EVIL PATH

Omari Paul

Tennessee State University

2019

Table of Contents

1. Introduction

Purpose

This document describes the software in its totality and is broken down into individual sections of the software engineering process. This document is intended to be used by the members of the project team that will implement and verify the correct functionality of the game. Unless otherwise noted, all aspects of the document are specified here are high priority and committed for release.

Scope

The game "Evil Path" will be played by gamers for entertainment purposes. The game will be a survival horror which is a subgenre of video games inspired by horror fiction that focuses on survival of the character as the game tries to frighten players with either horror graphics or scary ambience.

Definitions, Acronyms, and Abbreviations.

Refer to the Glossary

References

Code of Ethics https://www.igda.org/page/codeofethics.

Unreal Engine Documentation https://docs.unrealengine.com/en-us/

2. The Overall Description

Product Perspective

This Game will be played on either a PC.

Interfaces

The system is interfaced with Unreal Engine software and will be implemented using the Ureal Engine API. The Unreal Engine API contains a combination of assets and functions that will be used to create the game. The system will also be interfaced with a console or PC.

User Characteristics

The gamer will be one that enjoys the challenge of an RPG. The gamer will welcome the challenge of solving logic puzzles for game advancement. The gamer will also enjoy an occasional jump scare during the game and the usage of strategy as well as agility to defend the player character from enemies. The gamer will also enjoy a game from the 1st person perspective. The game will cater to a more mature audience. The design of the game will provide ample challenge for the experience gamer will not discouraging the casual gamer.

Constraints

(1) Platform to be released on will be restricted to (PC and Mac)
(2) The Computer must have 8Gb or memory, such and such graphics card, and such and such processing.
(3) Keyboard, mouse, and other computer compatible controls.
(4) PC with Windows 7 64-bit.
(5) 8 GB RAM and a quad-core Intel or AMD processor
(6) A DX11 compatible video card.

System Status

Work Accomplished

The demo of this product is set for release on 4/24/2019. The game has been tested to ensure that it is working properly. The database (load/save) features have been disabled/halted until further notice.

Hardware and Software Dependencies

The game should be played on a windows based system that has the specification which are stated in this document.

Expected Evolution

The first feature to be evolved will be the database aspect of the game. This will allow the user/gamer to created custom gamer ID and save/load that specific game at anytime.

3. Software Requirements Specification

Apportioning of Requirements

1st Iteration: This version of the game will incorporate basic functionality of the player character. It will also incorporate basic functionality of the enemies. It will also incorporate database to save the game progress along with player character resources.

2nd Iteration: This iteration of the will incorporate some of the level design aspects of the game and will include player and enemy functionality. This version of the game will also have some basic puzzle features in the game.

3rd Iteration: This version of the game will be the complete demo of the game. It will incorporate all the level design aspects. It will also incorporate sound in the form of music, dialogue, and SFX. This version will also contain a storyline that is able to be followed and a representation of the basic aspects of the game. This version will also contain interactive main, pause, and inventory menus.

Functional user Requirements

Requirement 1. Gamer will be able to play the game on a PC or Mac.

Requirement 2. Gamer will start playing from pre-defined location

Requirement 3. Gamer will navigate the environment from 1st person perspective.

Requirement 4. Gamer will be able to save and load the game progress.

Requirement 5. The game will have puzzles to solve.

Requirement 6. The game will have a main menu.

Requirement 7. The game will have a pause menu.

Requirement 8. The game will have an inventory menu

Non-Functional user Requirements

Requirement 9. Gamer will be able to play the game with no prior training.

Requirement 10. The environment will have a dark design to create haunting effects

Functional System Requirements

Requirement 1.1 Game shall be built using the launching feature for the PC as an .exe file and as a dmg for the Mac.

Requirement 1.2 The quality of features shall be chosen based on the capabilities of the PC and Mac.

Requirement 2.1 The game shall have a specific location in the front of the mansion as the play start location.

Requirement 2.2 The player character shall start with full health resources.

Requirement 3.1 The 1st person perspective shall display the hands of the character.

Requirement 3.2 The 1st person perspective shall display the contents contained in the hands (weapons, maps, etc…)

Requirement 4.1 The mansion environment will have specific destinations to save the game.

Requirement 4.2 The player character shall spawn back to the specific destination during the load within the mansion.

Requirement 4.3 The player character shall spawn with the same resources at the time of the save.

Requirement 5.1 The puzzle shall be designed to operate as an obstacle to impede progress until solved.

Requirement 5.2 The puzzle shall be in the form of basic pattern recognition.

Non-Functional System Requirements

Requirement 6.1 The game shall be offer options to adjust difficulty (TBD).

Requirement 7.1 The lighting in the game shall be kept to a minimum intensity.

Requirement 7.2 The environment shall be designed to mimic night time.

Requirement 7.3 The audio in the game shall be designed to enhance a dark atmosphere.

Requirement 8 The game shall be developed using Unreal Engine.

Requirement 9 The programming language used to develop the game will be c++.

4. Standards Compliance

Code of Ethics

As creators of interactive media, we, the members of the **International Game Developers Association (IGDA)**, recognize the importance of the effect of ideas conveyed through art, and especially the effect of ideas presented in an interactive choice-driven format.

Principles

As individual developers, we commit that we will:

1. Promote equal access and opportunity for game developers around the world, without regard to race, gender, creed, age, sexuality, family status, disability, national origin, or other accidental quality; and in analysis, demeanor, and expression shall be alert to the sensitivities of groups and individuals.
2. Continually strive to increase the recognition and respect of the profession; uphold the integrity of our work and credit contributions where they are due, never representing another's work as our own, or vice versa;
3. Present ourselves and our skills accurately;
4. Respect intellectual property rights;
5. Seek fair rights to ownership of content that we create;
6. Honor signed legal agreements in spirit and in letter;
7. Promote proper, responsible, and legal use of computing technology at our disposal;
8. Strive to create content appropriate for our stated audience, and never misrepresent or hide content from committees assigned to review content for communication to the public, and specifically we will work strenuously to cooperate with and support local/regional ratings boards (the ESRB, PEGI, CERO, USK, etc.).
9. Strive to share knowledge even while protecting intellectual property, for the growth of our peers as professional craftspeople and our industry;
10. Strive to promote public knowledge of technology and art, and the strengths of our industry in expanding the boundaries of art and science.
11. Promote this code of ethics within our companies, with third-party contractors and within the entire profession.

Workplace

As professionals committed to excellence in our field, we hold that:

1. Workplace safety, including physical and mental safety and comfort, is a basic right for every developer;
2. Discrimination or the tolerance of discrimination of any kind, whether on the basis of race, gender, creed, age, sexuality, family status, disability, or national origin, harms us as professionals, limits our craft, and violates this Code;
3. For the integrity of ourselves as professionals and as a professional organization, we will be aware of and adhere to all local laws in the region in which we operate, unless there is an overwhelming ethical conflict in so doing;
4. Fair treatment for developers at all levels, whether full time, part time, temporary, or student employees, is required for our operation at a professional standard.
5. Harassment of any kind, whether in a professional engagement or within the larger game development community, violates this Code and will not be tolerated;

6. Game developers deserve fair and full protection from their employer against all forms of harassment, including harassment by a colleague, by a community member, or by another party;

Leadership

As leaders in our professional field, we commit that we will:

1. Understand that an informed and physically healthy workforce benefits game development on ethical, creative, and business levels comprehensively;
2. Be forthright in communicating information pertinent to the talent that we lead, and will never knowingly deceive those whom we lead;
3. Adhere to fair schedules and contracts, never committing to delivering more than we can reasonably achieve while maintaining standards of workplace quality of life;
4. Ensure that all employment agreements are fair and legal;
5. Provide for the health of our employees to the best of our abilities;
6. Uphold trust between ourselves and those we lead by ensuring confidentiality of legal documentation and private information;
7. Promote the growth of our industry by supporting the exchange of knowledge and ideas between developers, for our mutual benefit;
8. Provide for the future of our developers and our industry by providing support for families and future developers, and acknowledge and respect the value of our veteran talent;
9. Not engage in harassment of any kind, including unwanted physical contact, or verbal or emotional abuse (both online and in person), nor tolerate those who do.
10. Ensure to the best of our ability the mental and physical well-being of those whom we lead, maintaining highest standards of workplace quality of life.

5. Attributes

Software System Attributes

The relevant attributes of the game are listed below with a description and a ranking of high (H), medium (M), or low (L) as weight of importance.

Correctness

The game will be developed to to satisfy the specifications.

Efficiency

The game will be optimized to run on as little resources as possible while not sacrificing on quality.

Maintainability

The game will be maintained overtime via new version developed through agile method as well as updates.

Reliability

The game will be screened to ensure that bugs are kept to a minimum.

Testability

Game will be tested by gamers to ensure that the gamer experience is satisfactory.

ID	Characteristic	H/M/L
1	Correctness	H
2	Efficiency	M
3	Maintainability	H
4	Reliability	M
5	Testability	H

Table 1. Characteristic satisfactory chart.

6. System Architecture

Architectural Model

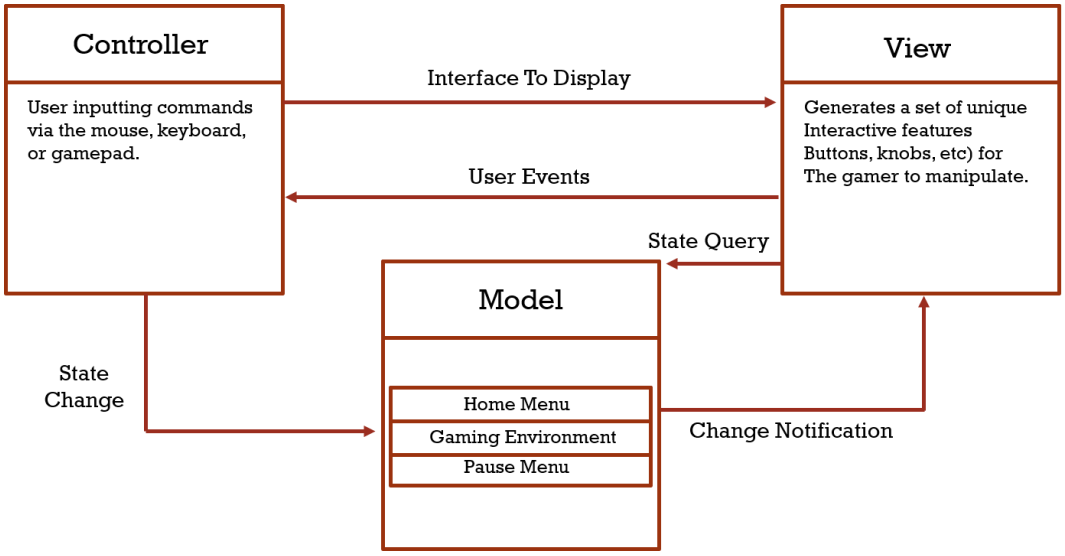


Figure 1: System Architecture.

The overall architecture of the system is determined to be best modeled by a combination of Model View Controller (MVC) and Layered Architecture. The Model View Controller illustrates the system as a whole. The controller represents the incoming commands into the system from the user. These inputs will change the state of the system. The changing state of the system will update the view in the user interface. Each view of the system will display a unique set of interactive components that the user can select. **Figure 1** illustrates the controller, model, view, and the description between their directed links.

System Context Model

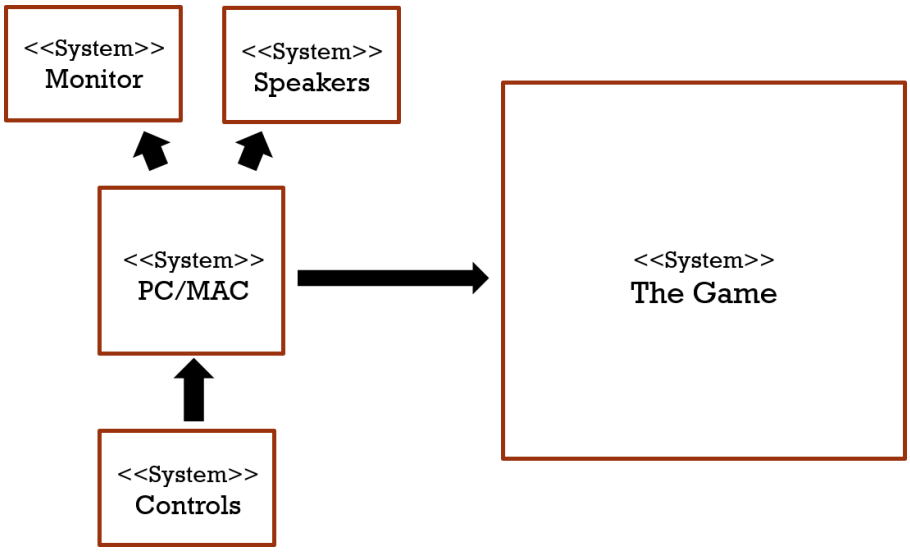


Figure 2: System Context Model.

The system can be looked at from an isolated perspective but often times it is beneficial to analyze a system in full context. The context model illustrated in **Figure 2** depicts other systems that the game will be interfaced with. The game contextually is run on a PC/Mac which will be realized through the GUI on a monitor and through the speakers.

Sub System Model

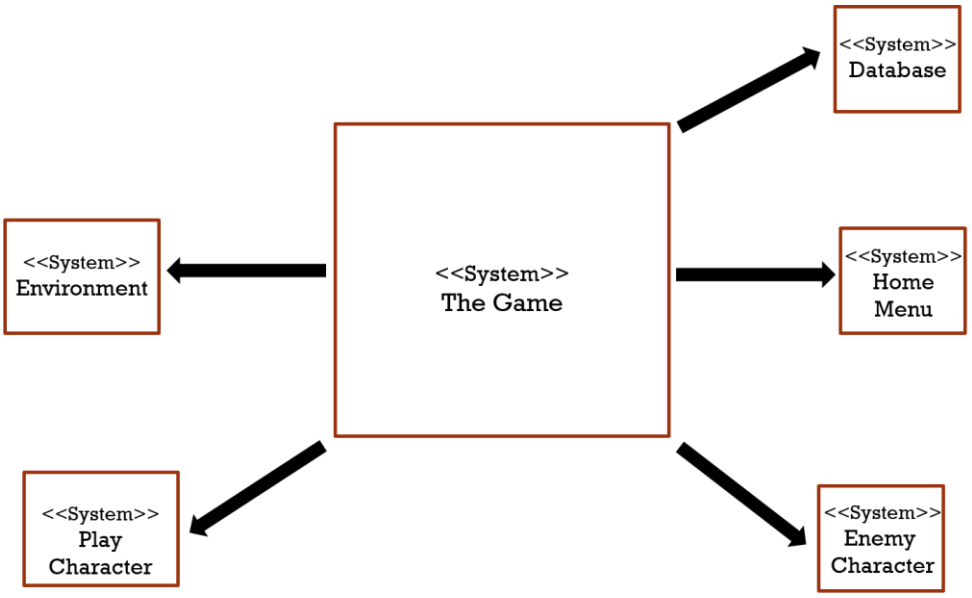


Figure 3: Subsystem Model.

The system/game itself is comprised of a number of subsystems. The systems are the environment, database, home menu, player character, and enemy character. During development, each of these subsystems will be developed as unique aspects of the system/game. Each subsystem will be programmed to interact with the other subsystems at an interface. More details on how these subsystems are interfaced is described in more detail below in the state machine diagram.

State Machine

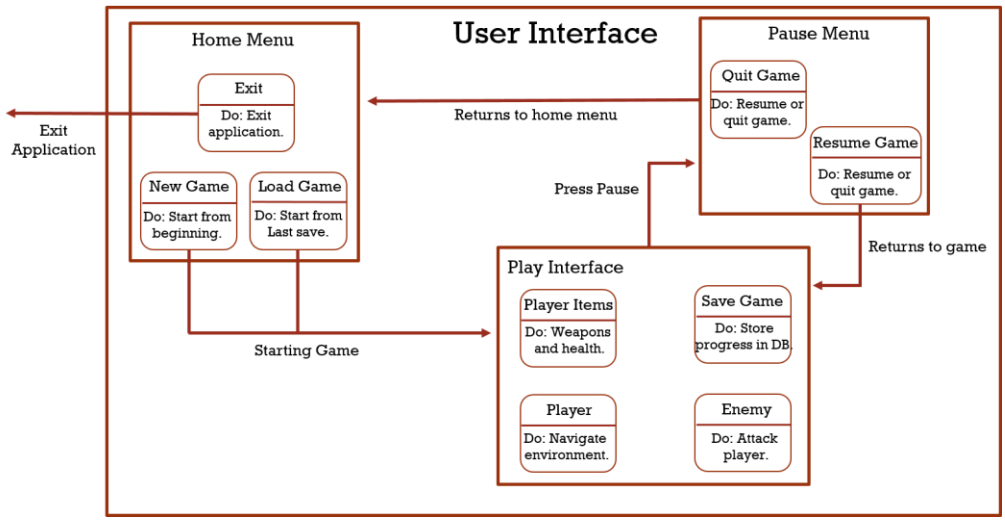


Figure 4: State Machine.

The state machine model gives description on how the subsystems are interfaced with each other. The interfacing depicts certain actions causing the state of the system to transition into another state. For example, the home state contains three features which allow the user to exit game, start new game, or load existing game. The exit feature will cause the system to exit the application. The start new game and load game feature will transition the state of the system into playing game state. In the playing game state, the user can interact with items, enemies, and can save the game progress. The user also has the option to pause the game which transition the game into a pause menu. The pause menu gives the user the option to resume the game or quit the game. The resume game option will transition the game back into the playing state. The quit game option will transition the game back to the home menu. Contained within the home menu in the load game section will be the games that have been previously saved by the user. These saved games will be stored in a database which will be discussed in more detail in the next section of the SDD.

7. Database Design

Tables Schema

Player Accounts			
Description	The username and current progress of the player character		
Attribute	**Description**	**Type**	**Examples of values**
Name	Name of a user	String	John The Explorer
Health	Current health of player	Float	0 to 100%
Location	Location of save	Coordinate	(x, y, z)
Experience	Current progress	Referenced	Completed Quest
Items	Possessions of player	Referenced	Hand gun, first aid kit

Tables 2. Database Attributes.

The table list the different attributes that the user will be able to save during the course of the game. When loading a saved game, the data base will be accessed and retrieve the name, health, location, experience, and items that the player character possessed at the last point of saving the game. This database feature will ensure that the gamers playing and progress to finish the game will remain consistent and unimpeded.

8. Graphical User Interface

Home Menu

Figure 5. Home Menu.

