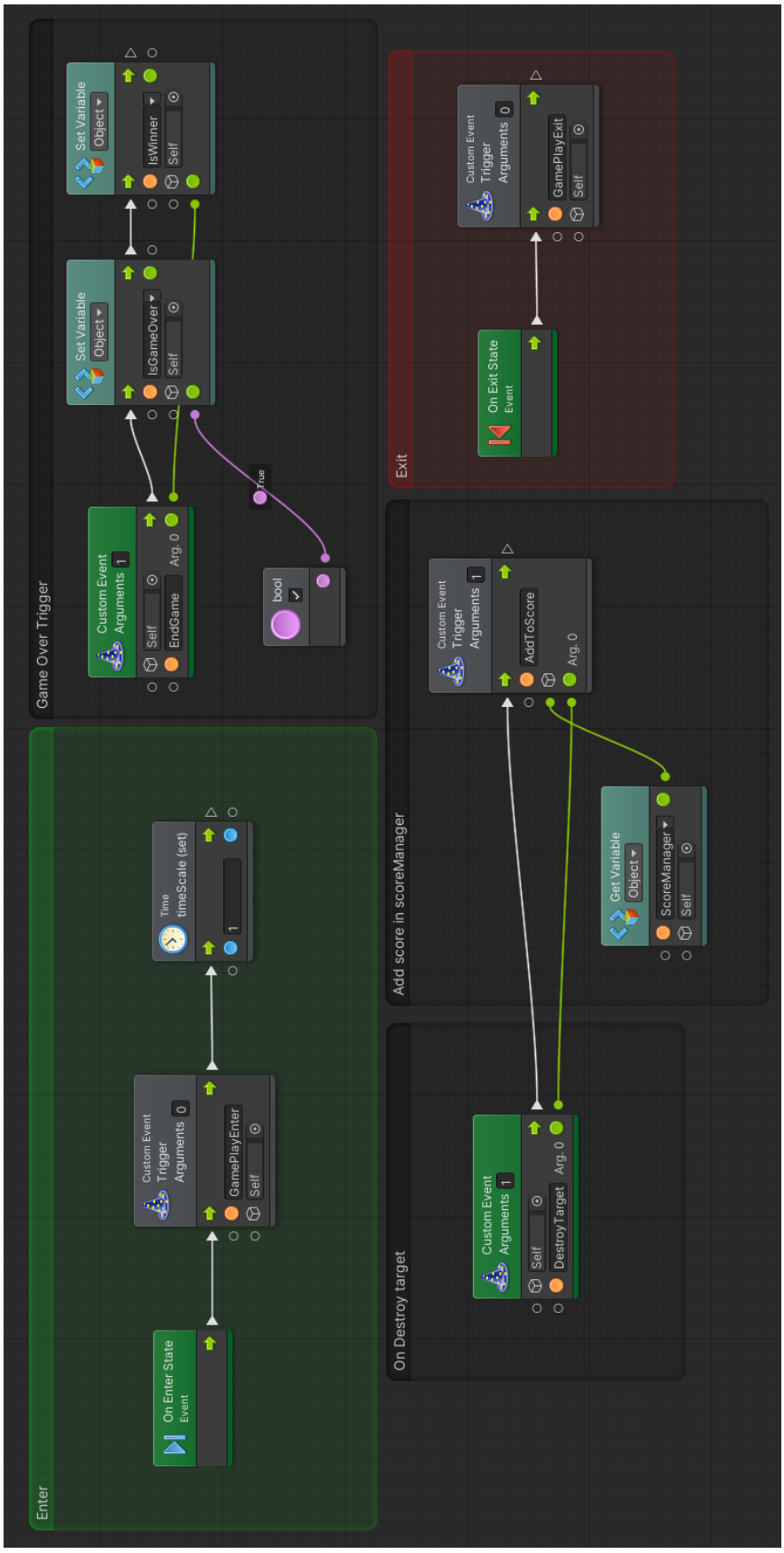
60 Warren M. Why is Reflection Slow? [Internet]. CodeProject. 2016 [cited 2021 May 9]. Available from: https://www.codeproject.com/Articles/1161127/Why-is-Reflection-Slow