The home menu will be the screen that the game defaults to after the application has loaded. The home menu derives from **user requirements 3** and **4** by allowing the user/gamer the option to start game from the beginning or previously saved location. The home menu contains the controls for new game, load game, and quit game. The new game control will transition the game into a default beginning which will start the player in a predefined location with predefined resources within the playing environment. The load game control according to **functional requirements 4.2** and **4.3** will spawn the player back to the specific destination, within the playing environment, with the same resources at the time of the last save. The quit game control will simply terminate the application taking the user/gamer back to the desktop of the PC/Mac.

Playing Environment

Figure 6. Playing Environment.

The playing environment will be called/triggered by the user/gamer selecting the new game or load game control. In the playing environment, the user/gamer will navigate in the 1st person perspective according to **user requirement 3**. The user/gamer will be able to interact with and equip them as weapons or store them as resources according to **functional requirements 3.1** and **3.2**. The user/gamer will also interact with puzzles in the playing environment according to **user requirement 5**. The puzzle will be interactive and placed to prevent progress until the puzzle is solved according to **functional requirement 5.1**. The gamer/user will also be able to save the game progress at specified locations within the playing environment according **user requirement 4**. The save game feature will allow the user to store their location, items, and health, and progress according to **functional requirements 4.2** and **4.3**. The user will also be able to call a pause menu.

Pause Menu

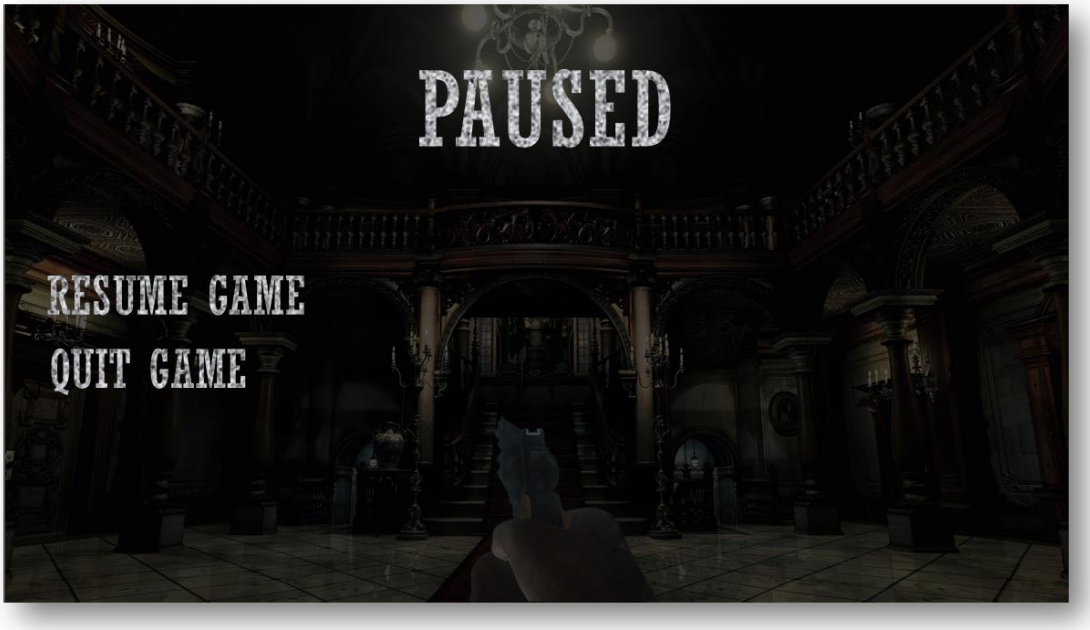


Figure 7. Pause Menu.

The pause menu will allow the user/gamer to freeze the game. While the game is in the pause state the user/gamer has the options of either quitting the game or resuming the game. If the user selects the control of resuming game, then it will transition back to the playing environment allowing the user/gamer to continue in the exact state before the pause. If the user/gamer selects the control of quit game then the game will transition back to the home menu with the three options of new game, load game or exit game.

9. Object Class

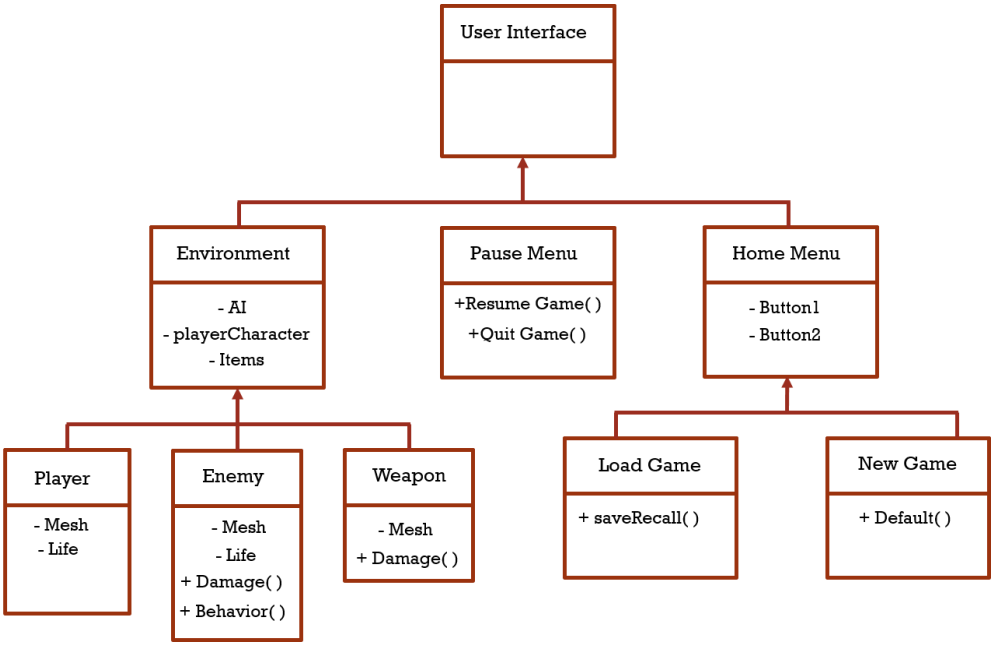


Figure 8: Objective Class Diagram.

The user interface can exist in one of three states at any moment in time which are the home menu, playing environment, and pause menu. The home menu will contain three control buttons which will perform unique methods. The new game button will generate a game which will return values with default predefined settings (location, health, items, progress, etc.) in the playing environment. These predefined setting that the new game starts with will be designated as the beginning of the game. The load game control will call a saveRecall method that will restore a game that has been previously saved by a user/gamer. The load game control will access the contents of a database that contains the saved settings (location, health, items, progress, etc.) and will return values for a game based on those features. The exit game control will simply terminate and exit the application.

The playing environment will generate a player character, AI, and interactive items. The player character will be generated or restored based on the attributes discussed load and save game. The weapons are generated with a specified threshold of damage they can inflict and are placed within the environment. Once the weapons are collected by the player character they are now able to be stored in an inventory and attached to the player's hand socket. The weapon location will return a value relative to the player character once collected and relative to the playing environment prior to being collect. The damage method will be called by the weapon when being used to return an integer value to inflict upon an enemy. The enemy will be of AI type and will be generated in specified locations in the environment. The enemy will have a certain amount of life which will be a float data type. The enemy will be able to call the damage method and will return a float value to inflict damage on the player character base on enemy class. The enemy will also call a behavior function that will return a set of

instructions for the enemy to follow based on the environment. The return data types will be location and movement related.

The pause menu contains two buttons which are resume game and quit game. If the user selects the control of resuming game, then it will transition back to the playing environment allowing the user/gamer to continue in the exact state before the pause. If the user/gamer selects the control of quit game then the game will transition back to the home menu with the three options of new game, load game or exit game.

10. Design process

The methodology used to for the design process was to keep in mind that the user/gamer should be able to have a set of options at any point after the launching of the application. When first launching the application what will be the set of options? While playing the game what will be the set of options? From this we were able to begin to design and develop around those options as the primary framework for the game indicated in the **Figure 8** below.

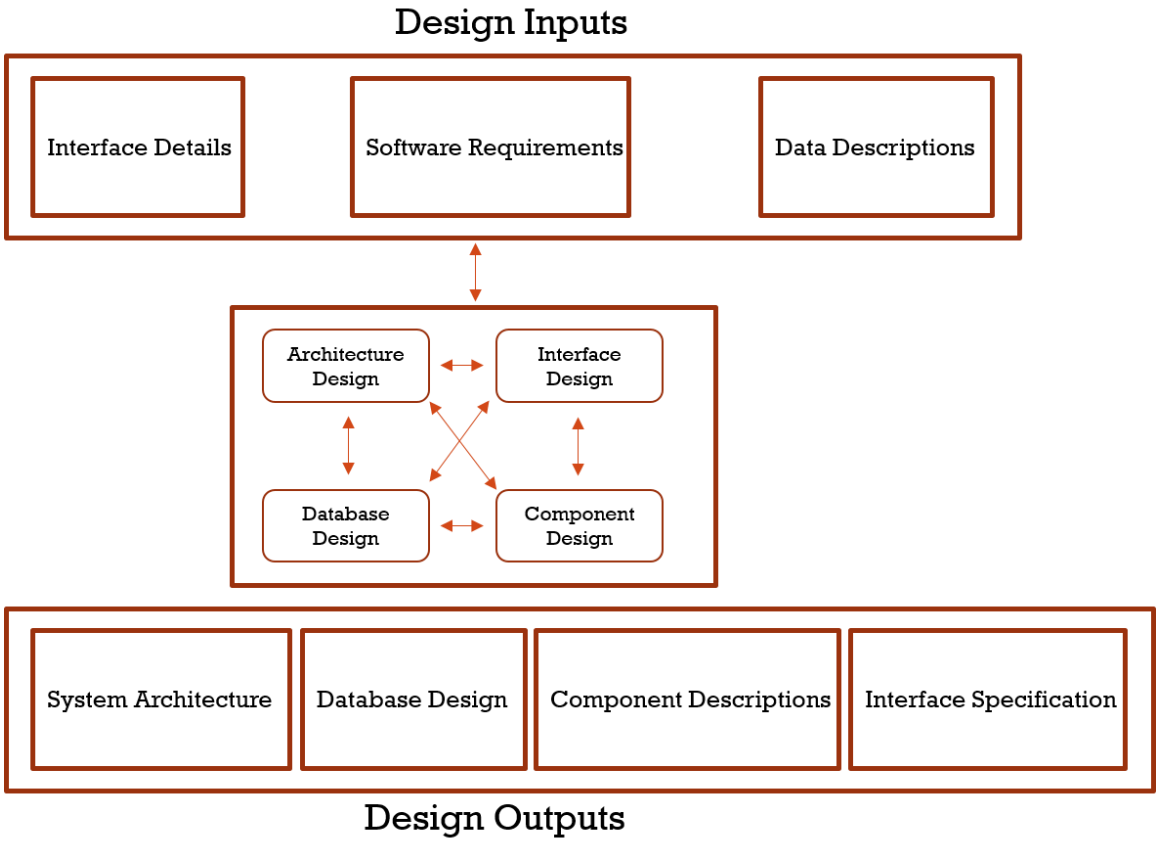


Figure 9: Design Process Model.

11. Test Design

Overview

To establish a plan for testing, we will need to identify the test items and what features to test by building forms for recording the test and summary report.
The verification will be done by performing a unit test while the validation will be in two stages, first System testing, secondly Beta testing.

Test Plan

The plan will be based on establishing a schedule for testing and identify what resources going to be used and then should have a fixed form that record what to test and what is the results.

Schedules and Resources

The form below should be filed out with actual date to do the test and what resources needed.

Test phase	From	To	By	Resources
Unit testing				
System testing				
Beta testing				

Table 3. For actual data.

Unit testing done by the developers of the software as they are working on each unit to ensure that each unit gives the right outputs before starting with the next unit. After finishing the main functions of the software, system testing should be performed by m1(not one of the developers). Beta test done by targeted customer with help from the developers.

Test Recording

The following form shall fill out after each test and it can be used for any kind of test.

Item	Tested by	date	Result expected	Test result	Correct and retest
Start menu			New, quit		M1
Pause menu			Resume, quit		M1
Saving location			Can save progress		M1
Environment			Easy to explore		M3
Door mechanics			Open and shut		M3
inventory			Save picked up items		M3
AI			Move and hit		M2
Puzzle			solvable		M2
health			Decrease on hit		M2

Table 4. For unit, system, beta testing records respectively.

Test Reporting

The following form shall summaries the test results. Could have one form for each test results or one form for all tests.

Item	Test result	Retesting date
Start menu		
Pause menu		
Saving location		
Environment		
Door mechanics		
inventory		
AI		
Puzzle		
health		

Table 5. For overall test report.

Verification Testing

Unit Testing

Each unit in the SDD should be tested by m2 and/or m3 to make sure it is work properly before moving to the next one. So, this is done by developers as they are writing the code. Here is a list of the basic units that form the Evil Path game and the approaches to test them:

Start menu: click on (New game) to load new game, click on (Exit game) to exit.

Pause menu: while playing, press the letter (P) on the keyboard to access the Pause menu. Click on (Resume game) to resume, click on (Quit game) to quit the game.

Saving location: take the main player to the save location (near the stairs). Press (K) to save, move to some where ealse, press (Z) to load the previous location (near the stairs). Note: watch the health par, should regain the saved value.

Environment: look around the whole level to observe that lights, colors, and all shapes make sure these are reflecting the game purpose and the user need.

Door mechanics: take the main player to any door in the game, when prompt, press (E) to open the door.

Inventory: while playing, press (I) to access the inventory. If you pick up something, make sure it will appear in the inventory.

AI: take the main player to the AI character (zombie). AI character shall fallow and try to damage the main player.

Puzzle: try different combinations, make sure that only one shall solve the puzzle.

Health: take the main player to the AI character (zombie). The AI shall apply damage to the main character and the health par shall decrease until destroying it.

Validation Testing

System Testing

Group member one (m1, not developer) perform highly integrated testing (System Testing) to validate all requirements on the same platform that has bean used to develop the game. Will use the following use case to validate each related requirement/s:

1) Start a new game by selecting new game button from the home/main menu.
Requirement 2. Gamer will start playing from pre-defined location.
Requirement 2.1 the game shall have a specific location in the front of the mansion as the play start location.

2) Check in the upper left corner of screen to ensure full health bar.
Requirement 2.2 the player character shall start with full health resources.

3) find enemy and verify that they attack and subtract life.
Updated in new requirement doc.

4) Go to the save location at the bottom of the stairway and save.
Requirement 4.1 the mansion environment will have specific destinations to save the game.

5) load game and verify that the same life was loaded.
Requirement 4.2 the player character shall spawn back to the specific destination during the load within the mansion.

6) Head to the gallery door. Use prompt to open door.
Updated in new requirement doc.

7) solve logic puzzle to obtain item.
Requirement 5.1 the puzzle shall be designed to operate as an obstacle to impede progress until solved.
Requirement 5.2 the puzzle shall be in the form of basic pattern recognition.
Requirement 7.3 the audio in the game shall be designed to enhance a dark atmosphere.

8) Navigate environment to find item to pick up.
Updated in new requirement doc.

9) Open item menu by pressing "I" on keyboard to ensure item is stored.
Updated in new requirement doc.

10) Close menu by pressing "Close" button.

Updated in new requirement doc.

11) Pause the game and ensure that mouse is activated for interaction.
Updated in new requirement doc.

12) Quit Game.
Updated in new requirement doc.

Note: m1 found that the blocks (statues) used in the puzzle could be stack in the corner which will need farther attention.

Beta Testing

Beta test performed in class by using an executable file of the game and the following use case:

1) Launch the game by opening exe. File.
Requirement 1.1 Game shall be build using the launching feature for the PC as an .exe file.
Requirement 1.2 the quality of features shall be chosen based on the capability of the PC.

2) Start a new game by selecting new game button from the home/main menu.
Requirement 2. Gamer will start playing from pre-defined location.
Requirement 2.1 the game shall have a specific location in the front of the mansion as the play start location.

3) Check in the upper left corner of screen to ensure full health bar.
Requirement 2.2 the player character shall start with full health resources.

4) find enemy and verify that they attack and subtract life.
Updated in new requirement doc.

5) Go to the save location at the bottom of the stairway and save.
Requirement 4.1 the mansion environment will have specific destinations to save the game.

6) load game and verify that the same life was loaded.
Requirement 4.2 the player character shall spawn back to the specific destination during the load within the mansion.

7) Head to the gallery door. Use prompt to open door.
Updated in new requirement doc.

8) solve logic puzzle to obtain item.
Requirement 5.1 the puzzle shall be designed to operate as an obstacle to impede progress until solved.

Requirement 5.2 the puzzle shall be in the form of basic pattern recognition.
Requirement 7.3 the audio in the game shall be designed to enhance a dark atmosphere.

9) Navigate environment to find item to pick up.
Updated in new requirement doc.

10) Open item menu by pressing "I" on keyboard to ensure item is stored.
Updated in new requirement doc.

11) Close menu by pressing "Close" button.
Updated in new requirement doc.

12) Pause the game and ensure that mouse is activated for interaction.
Updated in new requirement doc.

13) Quit Game.
Updated in new requirement doc.

Results and recommendations

The current version of Evil Path game has passed the most tested points. Note that the scenario above provides a test for some units that is not in current version of SRS. Since we use an increment method to build our software, all documents will be updates and this test will be useful for future updates. We did address, in system testing, that we will need to modify the puzzle. Farther more two more points came up in this test.
Frist, to dark to play: since it is a horror game, it is meant to be dark, but it shall not affect the playing ability. The actual version was not to dark but when build the exe. We tried to reduce quality for more portability and this was the reason behind the poor lighting.
Next exe. We will try to, even increase lighting or increase the quality.

Second, no clue to solve the puzzle: there is a clue in the same room where the puzzle at but probably we will need to add an audio or text for more clarification.

Test Schedule

Test phase	From	To	By	Resources
Unit testing	03/01/2019	03/10/2019	m2&m3	Cave lap
System testing	03/10/2019	03/13/2019	m1	Cave lap
Beta testing	03/27/2019	03/27/2019	in class	In class

Table 6. schedule and resources.

Item	Tested by	date	Result expected	Test result	Correct and retest

Start menu	M1	03/09/2019	New, quit	pass	M1
Pause menu	M1	03/08/2019	Resume, quit	pass	M1
Saving location	M1	03/07/2019	Can save progress	pass	M1
Environment	M3	03/03/2019	Easy to explore	pass	M3
Door mechanics	M3	03/02/2019	Open and shut	pass	M3
inventory	M3	03/05/2019	Save picked up items	pass	M3
AI	M2	03/01/2019	Move and hit	pass	M2
Puzzle	M2	03/10/2019	solvable	pass	M2
health	M2	03/04/2019	Decrease on hit	pass	M2

Table 7. test record for unit testing.