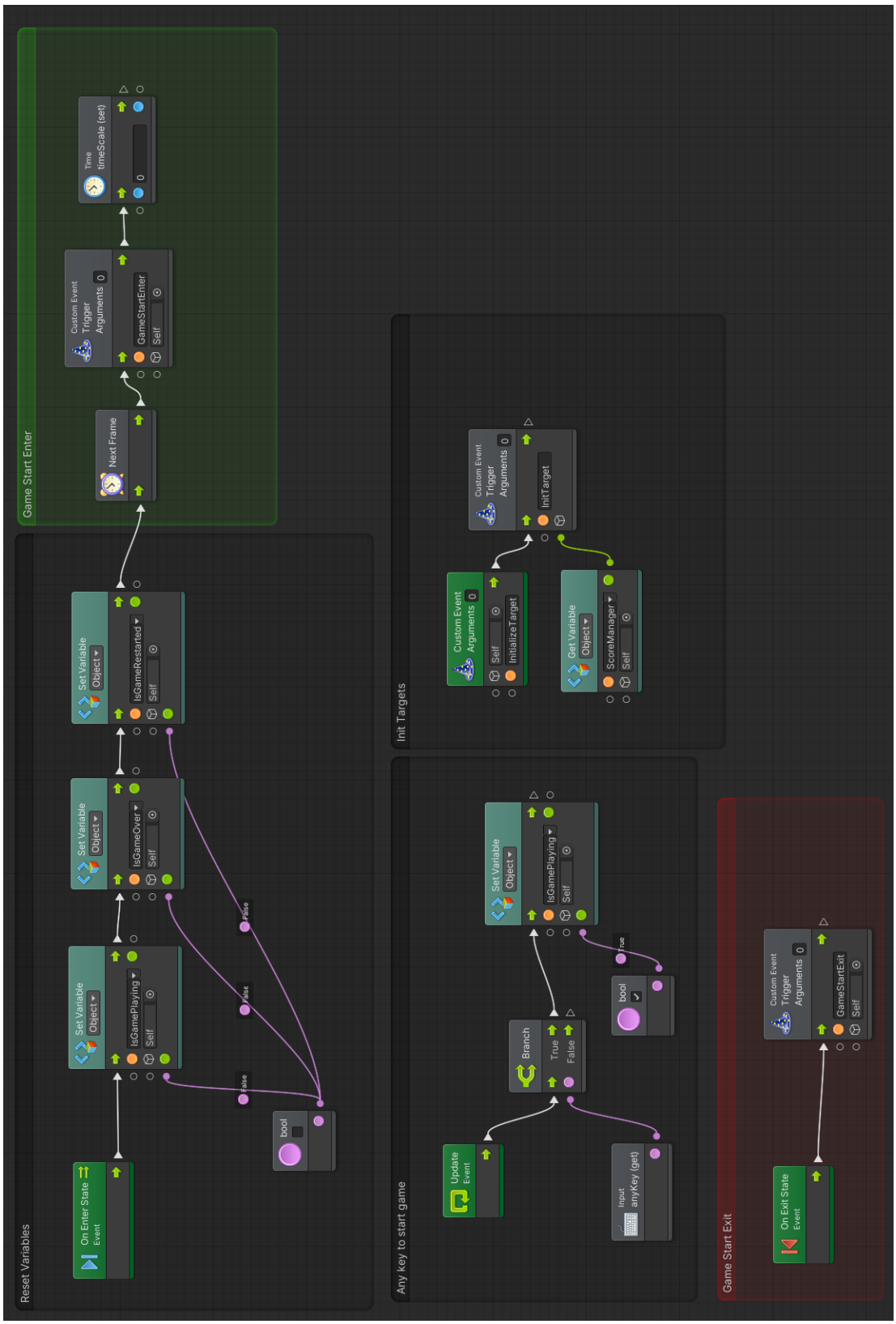
61 Frey G. From blockout to launch - a behind the scenes look at Kine's level design [Internet]. Unreal Engine. [cited 2021 May 9]. Available from: https://www.unrealengine.com/en-US/tech-blog/from-blockout-to-launch---a-behind-the-scenes-look-at-kine-s-level-design

Unity Visual Scripting Graphs

Below are the Unity Visual Scripting screenshots that were shown in the thesis. Here they have been enlarged for better viewing purposes.



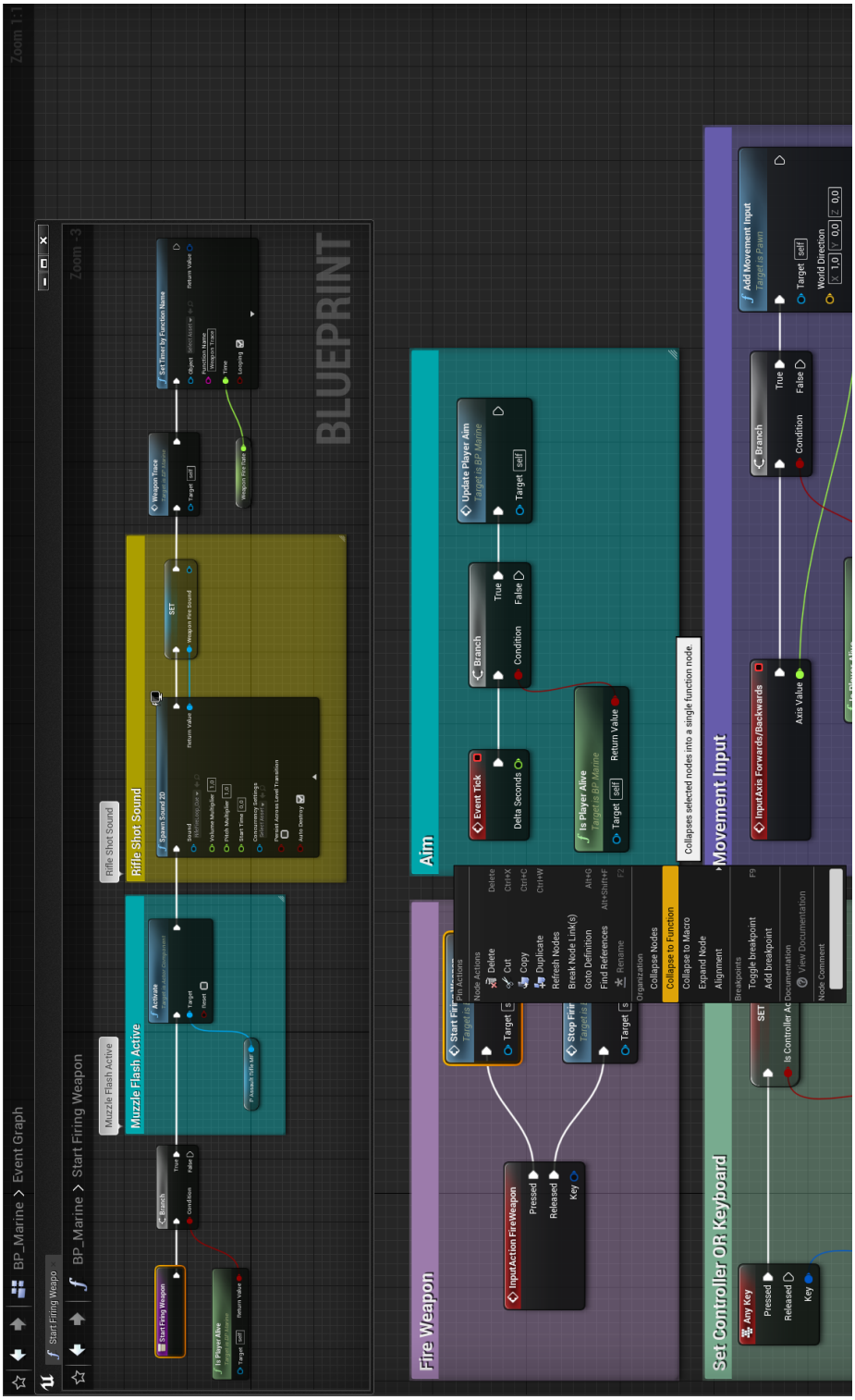

Enter
On Enter State
Event
Custom Event
Trigger
Arguments 0
GamePlayEnter
Self
Time
timeScale (set)
1
Game Over Trigger
Set Variable
Object
IsWinner
Self
Set Variable
Object
IsGameOver
Self
Custom Event
Arguments 1
Self
EndGame
Arg 0
bool
true
Exit
Custom Event
Trigger
Arguments 0
GamePlayExit
Self
On Exit State
Event
Add score in scoreManager
Custom Event
Trigger
Arguments 1
AddToScore
Arg 0
Get Variable
Object
ScoreManager
Self
On Destroy target
Custom Event
Arguments 1
Self
DestroyTarget
Arg 0