Item	Tested by	date	Result expected	Test result	Correct and retest
Start menu	M1	03/10/2019	New, quit	pass	M1
Pause menu	M1	03/10/2019	Resume, quit	pass	M1
Saving location	M1	03/10/2019	Can save progress	pass	M1
Environment	M1	03/11/2019	Easy to explore	pass	M3
Door mechanics	M1	03/11/2019	Open and shut	pass	M3
inventory	M1	03/12/2019	Save picked up items	pass	M3
AI	M1	03/12/2019	Move and hit	pass	M2
Puzzle	M1	03/13/2019	solvable	statues could be stack in the corners	M2
health	M1	03/13/2019	Decrease on hit	pass	M2

Table 8. test record for system testing.

Item	Tested by	date	Result expected	Test result	Correct and retest
Start menu	Groub#2	03/27/2019	New, quit	pass	M1
Pause menu	Groub#2	03/27/2019	Resume, quit	pass	M1
Saving location	Groub#2	03/27/2019	Can save progress	pass	M1
Environment	Groub#2	03/27/2019	Easy to explore	To dark	M3
Door mechanics	Groub#2	03/27/2019	Open and shut	pass	M3
inventory	Groub#2	03/27/2019	Save picked up items	pass	M3
AI	Groub#2	03/27/2019	Move and hit	pass	M2
Puzzle	Groub#2	03/27/2019	solvable	Not clear	M2
health	Groub#2	03/27/2019	Decrease on hit	pass	M2

Table 9. test record for beta testing.

Item	Test result	Retesting date
Start menu	p	
Pause menu	P	
Saving location	P	
Environment	F	TBD
Door mechanics	P	
inventory	P	
AI	P	
Puzzle	F	TBD
health	p	

Table 10. For overall test report.

12. User Manual

Overview of Document

The document contains a combination of verbal and visual description of features of the game. The icons, characters, and enemies will be defined in detail to inform the gamer of options during game play.

Controls

The game will be controlled using an Xbox Controller or keyboard and mouse. The first control feature is the ability to maneuver the character through the environment using the keyboard and mouse or Xbox controller. **Figure 10** shows the features of the mouse and keyboard that allow for the user/gamer to move the character left, right, forward, and backward.

(a) (b)

Figure 10: (a) Xbox Controller Left Thumb Stick and (b) Keyboard Right directional arrows.

The *right thumb stick* on the Xbox controller allows for adjustment in the Yaw or vertical axis view of the game by moving the thumb stick up to look up or down to look down. The mouse also controls the Yaw by moving the mouse *forward* to look up in the vertical axis or move the mouse *backward* to look down. **Figure 11** depicts the features on the Xbox controller and the mouse that allow for Yaw adjustment.

(a) (b)

Figure 12: (a) Xbox Controller Right Thumb Stick and (b) Keyboard Forward and Backward motion.

The pause functionality can be accessed by using the *right special button* depicted in **Figure 12**. The pause button opens up a menu that allows for the user/gamer to resume game, load game, or quit game.

(a) (b)

Figure 13: (a) Xbox Controller Right Special Button and (b) P on the Keyboard will pause the game.

The Inventory Menu can be accessed by using the *left special button* depicted in **Figure 14**. Opening the inventory menu will reveal five spaces for items to stored. These items can be accessed by clicking on the icon which is an indicator of the type of item being stored in the inventory. Once the user/gamer is finished with the inventory it can be closed by clicking on the close button.

(a) (b)

Figure 14: (a) Xbox Controller Left Special Button and (b) the 'I' on the Keyboard.

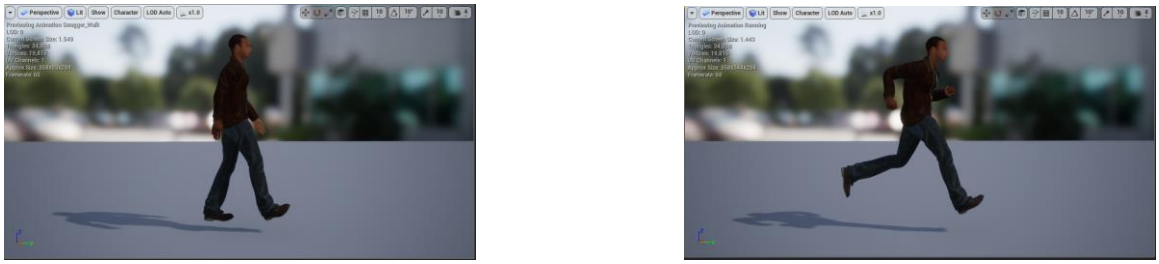
The *right face button* is the interactive/action button depicted in **Figure 15**. This button is used when prompted by the game to accomplish task such as opening doors, picking up items, and moving objects. This button is used to main interact with the different features in the environment. The *left face button* is the attacking button. The button will be used when Ben has a weapon equipped. The types of attacks will be unique to the weapon type and the abilities of Ben.

(a) (b) (c)

Figure 15: Xbox Controller (a) Left Face Button is for attacking and the Right Face Button is for interaction. (b) On the mouse the left mouse button is for attacking and the (c) 'E' Button on the Keyboard is for Interaction.

Main Character (Ben)

Help Ben remember and regain his sense of orientation in the world as you explore various challenges, obstacles, and foes! Explore the game as Ben while avoiding danger and collecting resources! Ben has two main modes of movement running and walking. These two stages of movement can be triggered based on the degree which the *left thumb stick* is tilted an any direction depicted in **Figure 16**. The less the tilt will result in walking and greater tilt will result in running.



(a) (b)

Figure 16: (a) Animation for walking. (b) Animation of running.

Ben is also able to interact with the environment when prompted by the game interface. An example is Ben's ability to push an object when prompted to press the interactive button which is the *right face button* depicted in **Figure 17**.

Figure 17: Animation for pushing.

There are more animations of Ben interacting with the environment such as: pulling, pressing buttons, lifting, jumping, etc. Pushing is just one example. Ben also has the ability to defend himself in the environment when the user/gamer presses the *left face button*. Ben can wield different types of weapons. **Figure 18** below shows Ben's animation for using a melee weapon such as a; knife, sword, or crowbar.

Figure 18: Animation for attacking.

Other Game Features

The game contains three main interfaces which are the Main Screen, Load Game, Game Mode, Pause Screen, and Inventory Screen depicted in **Figure 9**. The Main Screen contains three buttons which are *New Game*, Load Game, and *Quit Game*.

Figure 19: Main Screen.

Pressing the *Load Game* will take you into a screen where the user/gamer can load previously saved game or click *Main Menu* to go back to *Main Menu Screen*.

Figure 20: Load Screen.

Selecting the *New Game button* will start the game from the beginning. While playing the game the user/gamer can access items in the *Inventory* depicted in **Figure 21**.

Figure 21: Inventory Screen.

While playing the user/gamer can also pause the game to access the *Pause Menu* which contains buttons for *Resume Game*, *Load Game*, and *Exit Game*. *Resume Game* simply takes the user/gamer back to the normal game mode depicted in **Figure 22**. *Exit Game* returns the user/gamer back to the *Main Menu* and *Load Game* allows for the user/gamer to load previously saved game.

Figure 22: Inventory Screen.

Ben is also able to interact with certain objects such as *doors* and push able *statues* depicted in **Figure 23**.

Figure 23: Movable Object (Statue).

The user/gamer can also save the progress of the game by interacting with the Save Game objects depicted in **Figure 24**. The save feature will only be executed when Ben is directly next to the *Save Game* object.

Figure 24: Save Game Object.

13. Glossary

Unreal Engine - Unreal Engine 4 is a complete suite of game development tools made by game developers, for game developers.

Role Player Game (RPG) - A role-playing game is a game in which players assume the roles of characters in a fictional setting.

Application Programming Interface (API) - a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.

Survival Horror - Survival horror is a subgenre of video games inspired by horror fiction that focuses on survival of the character as the game tries to frighten players with either horror graphics or scary ambience.

M1 - person responsible on documentations.

M2 – developer.

M3 – developer.

Cave lap - the lap where the game developed ..

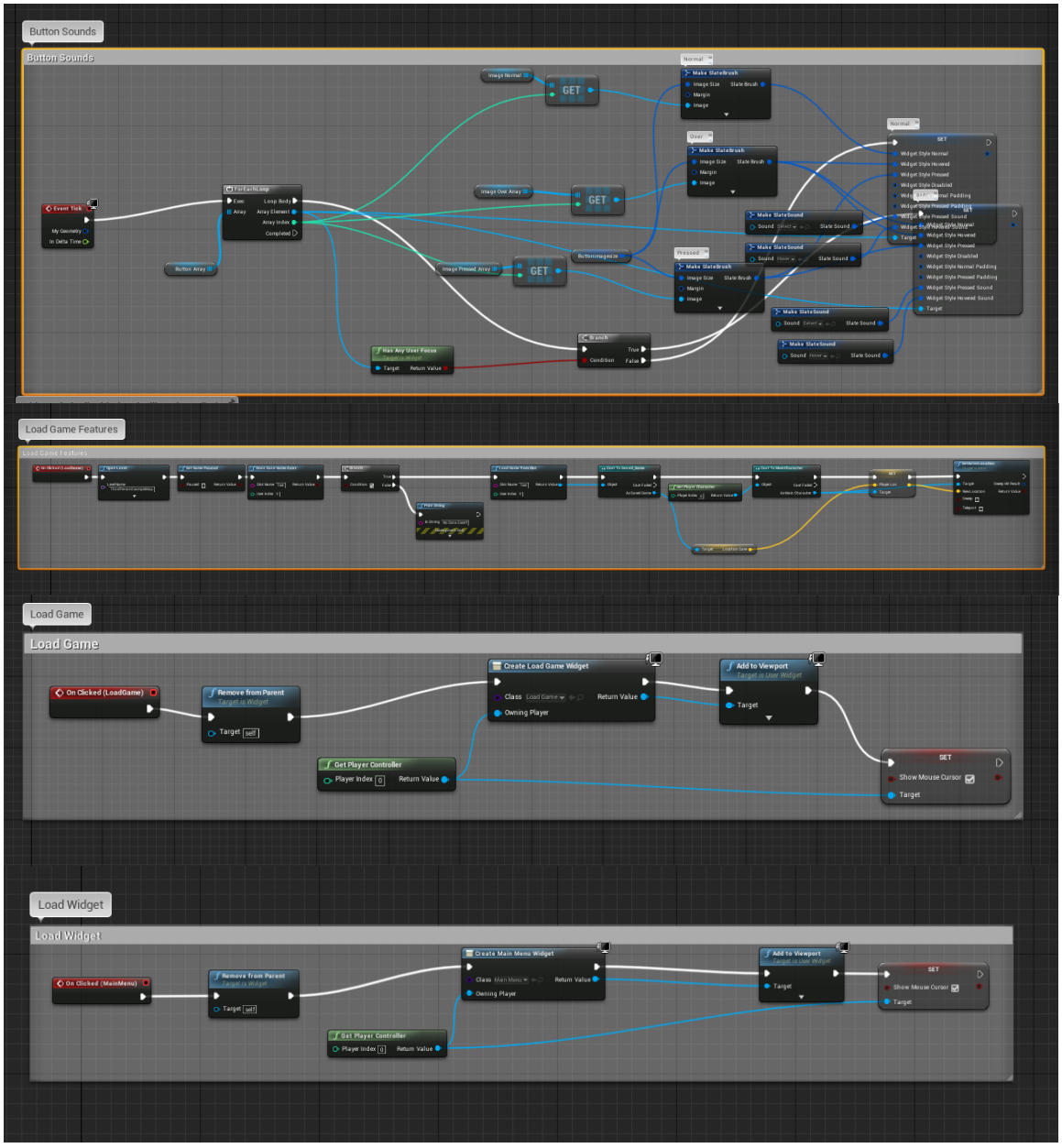
Environment - all the space inside the game including opjects..

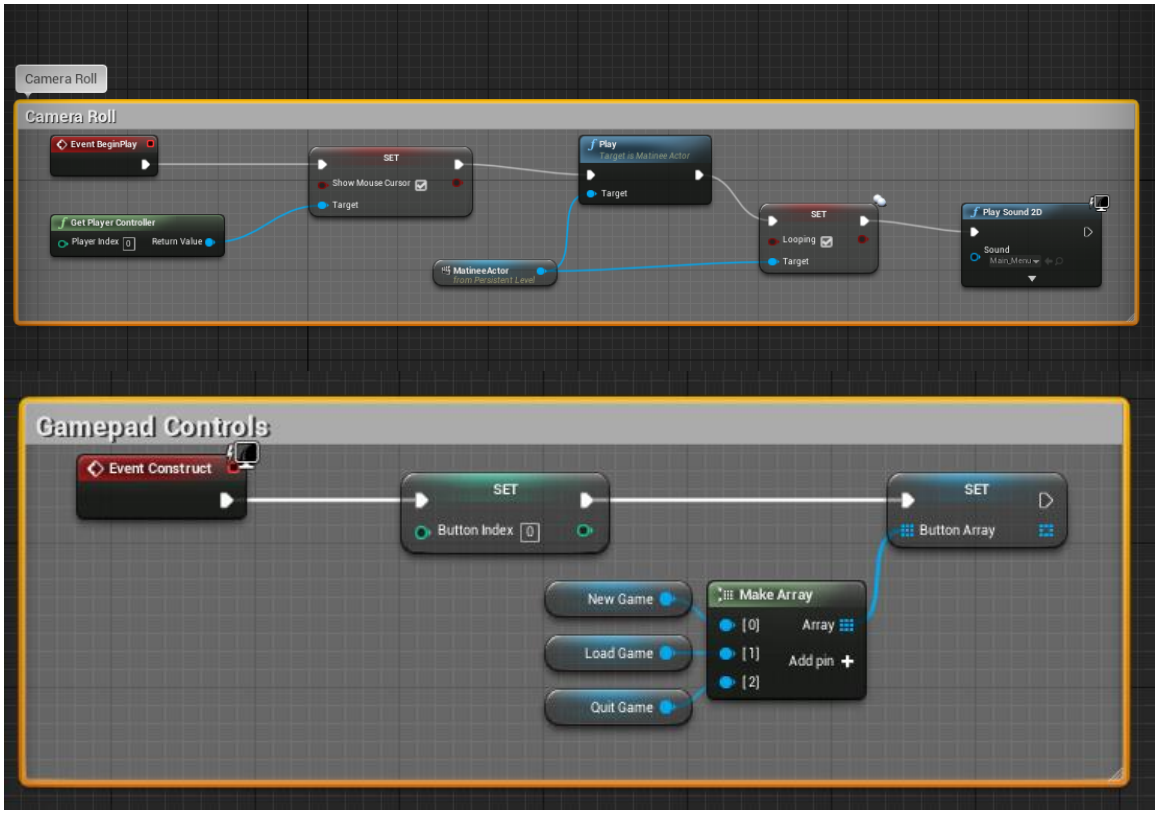
AI - the enemy (zombie).

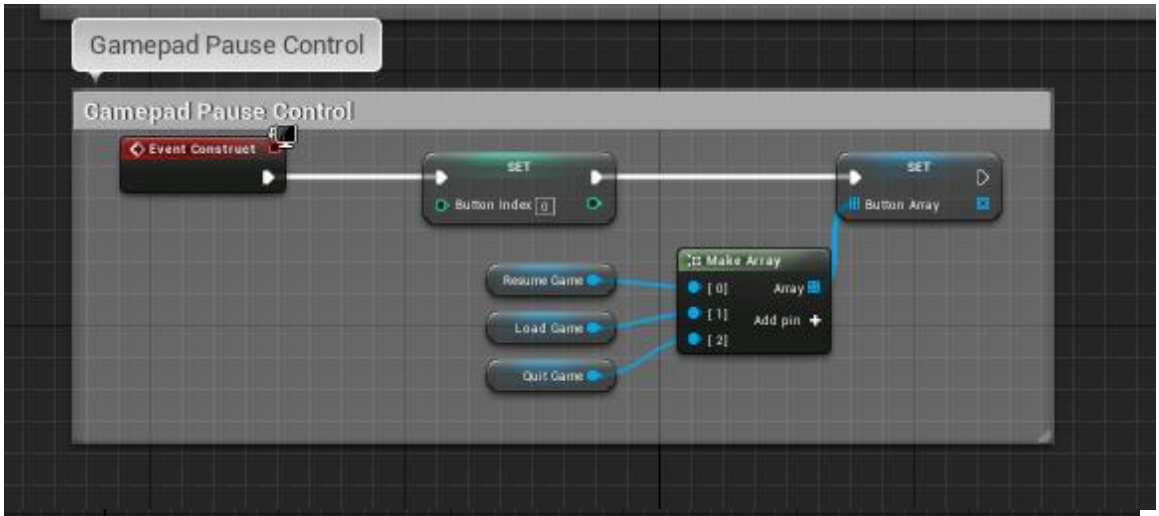
Main player (character) - The character with whom the gamer will navigate the game

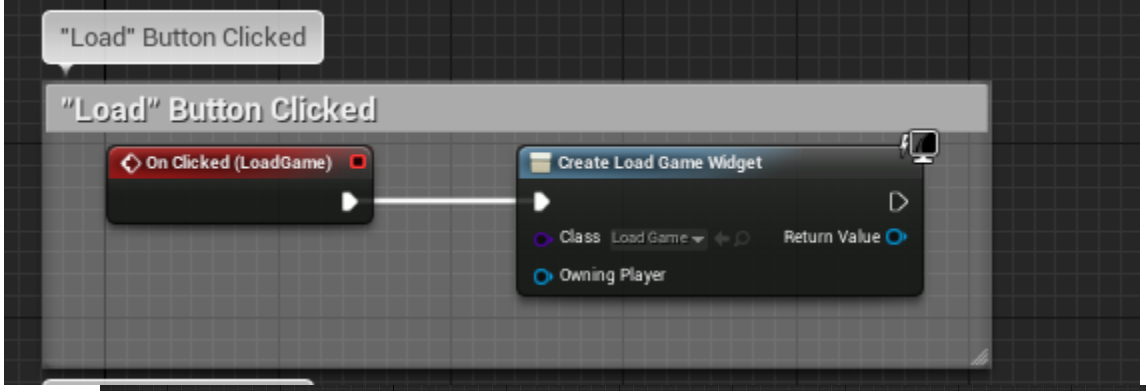
14. Appendix (Code)