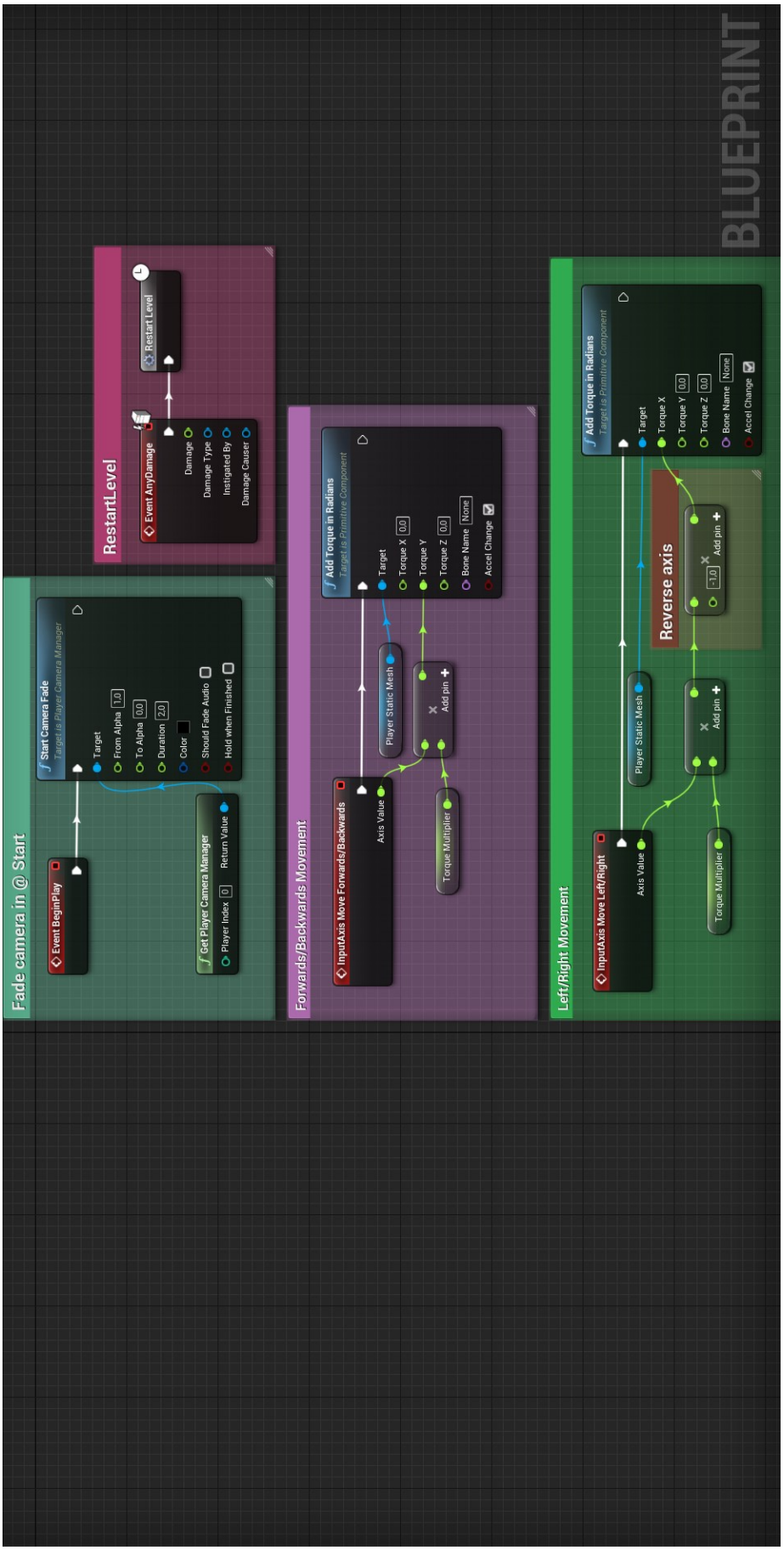

Game Start Enter
Time
timeScale (set)
0
Custom Event
Trigger
Arguments 0
GameStartEnter
Self
Next Frame
Reset Variables
On Enter State
Event
Set Variable
Object
IsGamePlaying
Self
Set Variable
Object
IsGameOver
Self
Set Variable
Object
IsGameRestarted
Self
bool
False
False
False
Init Targets
Custom Event
Trigger
Arguments 0
InitTarget
Custom Event
Arguments 0
Self
InitializeTarget
Get Variable
Object
ScoreManager
Self
Any key to start game
Update
Event
Branch
True
False
Input
anyKey (get)
Set Variable
Object
IsGamePlaying
Self
bool
True
Game Start Exit
On Exit State
Event
Custom Event
Trigger
Arguments 0
GameStartExit
Self