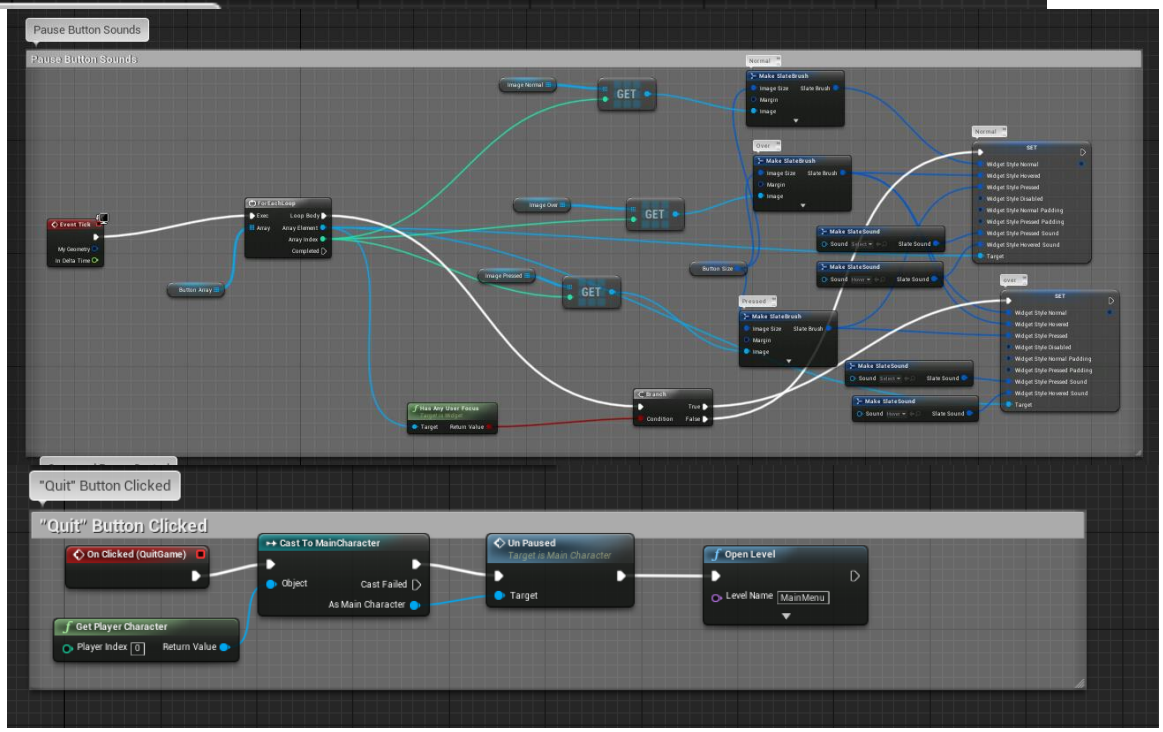
Main Menu

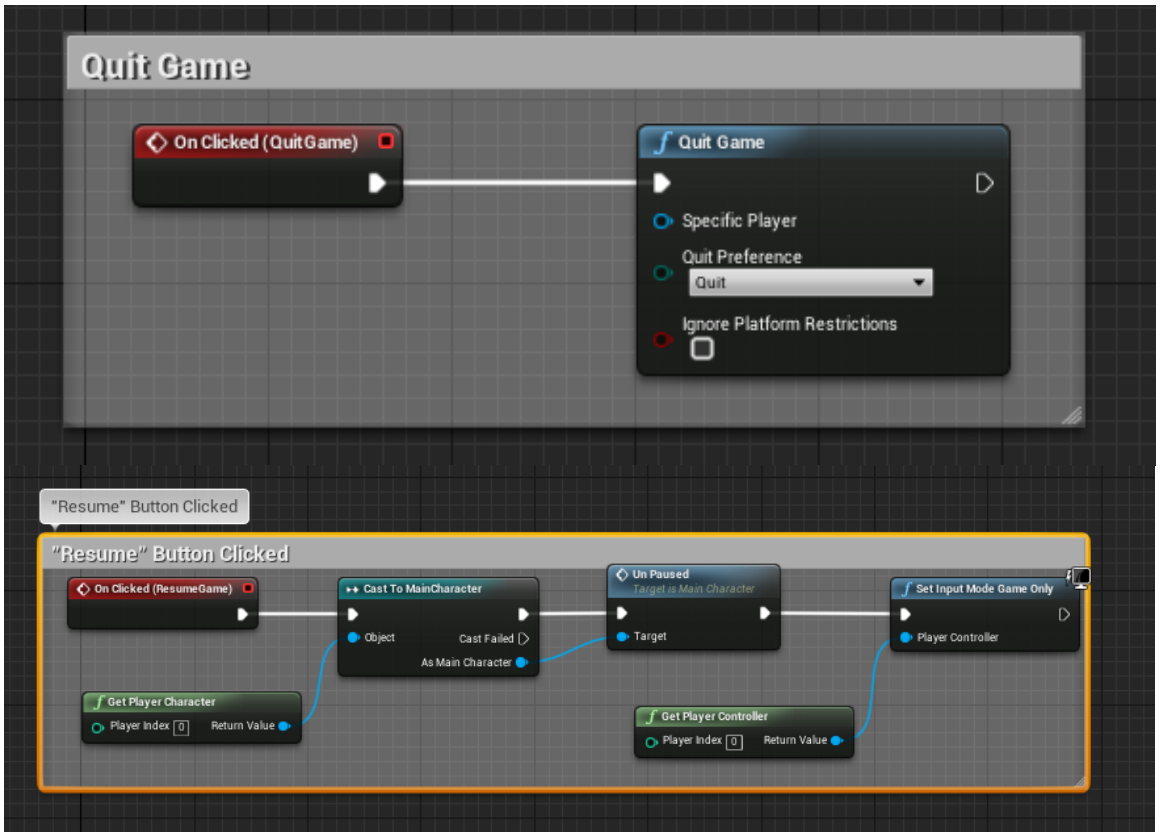


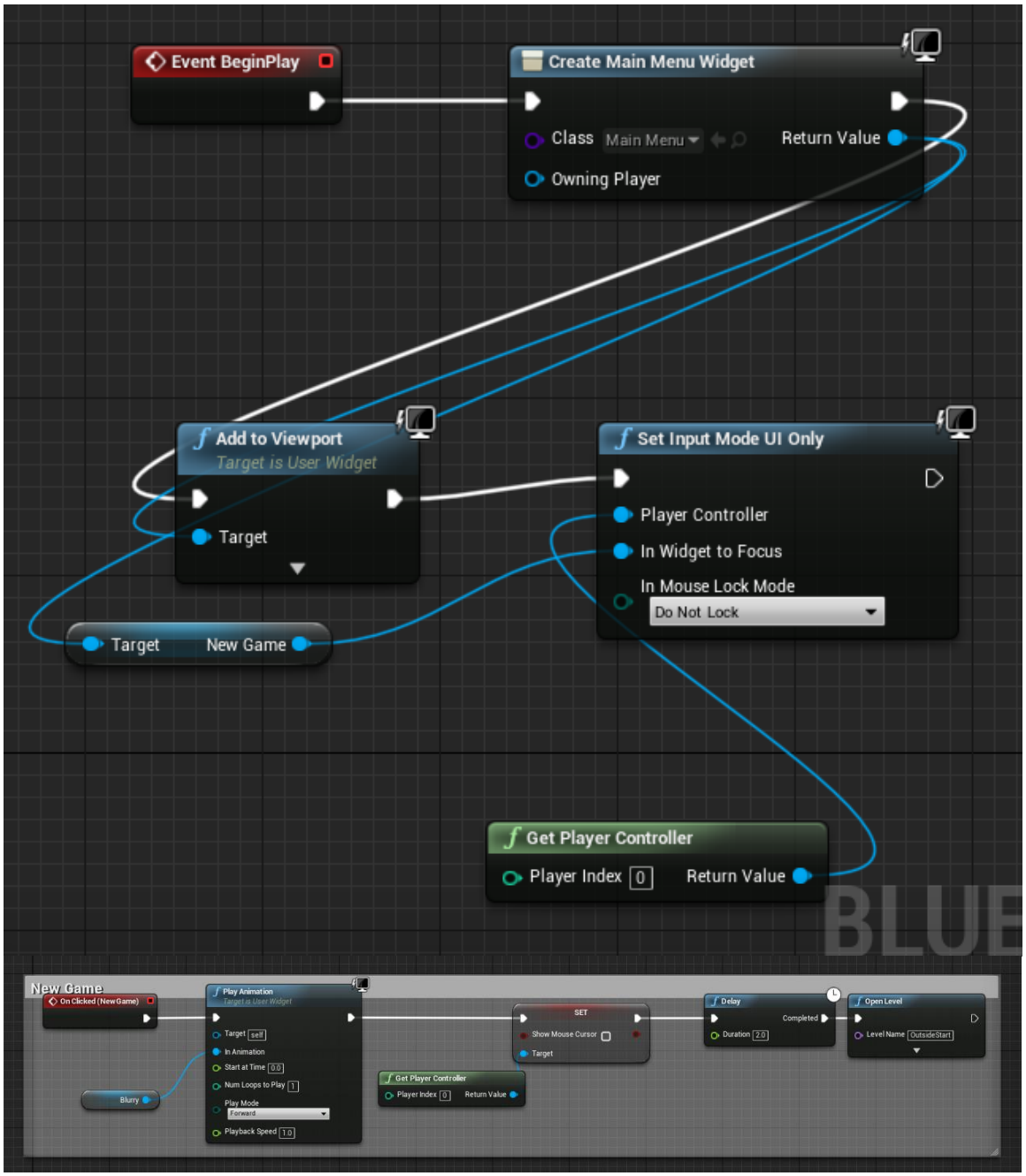

Camera Roll
Camera Roll
Event BeginPlay
SET
Show Mouse Cursor
Target
Play
Target is Matinee Actor
Target
Get Player Controller
Player Index 0
Return Value
MatineeActor
from Persistent Level
SET
Looping
Target
Play Sound 2D
Sound
Main_Menu
Gamepad Controls
Event Construct
SET
Button Index 0
SET
Button Array
New Game
Load Game
Quit Game
Make Array
[0]
[1]
[2]
Array
Add pin


Gamepad Pause Control
Gamepad Pause Control
Event Construct
SET
Button Index 0
SET
Button Array
Make Array
Resume Game
Load Game
Quit Game
[0]
[1]
[2]
Array
Add pin

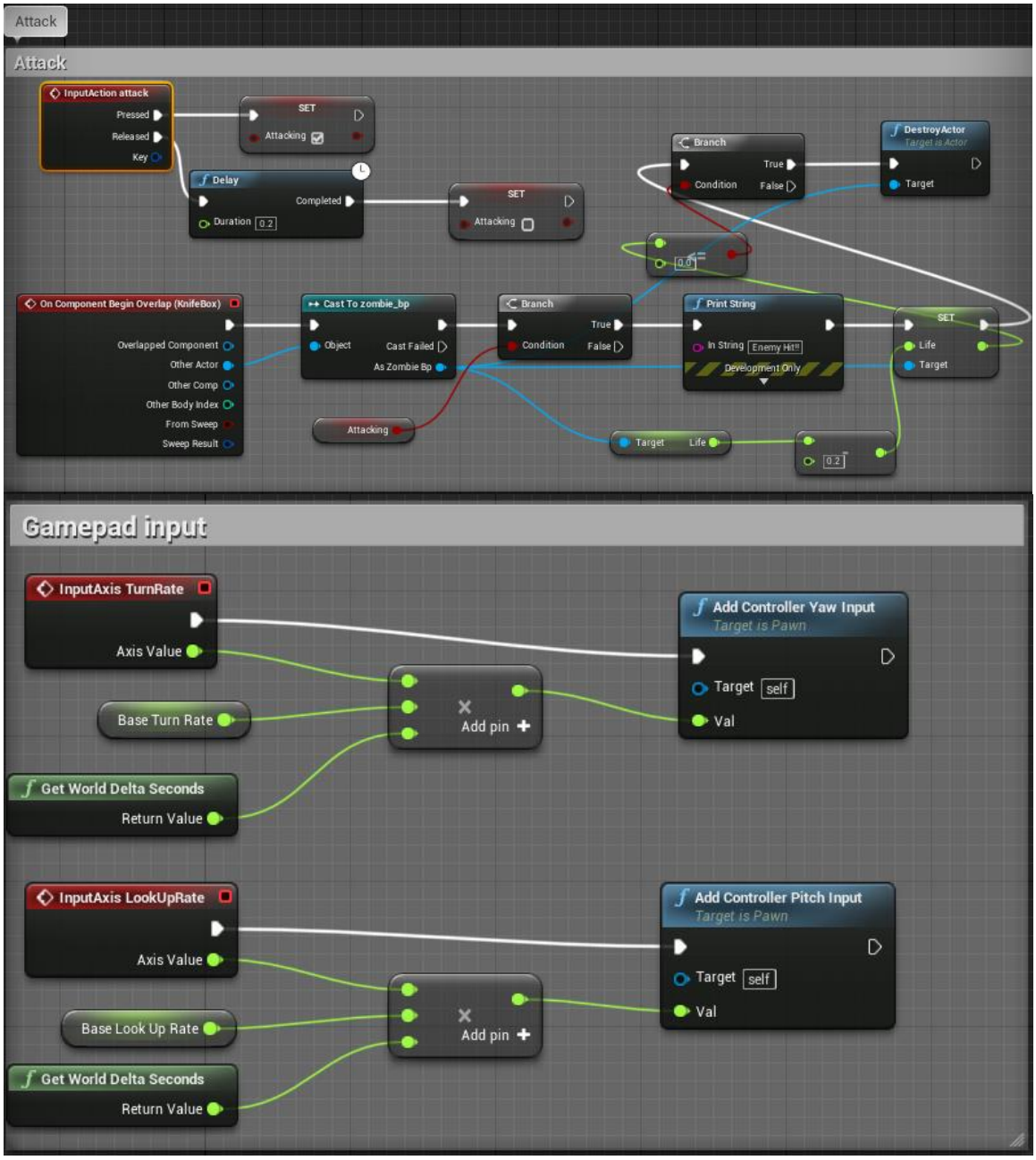

"Load" Button Clicked
"Load" Button Clicked
On Clicked (LoadGame)
Create Load Game Widget
Class
Load Game
Return Value
Owning Player

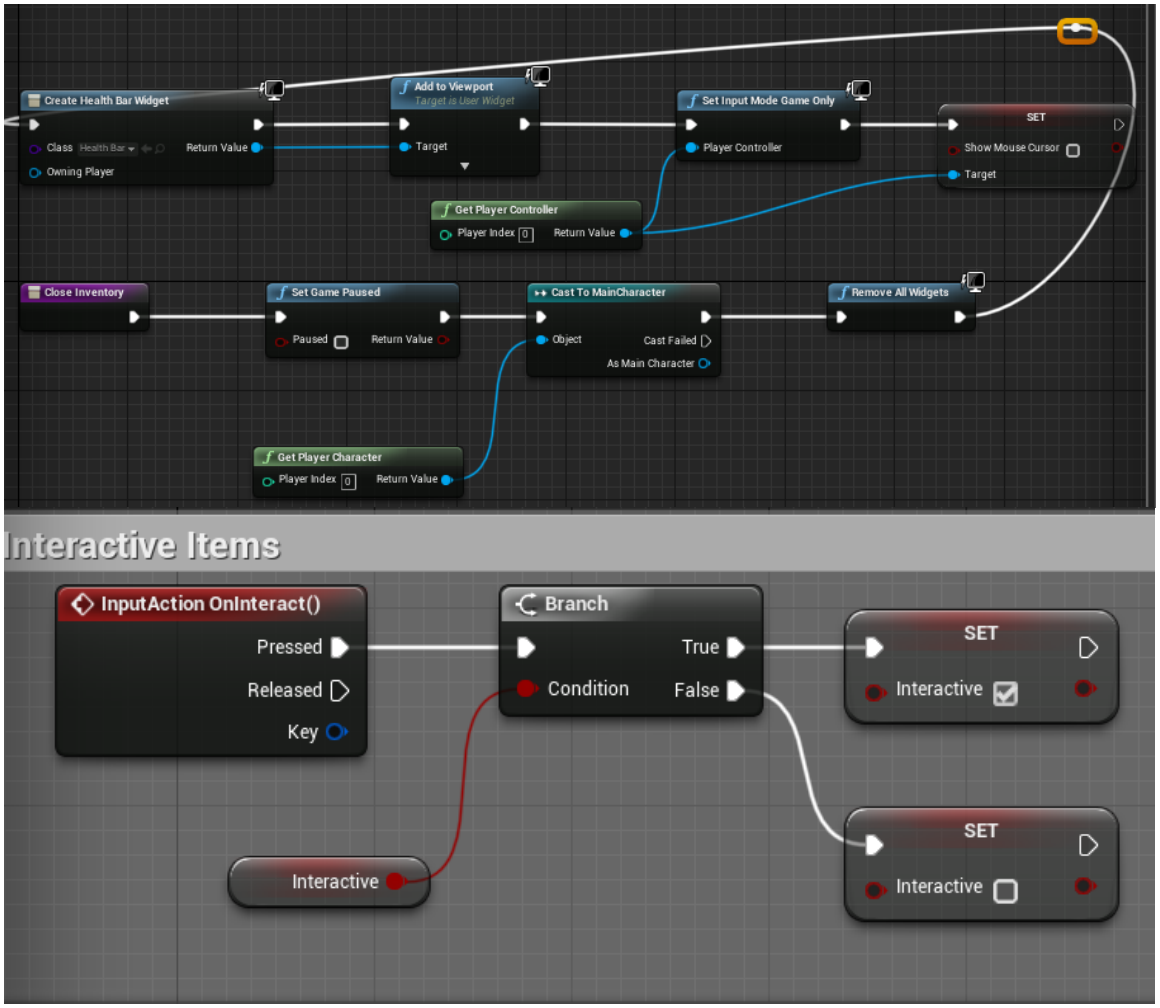

Pause Button Sounds
Pause Button Sounds
"Quit" Button Clicked
"Quit" Button Clicked
On Clicked (QuitGame)
Cast To MainCharacter
Object
Cast Failed
As Main Character
Un Paused
Target is Main Character
Target
Open Level
Level Name
MainMenu
Get Player Character
Player Index 0
Return Value