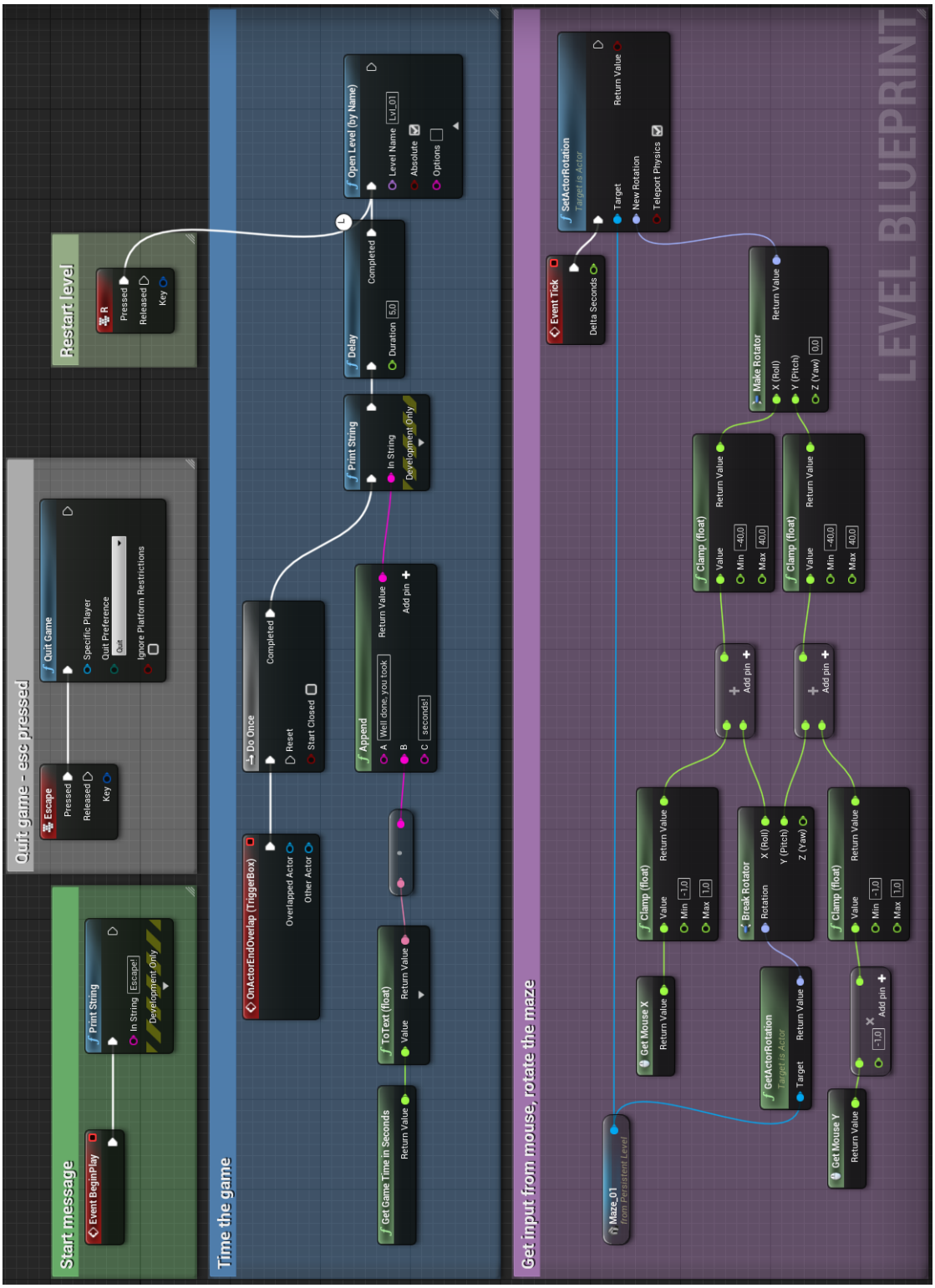
2020.3 Test_URP - SampleScene - PC, Mac & Linux Standalone (Unity 2020.3.1f1 Personal* <DX11>
2021 Test - SampleScene - PC, Mac & Linux Standalone (Unity 2021.1.0f1 Personal <DX11>
File Edit Assets GameObject Component Tutorial Tools Window Help
Hierarchy
SampleScene*
Main Camera
Directional Light
Post-process Volume
Example Assets
Scene Variables
Scene
Game
Flow Graph
Script Graph
Fixed Update Event
Transform Rotate
Self
xAngle 0
yAngle 1
zAngle 0
relativeTo World
X Angle 0
Y Angle 1
Z Angle 0
Relative To World
Project
Console
Favorites
Assets
ExampleAssets
Install Bolt
Ludiq
Macros
Materials
Presets
Scenes
Scripts
Settings
TutorialInfo
Packages
Unity.VisualScripting.Generated
_VS_Macros
Readme
Unity VisualScript...