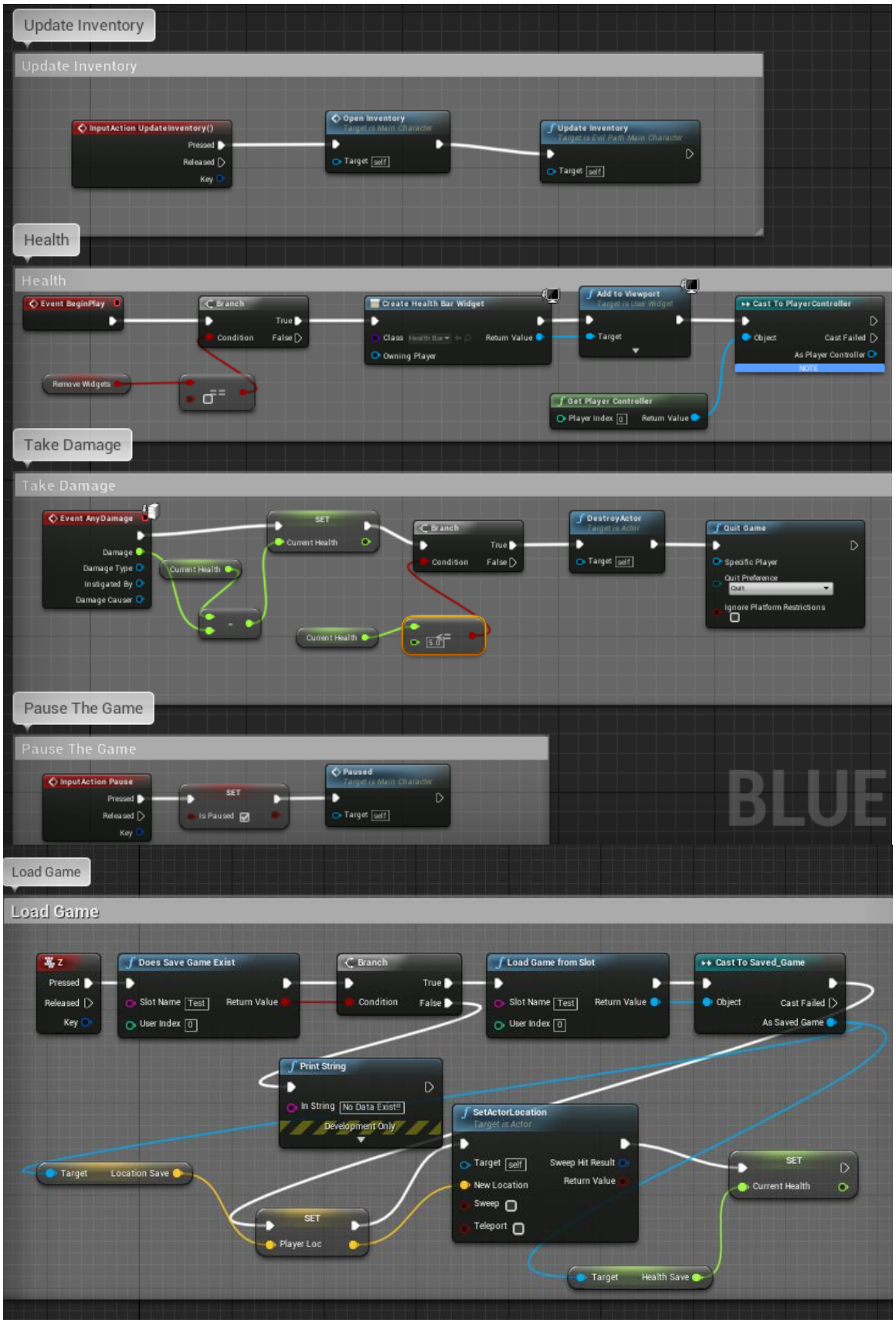

Quit Game
On Clicked (QuitGame)
Quit Game
Specific Player
Quit Preference
Quit
Ignore Platform Restrictions
"Resume" Button Clicked
"Resume" Button Clicked
On Clicked (ResumeGame)
Cast To MainCharacter
Object
Cast Failed
As Main Character
Un Paused
Target is Main Character
Target
Set Input Mode Game Only
Player Controller
Get Player Character
Player Index 0
Return Value
Get Player Controller
Player Index 0
Return Value

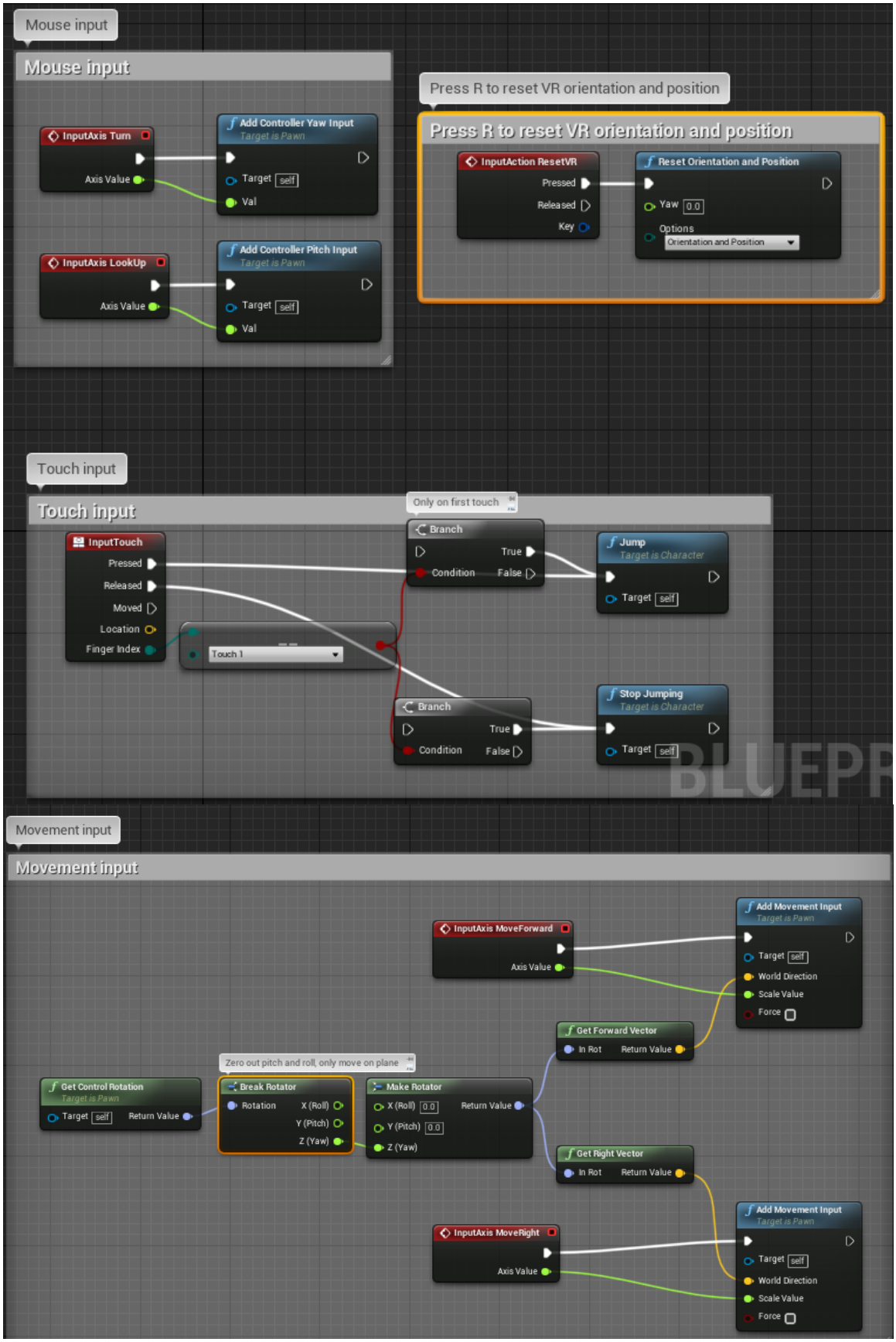

Event BeginPlay
Create Main Menu Widget
Class
Main Menu
Return Value
Owning Player
Add to Viewport
Target is User Widget
Target
Set Input Mode UI Only
Player Controller
In Widget to Focus
In Mouse Lock Mode
Do Not Lock
Target
New Game
Get Player Controller
Player Index
0
Return Value
New Game
On Clicked (New Game)
Play Animation
Target is User Widget
Target
self
In Animation
Start at Time
0.0
Num Loops to Play
1
Play Mode
Forward
Playback Speed
1.0
Blurry
SET
Show Mouse Cursor
Target
Get Player Controller
Player Index
0
Return Value
Delay
Completed
Duration
2.0
Open Level
Level Name
OutsideStart

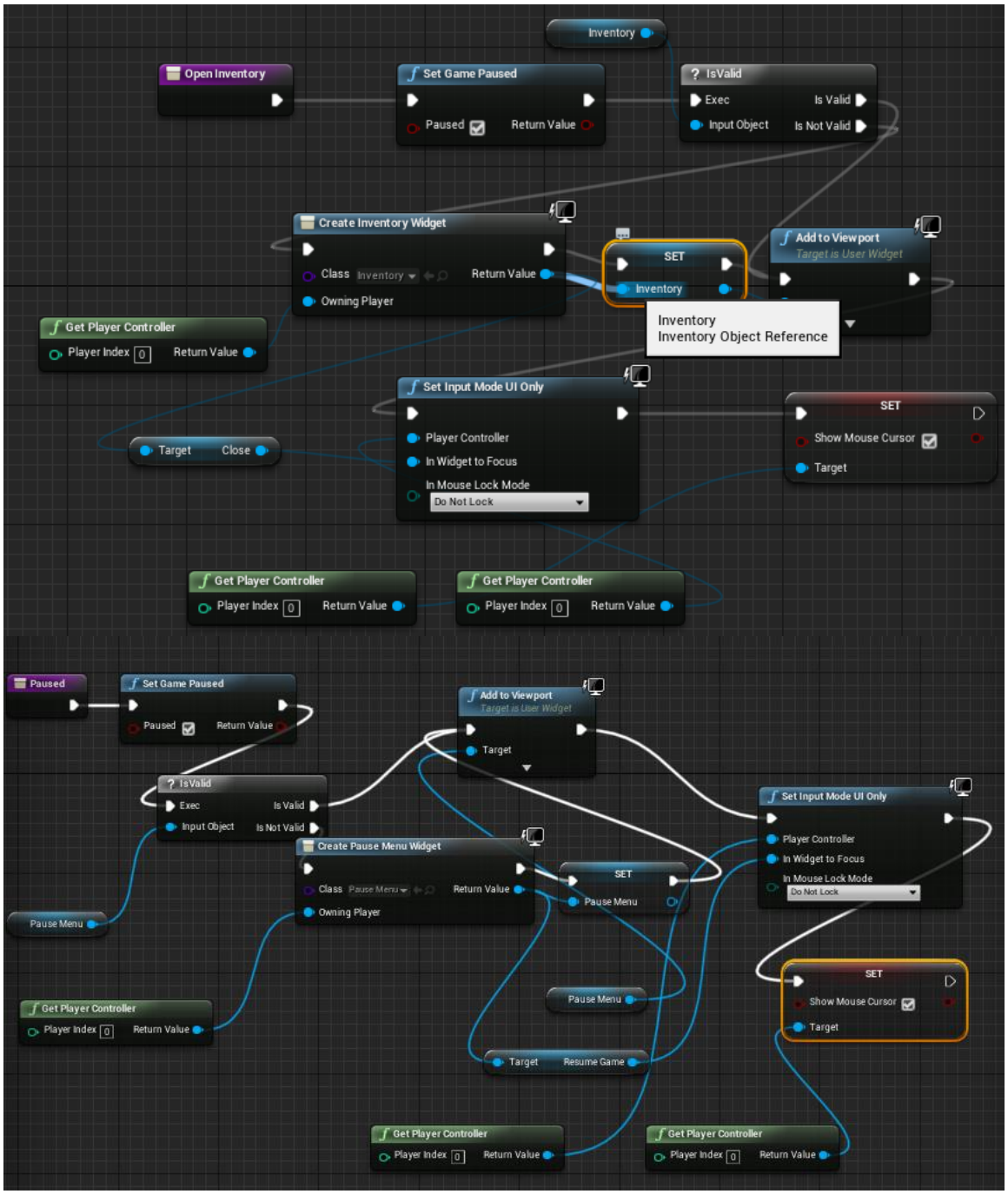
Main Player Character

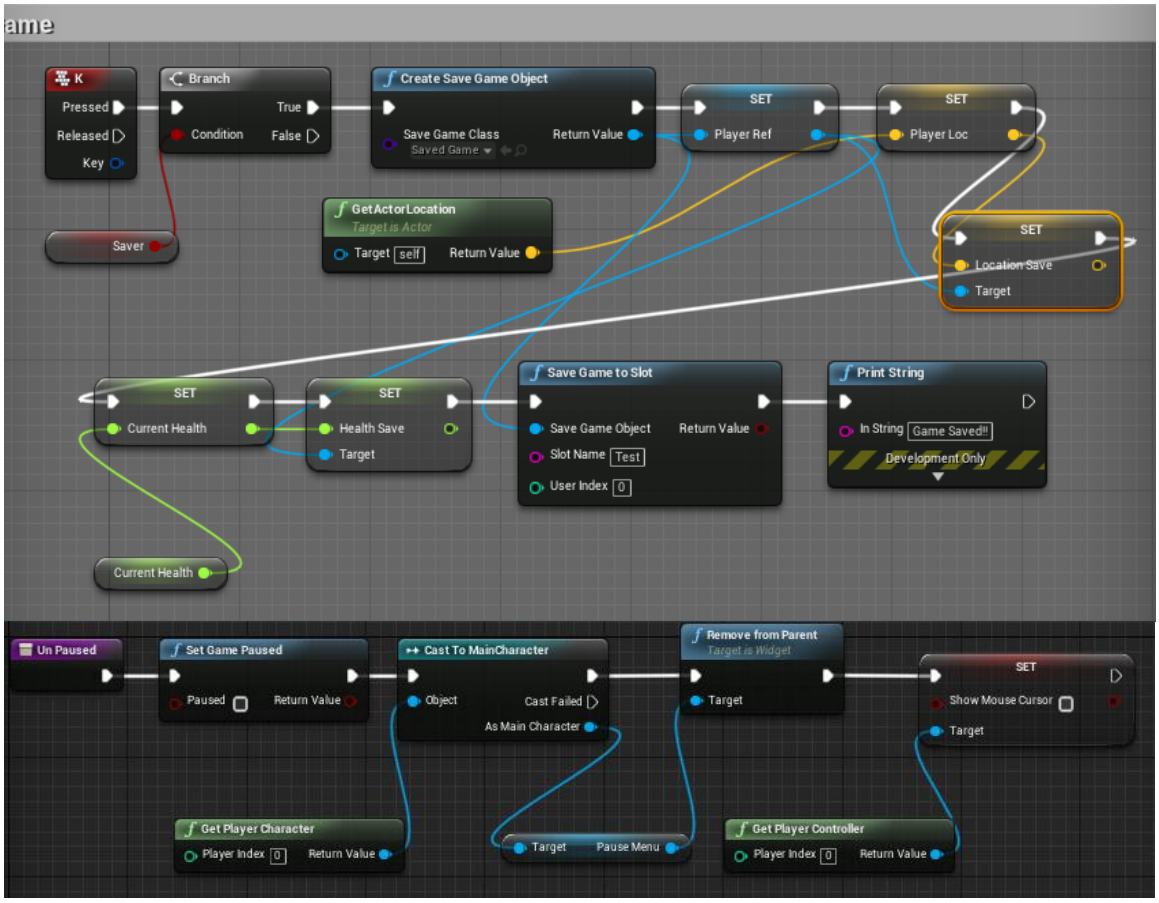



Create Health Bar Widget
Class Health Bar
Owning Player
Return Value
Add to Viewport
Target is User Widget
Target
Set Input Mode Game Only
Player Controller
SET
Show Mouse Cursor
Target
Get Player Controller
Player Index 0
Return Value
Close Inventory
Set Game Paused
Paused
Return Value
Cast To MainCharacter
Object
Cast Failed
As Main Character
Remove All Widgets
Get Player Character
Player Index 0
Return Value
Interactive Items
InputAction OnInteract()
Pressed
Released
Key
Branch
Condition
True
False
SET
Interactive
SET
Interactive
Interactive

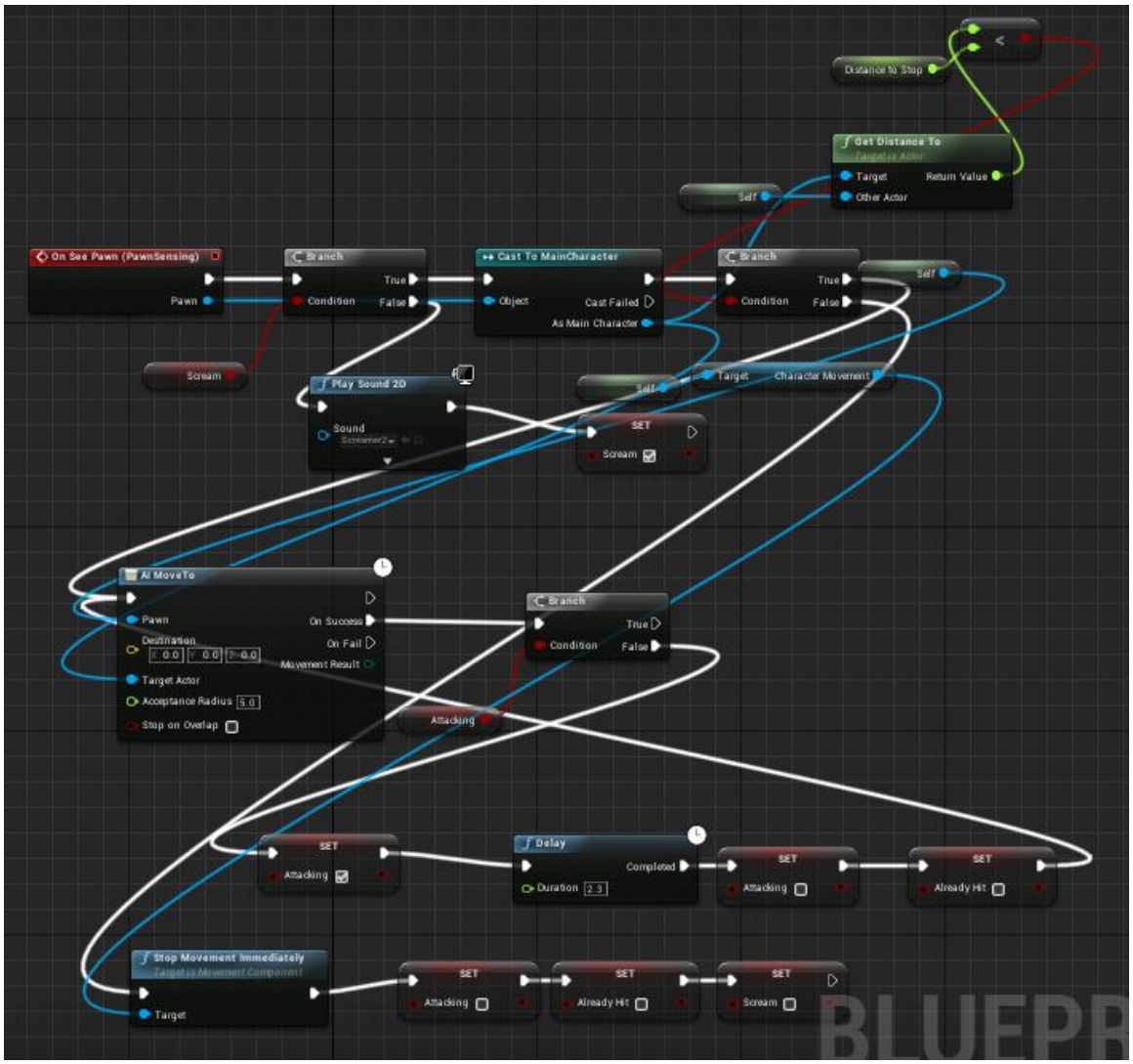

Update Inventory
Update Inventory
InputAction UpdateInventory()
Pressed
Released
Key
Open Inventory
Target is Main Character
Target
self
Update Inventory
Target is Evil Path Main Character
Target
self
Health
Health
Event BeginPlay
Branch
Condition
True
False
Create Health Bar Widget
Class
Owning Player
Return Value
Add to Viewport
Target is User Widget
Target
Cast To PlayerController
Object
Cast Failed
As Player Controller
NOTE
Remove Widgets
Get Player Controller
Player Index
0
Return Value
Take Damage
Take Damage
Event AnyDamage
Damage
Damage Type
Instigated By
Damage Causer
SET
Current Health
Branch
Condition
True
False
DestroyActor
Target is Actor
Target
self
Quit Game
Specific Player
Quit Preference
Quit
Ignore Platform Restrictions
Current Health
5.0
Pause The Game
Pause The Game
InputAction Pause
Pressed
Released
Key
SET
Is Paused
Paused
Target is Main Character
Target
self
BLUE
Load Game
Load Game
Z
Pressed
Released
Key
Does Save Game Exist
Slot Name
Test
User Index
0
Return Value
Branch
Condition
True
False
Load Game from Slot
Slot Name
Test
User Index
0
Return Value
Cast To Saved_Game
Object
Cast Failed
As Saved Game
Print String
In String
No Data Exist!!
Development Only
SetActorLocation
Target is Actor
Target
self
New Location
Sweep
Teleport
Sweep Hit Result
Return Value
SET
Current Health
Target
Location Save
SET
Player Loc
Target
Health Save

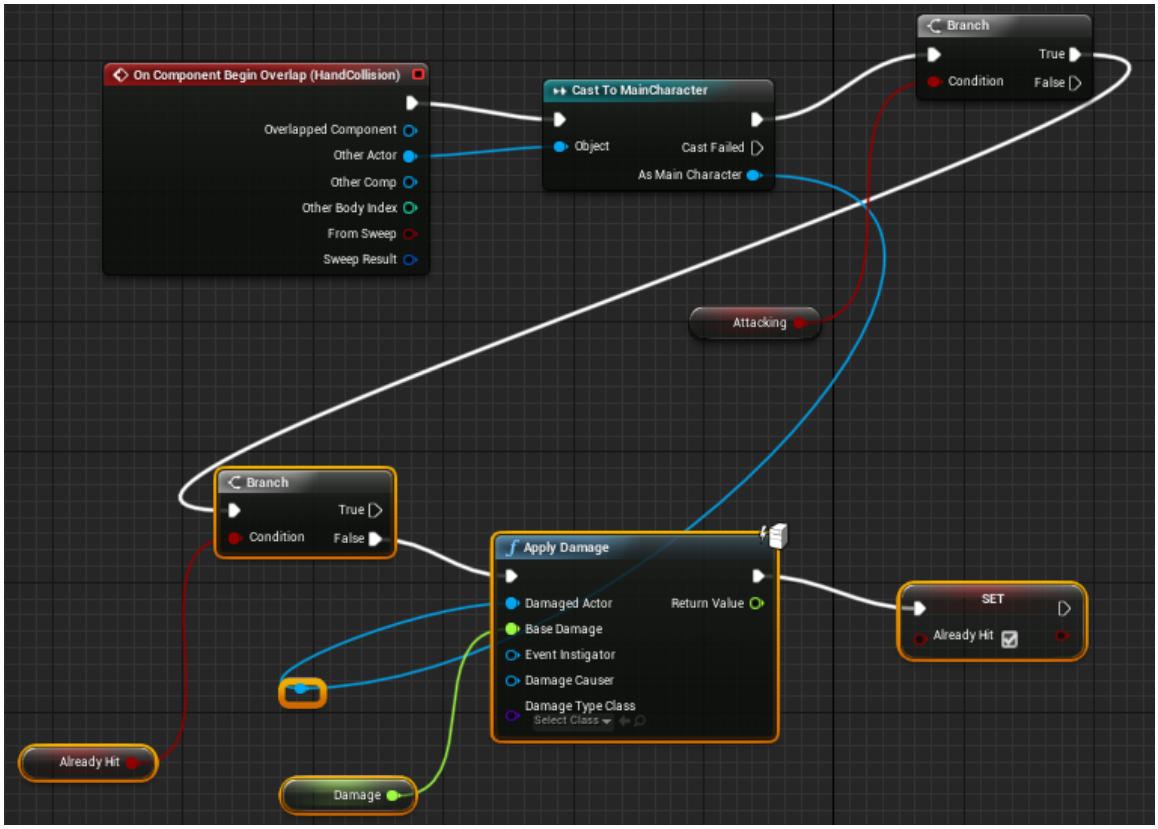

Mouse input
Mouse input
InputAxis Turn
Axis Value
Add Controller Yaw Input
Target is Pawn
Target self
Val
InputAxis LookUp
Axis Value
Add Controller Pitch Input
Target is Pawn
Target self
Val
Press R to reset VR orientation and position
Press R to reset VR orientation and position
InputAction ResetVR
Pressed
Released
Key
Reset Orientation and Position
Yaw 0.0
Options
Orientation and Position
Touch input
Touch input
Only on first touch
InputTouch
Pressed
Released
Moved
Location
Finger Index
Touch 1
Branch
True
Condition
False
Jump
Target is Character
Target self
Branch
True
Condition
False
Stop Jumping
Target is Character
Target self
Movement input
Movement input
InputAxis MoveForward
Axis Value
Add Movement Input
Target is Pawn
Target self
World Direction
Scale Value
Force
Get Forward Vector
In Rot
Return Value
Zero out pitch and roll, only move on plane
Get Control Rotation
Target is Pawn
Target self
Return Value
Break Rotator
Rotation
X (Roll)
Y (Pitch)
Z (Yaw)
Make Rotator
X (Roll) 0.0
Y (Pitch) 0.0
Z (Yaw)
Return Value
Get Right Vector
In Rot
Return Value
InputAxis MoveRight
Axis Value
Add Movement Input
Target is Pawn
Target self
World Direction
Scale Value
Force


Inventory
Open Inventory
Set Game Paused
Paused
Return Value
IsValid
Exec
Input Object
Is Valid
Is Not Valid
Create Inventory Widget
Class
Inventory
Owning Player
SET
Add to Viewport
Target is User Widget
Get Player Controller
Player Index
0
Inventory
Inventory Object Reference
Set Input Mode UI Only
Player Controller
In Widget to Focus
In Mouse Lock Mode
Do Not Lock
Target
Close
Show Mouse Cursor
Paused
Pause Menu
Create Pause Menu Widget
Resume Game

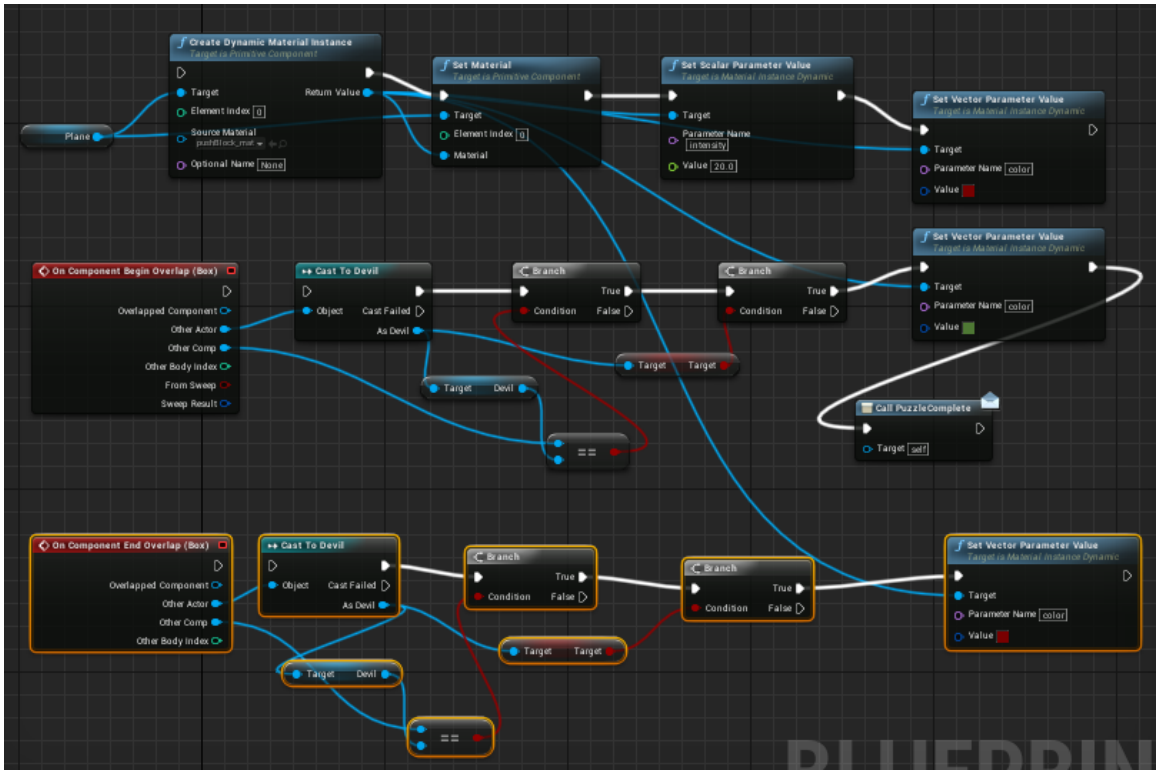

ame
K
Pressed
Released
Key
Branch
Condition
True
False
Create Save Game Object
Save Game Class
Saved Game
Return Value
SET
Player Ref
SET
Player Loc
Saver
GetActorLocation
Target is Actor
Target self
Return Value
SET
Location Save
Target
SET
Current Health
SET
Health Save
Target
Save Game to Slot
Save Game Object
Return Value
Slot Name Test
User Index 0
Print String
In String Game Saved!!
Development Only
Current Health
Un Paused
Set Game Paused
Paused
Return Value
Cast To MainCharacter
Object
Cast Failed
As Main Character
Remove from Parent
Target is Widget
Target
SET
Show Mouse Cursor
Target
Get Player Character
Player Index 0
Return Value
Target
Pause Menu
Get Player Controller
Player Index 0
Return Value

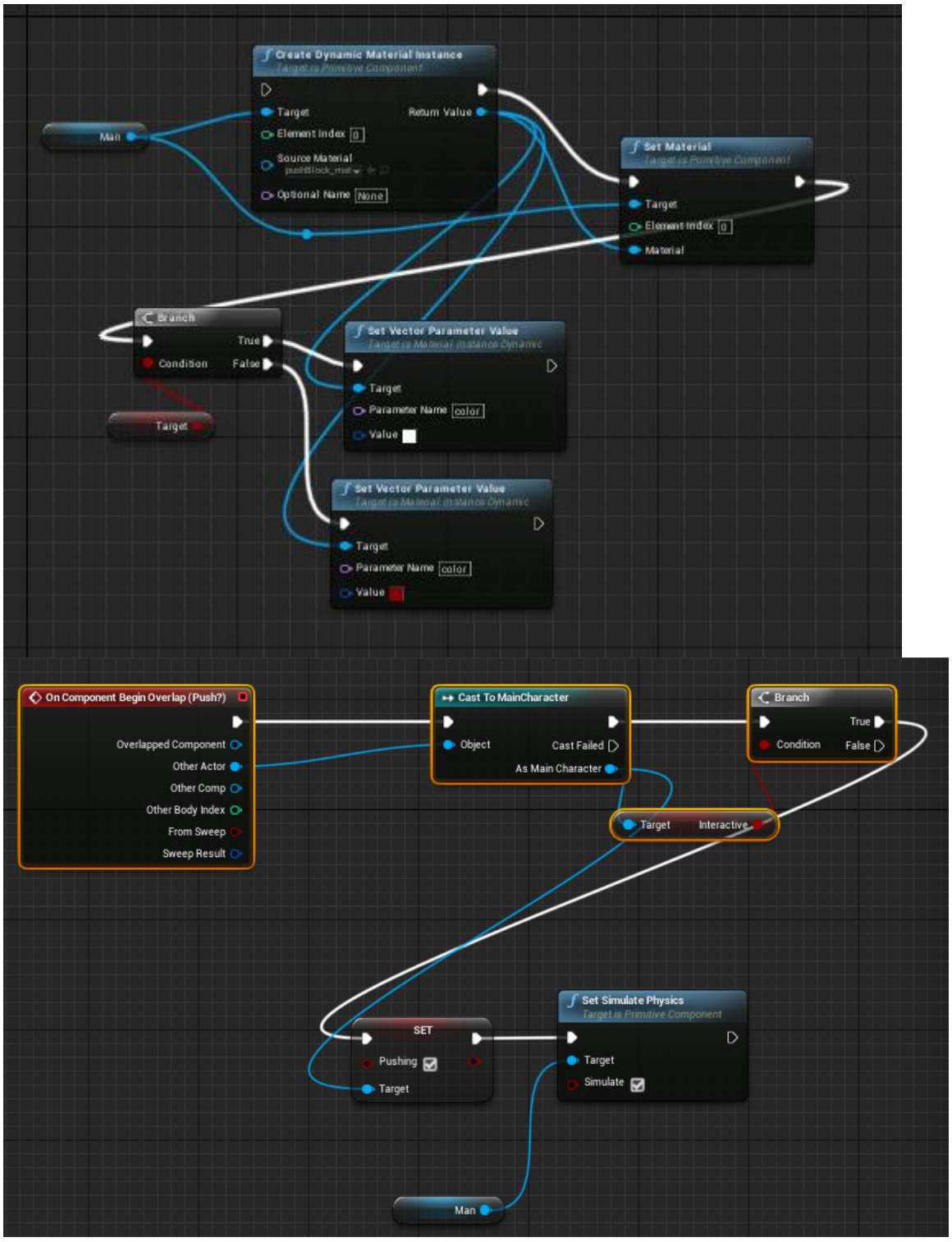
Zombie


Distance to Stop
Get Distance To
Target
Other Actor
Return Value
Self
On See Pawn (PawnSensing)
Pawn
Branch
Condition
True
False
Cast To MainCharacter
Object
Cast Failed
As Main Character
Scream
Play Sound 2D
Sound
SET
Target
Character Movement
AI MoveTo
Pawn
Destination
Target Actor
Acceptance Radius
Stop on Overlap
On Success
On Fail
Movement Result
Attacking
Delay
Completed
Duration
Already Hit
Stop Movement Immediately
BLUEPR

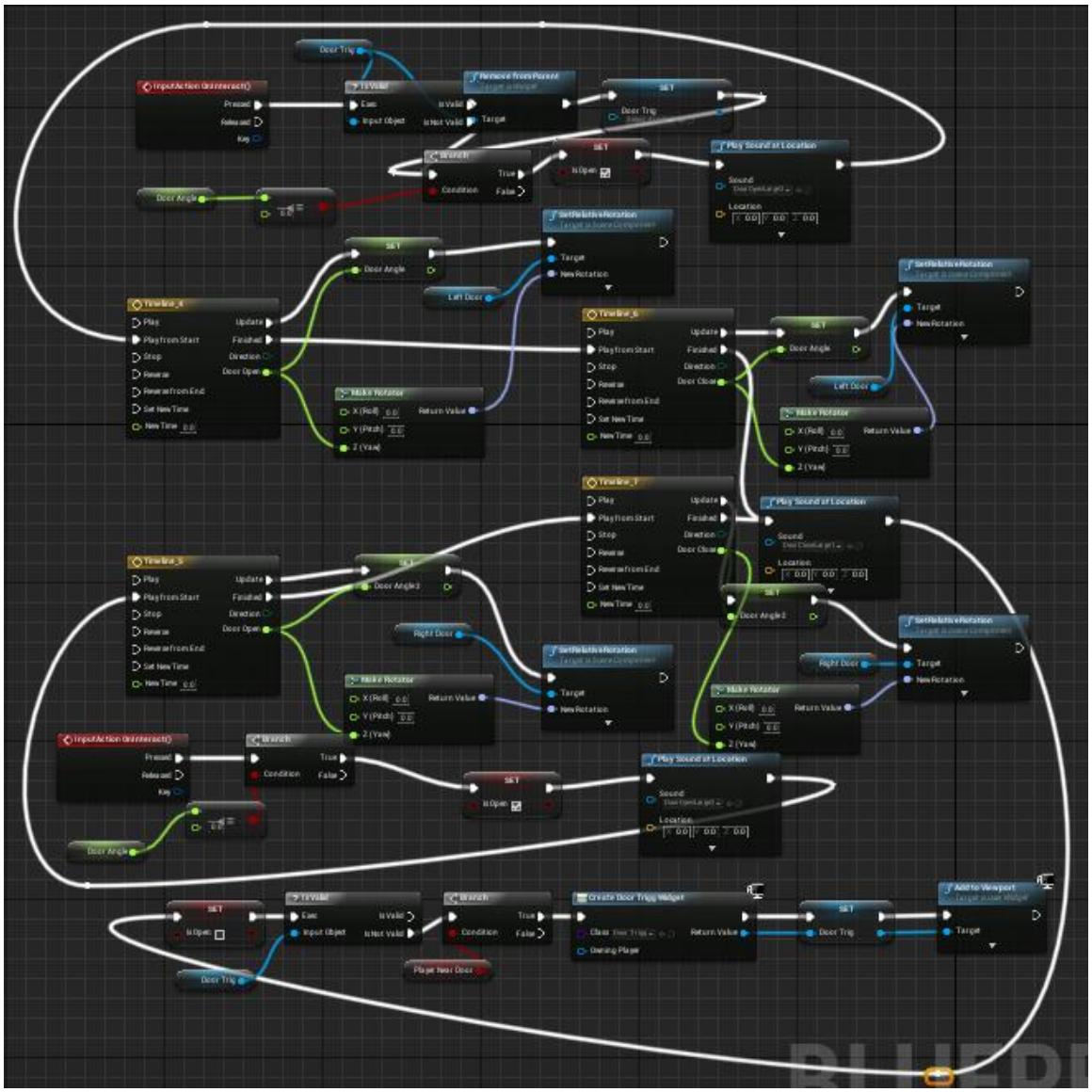

On Component Begin Overlap (HandCollision)
Overlapped Component
Other Actor
Other Comp
Other Body Index
From Sweep
Sweep Result
Cast To MainCharacter
Object
Cast Failed
As Main Character
Branch
True
Condition
False
Attacking
Branch
True
Condition
False
Apply Damage
Damaged Actor
Return Value
Base Damage
Event Instigator
Damage Causer
Damage Type Class
Select Class
SET
Already Hit
Already Hit
Damage