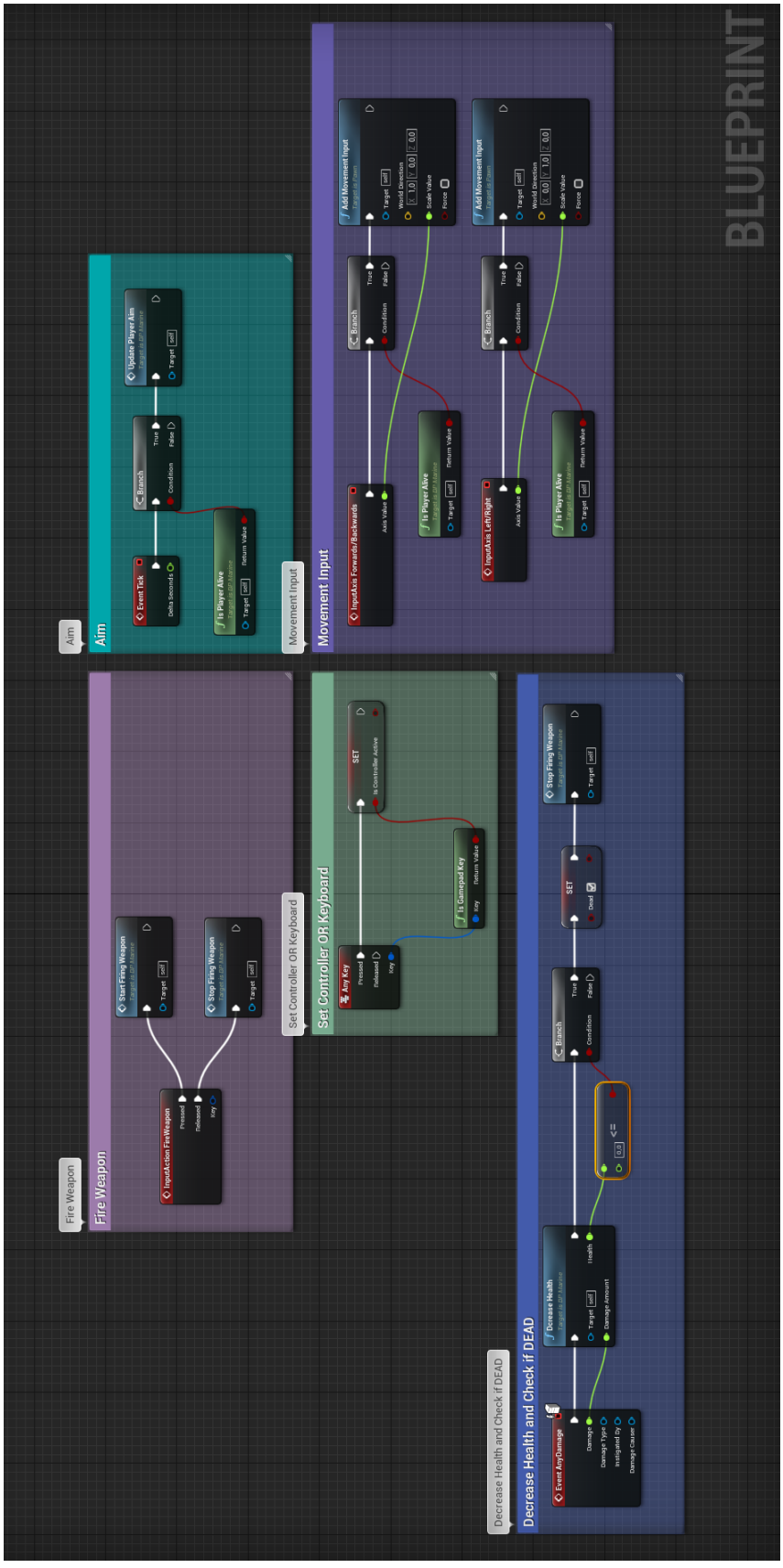
Unreal Engine Blueprints

Below are the Unreal Engine Blueprint screenshots that were shown in the thesis. Here they have been enlarged for better viewing purposes.