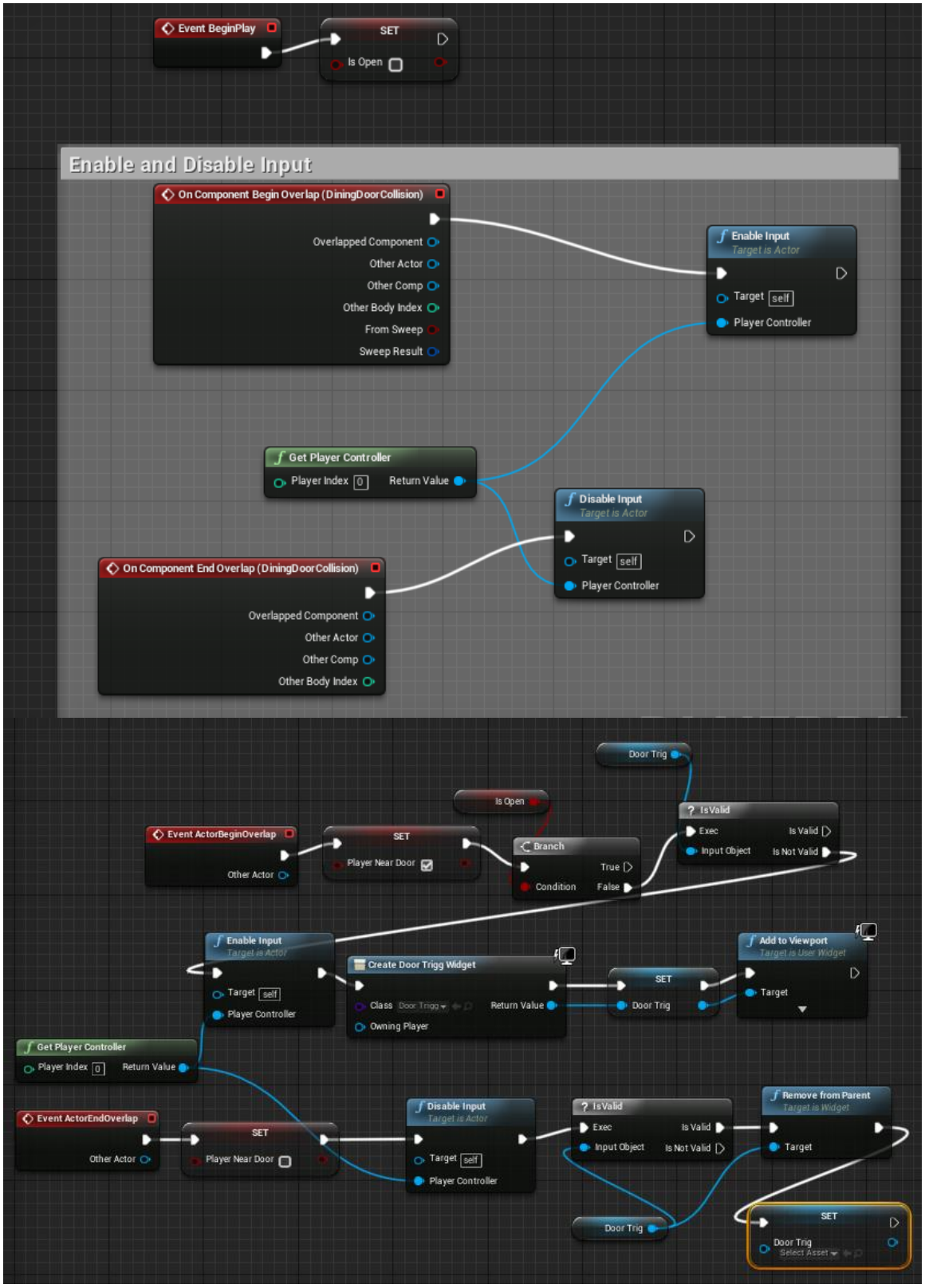
Puzzle


Create Dynamic Material Instance
Target is Primitive Component
Target
Element Index 0
Source Material
Optional Name None
Return Value
Plane
Set Material
Target is Primitive Component
Target
Element Index 0
Material
Set Scalar Parameter Value
Target is Material Instance Dynamic
Target
Parameter Name intensity
Value 20.0
Set Vector Parameter Value
Target is Material Instance Dynamic
Target
Parameter Name color
Value
On Component Begin Overlap (Box)
Overlapped Component
Other Actor
Other Comp
Other Body Index
From Sweep
Sweep Result
Cast To Devil
Object
Cast Failed
As Devil
Branch
True
Condition
False
Target
Devil
Call PuzzleComplete
Target self
On Component End Overlap (Box)
Overlapped Component
Other Actor
Other Comp
Other Body Index

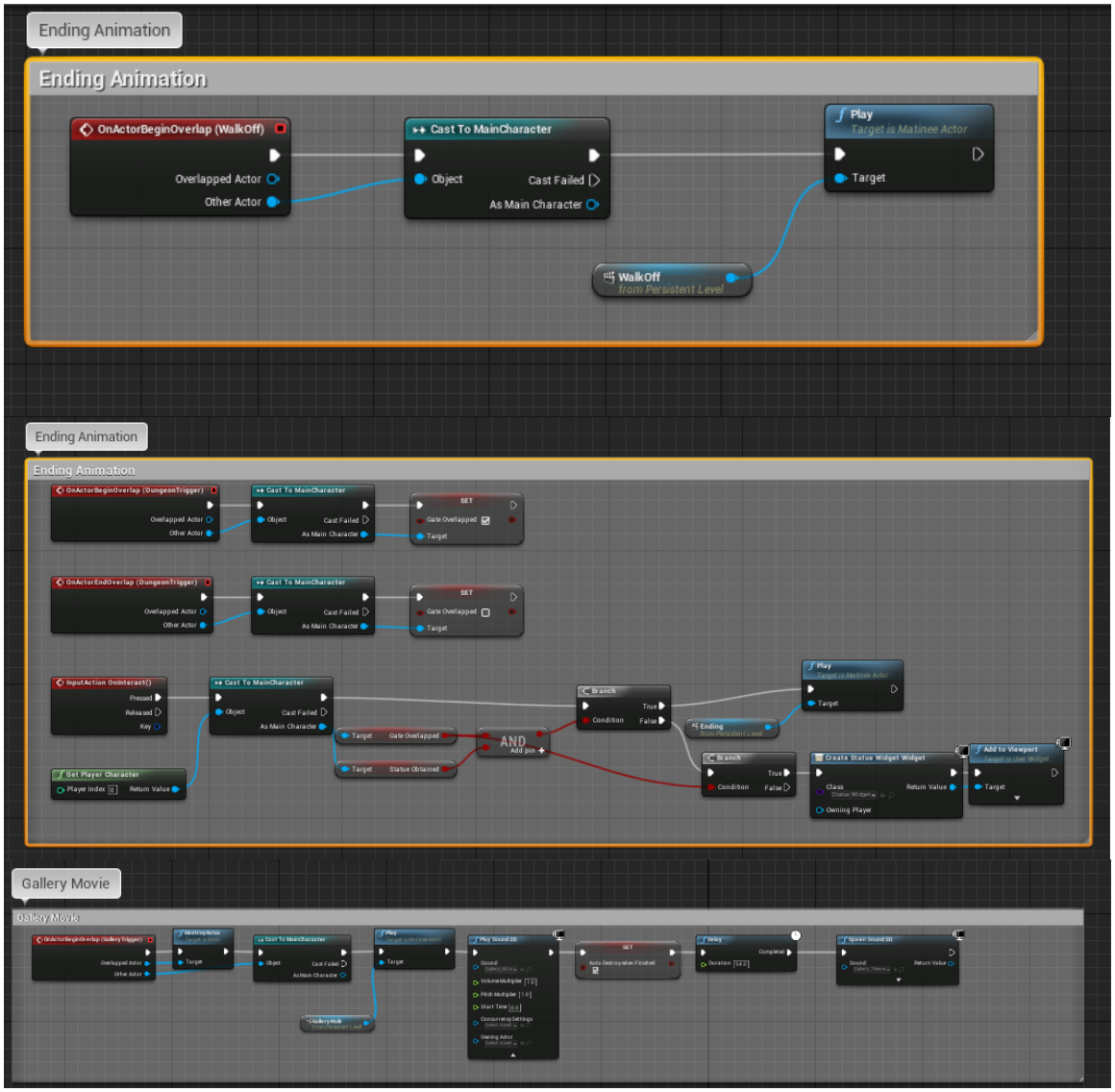

Create Dynamic Material Instance
Target is Primitive Component
Target
Element Index 0
Source Material
Optional Name None
Return Value
Man
Set Material
Target is Primitive Component
Target
Element Index 0
Material
Branch
Condition
True
False
Target
Set Vector Parameter Value
Target is Material Instance Dynamic
Target
Parameter Name color
Value
Set Vector Parameter Value
Target is Material Instance Dynamic
Target
Parameter Name color
Value
On Component Begin Overlap (Push?)
Overlapped Component
Other Actor
Other Comp
Other Body Index
From Sweep
Sweep Result
Cast To MainCharacter
Object
Cast Failed
As Main Character
Branch
Condition
True
False
Target
Interactive
SET
Pushing
Target
Set Simulate Physics
Target is Primitive Component
Target
Simulate
Man