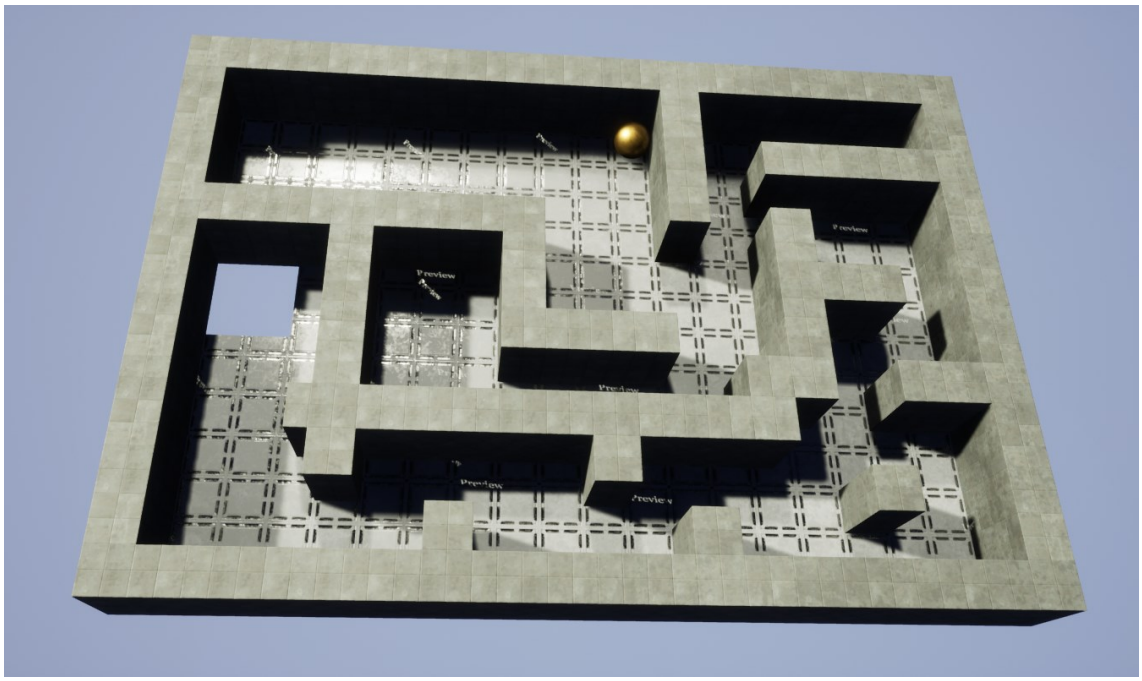

Fade camera in @ Start
Event BeginPlay
Start Camera Fade
Target is Player Camera Manager
Target
From Alpha 1.0
To Alpha 0.0
Duration 2.0
Color
Should Fade Audio
Hold when Finished
Get Player Camera Manager
Player Index 0
Return Value
RestartLevel
Event AnyDamage
Damage
Damage Type
Instigated By
Damage Causer
Restart Level
Forwards/Backwards Movement
InputAxis Move Forwards/Backwards
Axis Value
Player Static Mesh
Torque Multiplier
Add Torque in Radians
Target is Primitive Component
Target
Torque X 0.0
Torque Y
Torque Z 0.0
Bone Name None
Accel Change
Left/Right Movement
InputAxis Move Left/Right
Axis Value
Player Static Mesh
Torque Multiplier
Reverse axis
-1.0
Add Torque in Radians
Target is Primitive Component
Target
Torque X
Torque Y 0.0
Torque Z 0.0
Bone Name None
Accel Change
BLUEPRINT


Start message
Event BeginPlay
Print String
In String
Escape
Development Only
Quit game - esc pressed
Escape
Pressed
Released
Key
Quit Game
Specific Player
Quit Preference
Quit
Ignore Platform Restrictions
Restart level
R
Pressed
Released
Key
Time the game
OnActorEndOverlap (TriggerBox)
Overlapped Actor
Other Actor
Do Once
Reset
Start Closed
Completed
Delay
Duration
5.0
Completed
Open Level (by Name)
Level Name
lvl_01
Absolute
Options
Print String
In String
Development Only
Append
A
Well done, you took
B
C
seconds!
Return Value
Add pin
ToText (float)
Value
Return Value
Get Game Time in Seconds
Return Value
Get input from mouse, rotate the maze
Event Tick
Delta Seconds
SetActorRotation
Target is Actor
Target
New Rotation
Teleport Physics
Return Value
Maze_01
Clamp (float)
Value
Min
Max
Return Value
Get Mouse X
Return Value
Get Mouse Y
Return Value
Break Rotator
Rotation
X (Roll)
Y (Pitch)
Z (Yaw)
GetActorRotation
Target is Actor
Target
Return Value
Make Rotator
X (Roll)
Y (Pitch)
Z (Yaw)
Return Value
LEVEL BLUEPRINT


BLUEPRINT
Fire Weapon
Fire Weapon
Start Firing Weapon
Stop Firing Weapon
InputAction FireWeapon
Set Controller OR Keyboard
Set Controller OR Keyboard
Any Key
SET
Is Gamepad Key
Aim
Aim
Event Tick
Delta Seconds
Branch
Condition
Update Player Aim
Is Player Alive
Movement Input
Movement Input
InputAxis Forwards/Backwards
InputAxis Left/Right
Add Movement Input
Is Player Alive
Decrease Health and Check if DEAD
Decrease Health and Check if DEAD
Event AnyDamage
Decrease Health
Stop Firing Weapon
SET
Branch

Game Screenshots

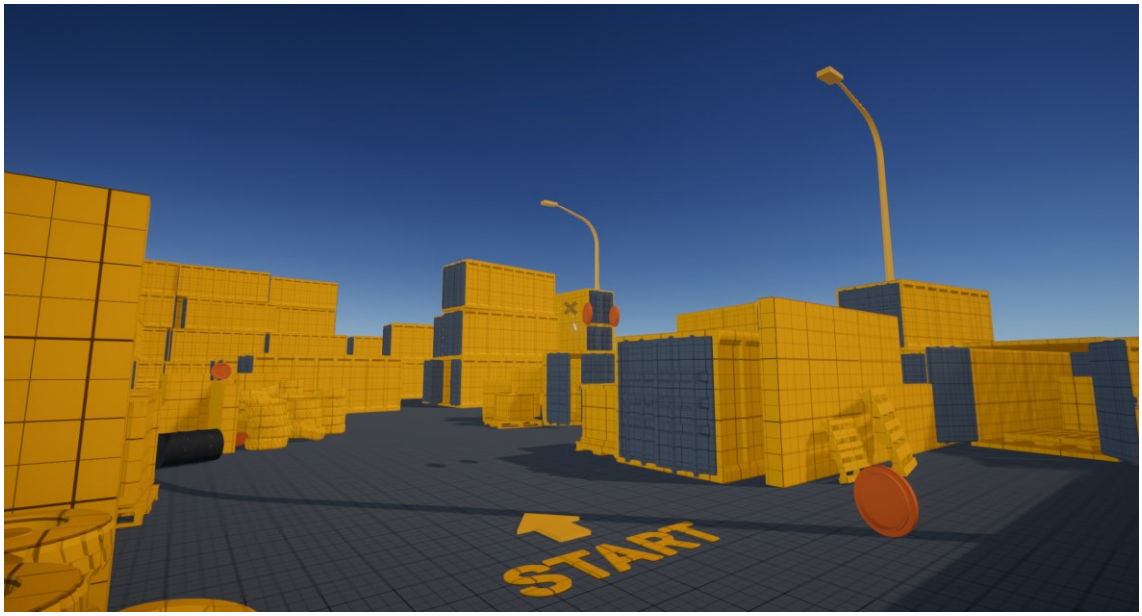
Below are screenshots from the games made in the practical portion of the thesis.



Marble Run, made in the Unreal Engine.

Crystal Cavern, made in the Unreal Engine.

Mars Marine, made in the Unreal Engine.



FPS made in Unity Engine.