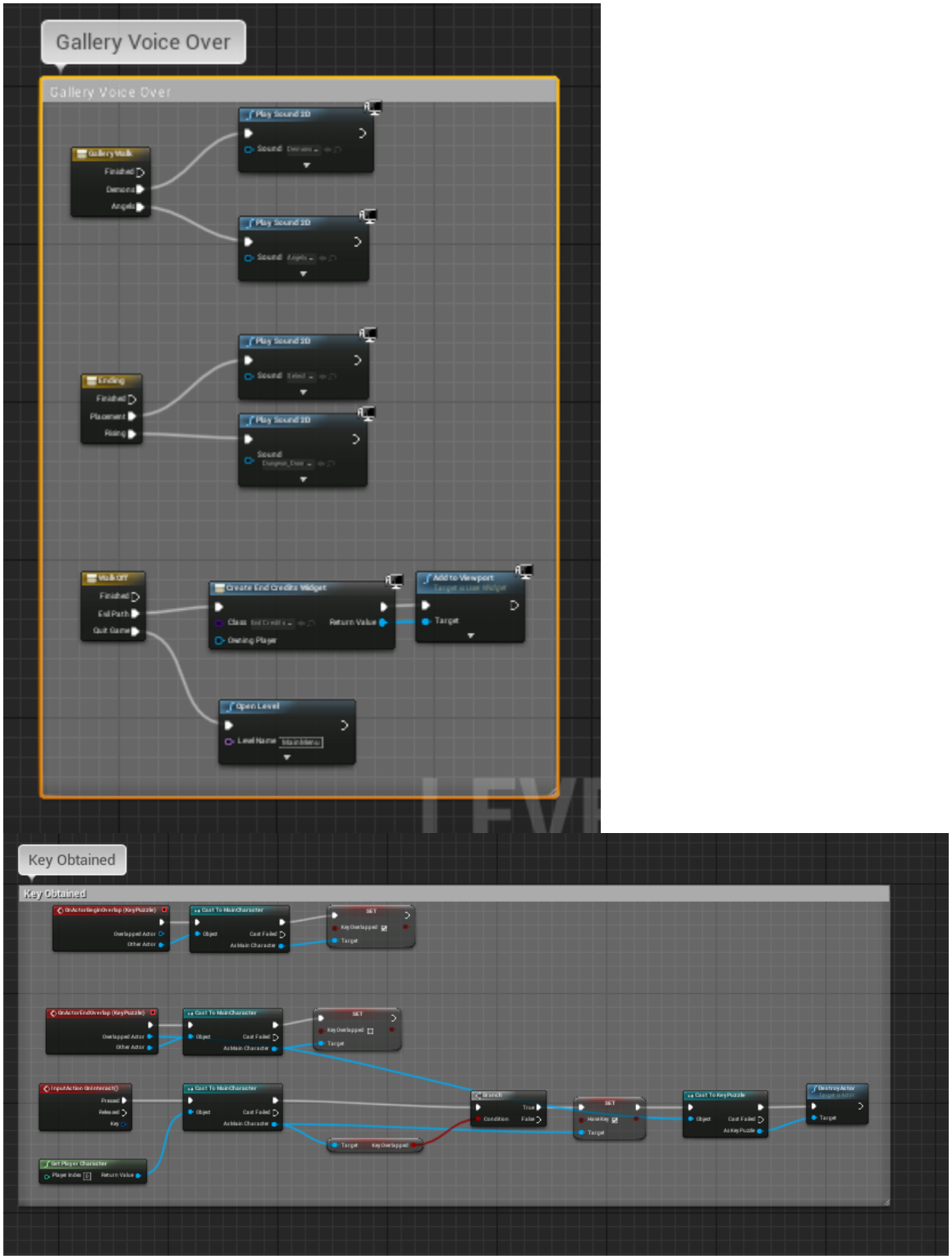
Door Mechanics

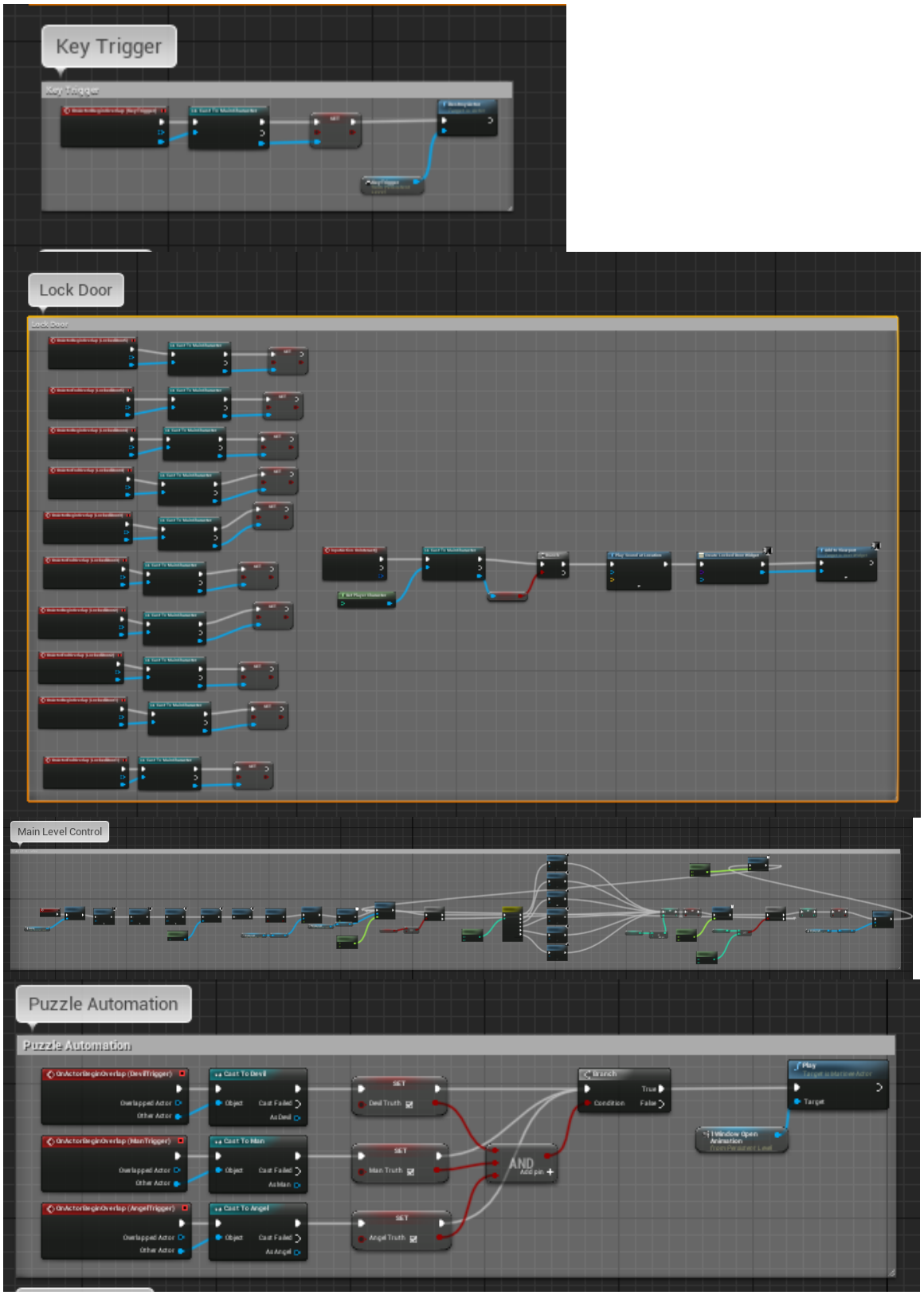


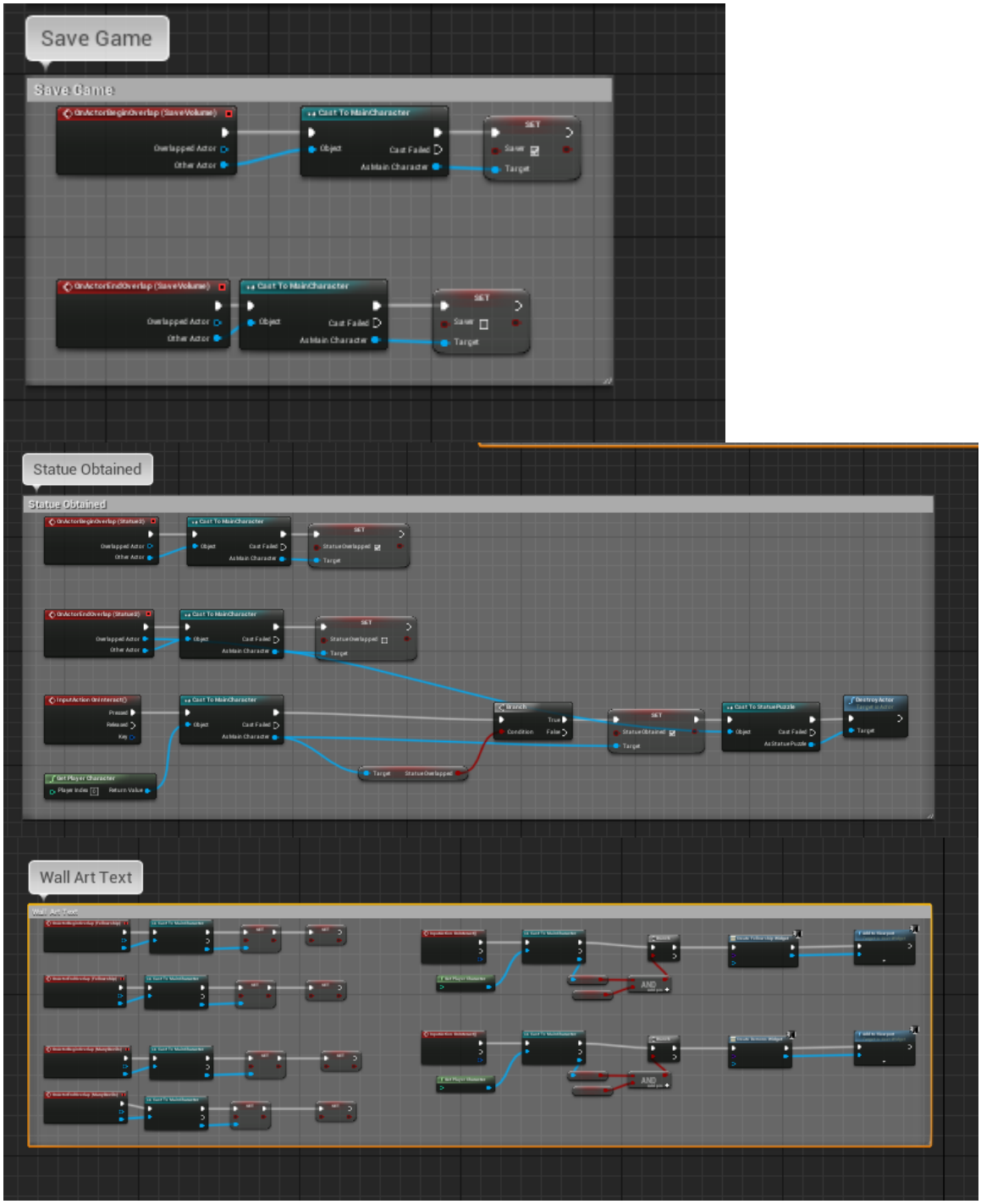

Event BeginPlay
SET
Is Open
Enable and Disable Input
On Component Begin Overlap (DiningDoorCollision)
Overlapped Component
Other Actor
Other Comp
Other Body Index
From Sweep
Sweep Result
Enable Input
Target is Actor
Target self
Player Controller
Get Player Controller
Player Index 0
Return Value
Disable Input
Target is Actor
Target self
Player Controller
On Component End Overlap (DiningDoorCollision)
Overlapped Component
Other Actor
Other Comp
Other Body Index
Door Trig
Is Open
Event ActorBeginOverlap
Other Actor
SET
Player Near Door
Branch
True
Condition
False
IsValid
Exec
Input Object
Is Valid
Is Not Valid
Enable Input
Target is Actor
Target self
Player Controller
Create Door Trigg Widget
Class Door Trigg
Owning Player
Return Value
SET
Door Trig
Add to Viewport
Target is User Widget
Target
Get Player Controller
Player Index 0
Return Value
Event ActorEndOverlap
Other Actor
SET
Player Near Door
Disable Input
Target is Actor
Target self
Player Controller
IsValid
Exec
Input Object
Is Valid
Is Not Valid
Remove from Parent
Target is Widget
Target
Door Trig
SET
Door Trig
Select Asset

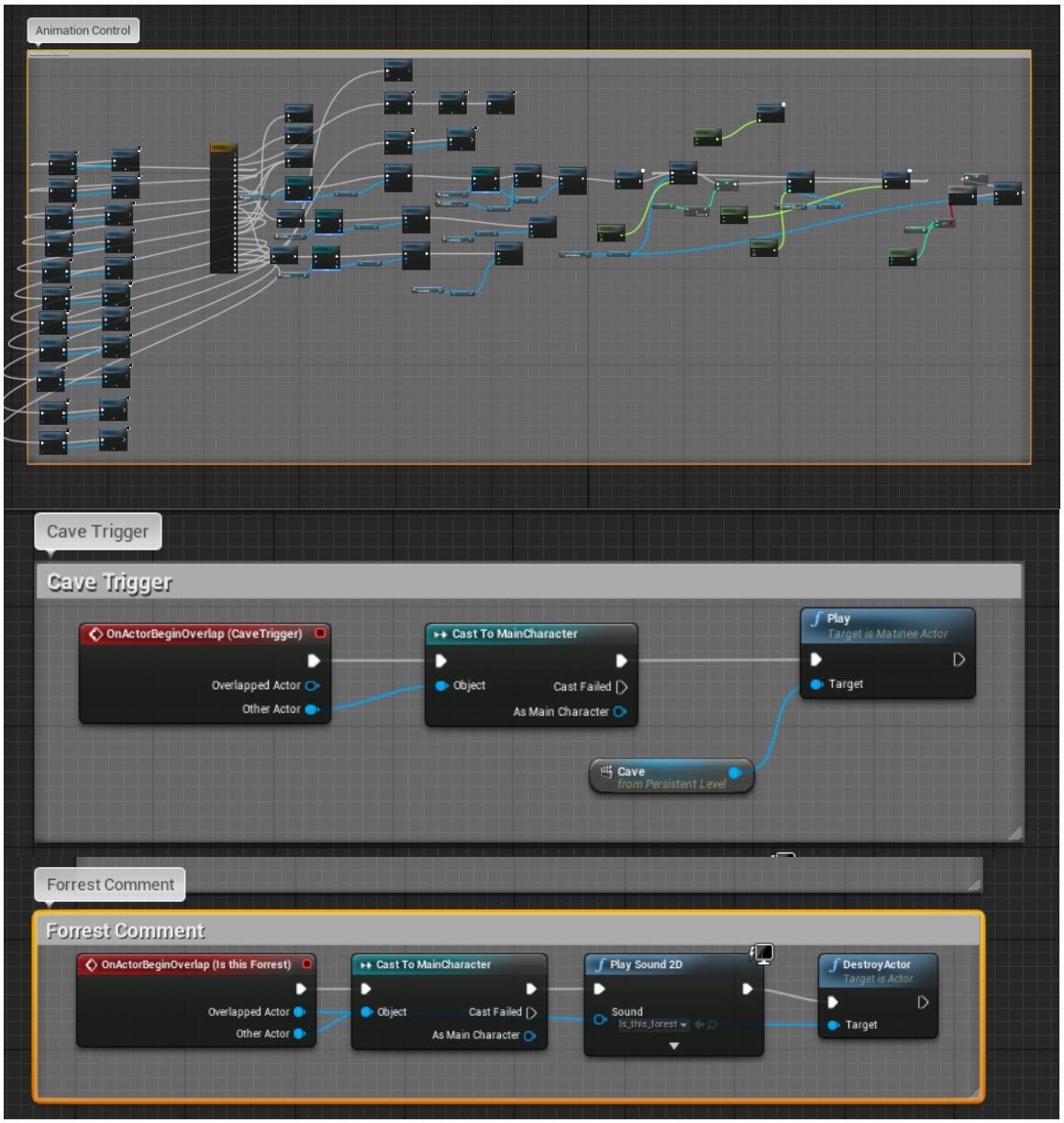
Level Blueprints

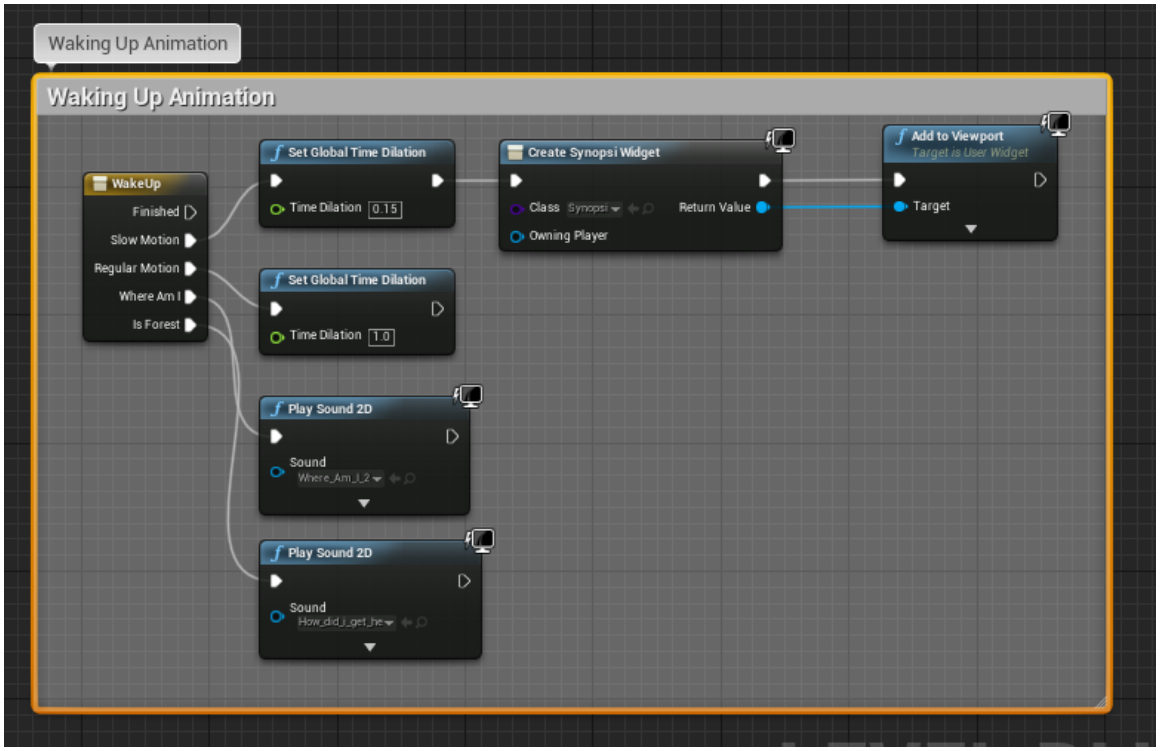



Gallery Voice Over
Gallery Voice Over
Play Sound 2D
Gallery Walk
Finished
Demons
Angels
Play Sound 2D
Sound
Play Sound 2D
Ending
Finished
Placement
Rising
Play Sound 2D
Sound
Walk Off
Finished
Exit Path
Quit Game
Create End Credits Widget
Class
Return Value
Owning Player
Add to Viewport
Target
Open Level
Level Name
Key Obtained
Key Obtained


Key Trigger
Lock Door
Main Level Control
Puzzle Automation
AND


Save Game
Save Game
Statue Obtained
Statue Obtained
Wall Art Text


Animation Control
Cave Trigger
Cave Trigger
OnActorBeginOverlap (CaveTrigger)
Overlapped Actor
Other Actor
Cast To MainCharacter
Object
Cast Failed
As Main Character
Play
Target is Matinee Actor
Target
Cave
from Persistent Level
Forrest Comment
Forrest Comment
OnActorBeginOverlap (Is this Forrest)
Overlapped Actor
Other Actor
Cast To MainCharacter
Object
Cast Failed
As Main Character
Play Sound 2D
Sound
Is_this_forest
DestroyActor
Target is Actor
Target



C++ Inventory Code

Inventory Items (.h file)

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Components/BoxComponent.h"
#include "ItemPickup.generated.h"

UCLASS()
class MYPROJECT_API AItemPickup : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AItemPickup();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

	virtual void Show(bool visible);

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	//SceneCoomponent as Root
```

```
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Pickup)
		USceneComponent* SceneComponent;

	// MeshComponent

	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Pickup)
		UStaticMeshComponent* ItemMesh;

	UPROPERTY(VisibleAnywhere)
		UBoxComponent* BoxCollider;

	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Pickup)
		bool canPickup;

	UFUNCTION(BlueprintCallable)
	virtual void OnInteract();

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Pickup)
		FString Name;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Pickup)
		UTexture2D* Image;

	UFUNCTION()
		void OnOverlapBegin(UPrimitiveComponent* OverlappedComp, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep,
const FHitResult& SweepResult);

	UFUNCTION()
		void OnOverlapEnd(class UPrimitiveComponent* OverlappedComp, class
AActor* OtherActor, class UPrimitiveComponent* OtherComp, int32 OtherBodyIndex);
};
```

Inventory Items (Cpp file)

```
#include "ItemPickup.h"
#include "Components/StaticMeshComponent.h"
#include "Engine.h"
#include "EvilPathMainCharacter.h"

// Sets default values
AItemPickup::AItemPickup()
{
	// Set this actor to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	this->SceneComponent =
CreateDefaultSubobject<USceneComponent>(TEXT("SceneComponent"));

	this->RootComponent = SceneComponent;

	this->ItemMesh =
CreateDefaultSubobject<UStaticMeshComponent>(TEXT("ItemMesh"));
```

```
	//this->ItemMesh->AttachToComponent(this->RootComponent,
FAttachmentTransformRules::KeepWorldTransform);
	this->RootComponent->SetupAttachment(this->ItemMesh);

	// Creating Collider
	this->BoxCollider =
CreateDefaultSubobject<UBoxComponent>(TEXT("BoxCollider"));
	this->BoxCollider->SetGenerateOverlapEvents(true);
	this->BoxCollider->SetWorldScale3D(FVector(1.0f, 1.0f, 1.0f));
	this->BoxCollider->OnComponentBeginOverlap.AddDynamic(this,
&AItemPickup::OnOverlapBegin);
	//this->BoxCollider->AttachToComponent(this->RootComponent,
FAttachmentTransformRules::KeepWorldTransform);
	this->RootComponent->SetupAttachment(this->BoxCollider);

	canPickup = false;
}

void AItemPickup::OnInteract()
{
	if (canPickup == true)
	{
		Name = GetName();
		FString pickup = FString::Printf(TEXT("Pickedup: %s"), *Name);

		AEvilPathMainCharacter* player =
Cast<AEvilPathMainCharacter>(UGameplayStatics::GetPlayerCharacter(this, 0));

		if (player && player->AddToInventory(this))
		{
			Show(false);
			GEngine->AddOnScreenDebugMessage(1, 3, FColor::Red, pickup);
		}
	}
	//Destroy();
}

//Overlap handler
void AItemPickup::OnOverlapBegin(UPrimitiveComponent* OverlappedComp, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep,
const FHitResult& SweepResult)
{
	//Check if the Other Actor is not me and if it is not Null
	if ((OtherActor != nullptr) && (OtherActor != this) && (OtherComp !=
nullptr))
	{
		canPickup = true;
	}
}

void AItemPickup::OnOverlapEnd(class UPrimitiveComponent* OverlappedComp, class
AActor* OtherActor, class UPrimitiveComponent* OtherComp, int32 OtherBodyIndex)
{
	canPickup = false;
}

// Called when the game starts or when spawned
```

```
void AItemPickup::BeginPlay()
{
	Super::BeginPlay();

}

void AItemPickup::Show(bool visible)
{
	ECollisionEnabled::Type collision = visible ?
		ECollisionEnabled::QueryAndPhysics :
		ECollisionEnabled::NoCollision;

	this->SetActorTickEnabled(visible);

	this->ItemMesh->SetVisibility(visible);
	this->ItemMesh->SetCollisionEnabled(collision);

	this->BoxCollider->SetCollisionEnabled(collision);

}

// Called every frame
void AItemPickup::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}
```

Connecting Inventory to Main Character (.h file)

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "ItemPickup.h"
#include "EvilPathMainCharacter.generated.h"


DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FUpdateInventoryDelegate, const
TArray<AItemPickup*>&, InventoryItems);

UCLASS()
class MYPROJECT_API AEvilPathMainCharacter : public ACharacter
{
	GENERATED_BODY()

private:
	TArray<AItemPickup*>_inventory;
	int sizeOfInventory;
	const int maxSizeOfInventory = 5;

public:
	bool AddToInventory(AItemPickup* actor);

	int getSizeOfInventory();

	UFUNCTION(BlueprintCallable)
```

```
	void UpdateInventory();

	UPROPERTY(BlueprintAssignable, Category =  "Pickup")
	FUpdateInventoryDelegate OnUpdateInventory;

public:
	// Sets default values for this character's properties
	AEvilPathMainCharacter();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	// Called to bind functionality to input
	virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

};
```

Connecting Inventory to Main Character (Cpp file)

```
#include "EvilPathMainCharacter.h"
#include "Engine.h"
#include "ItemPickup.h"

bool AEvilPathMainCharacter::AddToInventory(AItemPickup * actor)
{
	if (getSizeOfInventory() < maxSizeOfInventory)
	{
		_inventory.Add(actor);
		sizeOfInventory++;
		return true;
	}
	return false;
}

void AEvilPathMainCharacter::UpdateInventory()
{
	//Call Update
	OnUpdateInventory.Broadcast(_inventory);
}

// Sets default values
AEvilPathMainCharacter::AEvilPathMainCharacter()
{
	// Set this character to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
```

```
	PrimaryActorTick.bCanEverTick = true;
	sizeOfInventory = 0;

}

int AEvilPathMainCharacter::getSizeOfInventory()
{
	return sizeOfInventory;
}

// Called when the game starts or when spawned
void AEvilPathMainCharacter::BeginPlay()
{
	Super::BeginPlay();

}

// Called every frame
void AEvilPathMainCharacter::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}

// Called to bind functionality to input
void AEvilPathMainCharacter::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);

}
```

Rotating Item (.h file)

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "PickupItem.generated.h"

UCLASS()
class MYPROJECT_API APickupItem : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	APickupItem();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	UPROPERTY(EditAnyWhere, BlueprintReadWrite, Category = Pickup)
```

```
	FRotator RotationRate;
	//SceneCoomponent as Root

	UPROPERTY(VisibleAnyWhere, BlueprintReadWrite, Category = Pickup)
	USceneComponent* SceneComponent;

	// MeshComponent

	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Pickup)
	UStaticMeshComponent* ItemMesh;
};
```

Rotating Item (Cpp file)

```
#include "PickupItem.h"
#include "Components/StaticMeshComponent.h"

// Sets default values

APickupItem::APickupItem()
{
	// Set this actor to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	this->SceneComponent =
CreateDefaultSubobject<USceneComponent>(TEXT("SceneComponent"));

	this->RootComponent = SceneComponent;

	this->ItemMesh =
CreateDefaultSubobject<UStaticMeshComponent>(TEXT("ItemMesh"));

	//this->ItemMesh->AttachToComponent(this->RootComponent,
FAttachmentTransformRules::KeepWorldTransform);
	this->RootComponent->SetupAttachment(this->ItemMesh);
	this->RotationRate = FRotator(0.0f, 180.0f, 0.0f);

}

// Called when the game starts or when spawned
void APickupItem::BeginPlay()
{
	Super::BeginPlay();

}

// Called every frame
void APickupItem::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	this->AddActorLocalRotation(this->RotationRate * DeltaTime);
}
```