Technical Design Document

Connor Martin XB3992 – Honours Project

MOCKTROID

Date Created 9/11/20

Last Updated 13/04/21

Version 1.0

Contents

1.0 Executive Summary

1.1 Project Overview

Mocktroid is a gameplay demo in the style of a Metroid Prime/First Person Adventure game that is used to demonstrate a collection of mechanics and systems that could be applied to any number of games, across a multitude of genres.

1.2 Technical Overview

Mocktroid is an implementation of mechanics, systems and features created using Epic Games' Unreal Engine 4 software development kit. Unreal Engine's Blueprints visual scripting language will be used to develop a both gameplay mechanics and back-end systems that will then be used in a gameplay demo based around the Metroid Prime series of games. However, the many of the features are intended to be laterally designed so that they can be implemented into a wide variety of games across multiple genres. This gameplay demo is merely one example of how they could be utilised in a game project.

Other programs such as Autodesk Maya and Adobe Photoshop will be used in a support role to the project to help elevate the gameplay demo above white/greybox status to further support the idea that the demo is an example of the integration of features in a game intended for commercial release.

1.3 Document Summary

This Technical Design Document (TDD) is intended to outline the software used to create the project, as well as give a detailed explanation of the development and implementation of the technical features present in the project.

2.0 Software

The following is a complete list of all the software packages I intend to use in this project.

2.1 2D Software

Adobe Photoshop
Lucidchart Flowchart Maker

2.2 3D Software

Unreal Engine 4
Autodesk Maya
Adobe Substance Painter

2.3 Sound Software

Audacity

2.4 Additional Software

Microsoft Office Word
Microsoft Office PowerPoint
Microsoft Office Publisher
GitHub

3.0 Hardware Requirements

3.1 Minimum

Processor: Intel Core i5/AMD Ryzen 5
RAM: 8GB
GPU: Integrated

3.2 Recommended

Processor: Intel Core i5/AMD Ryzen 5
RAM: 16GB
GPU: Nvidia GTX960 4GB or equivalent

4.0 Asset List

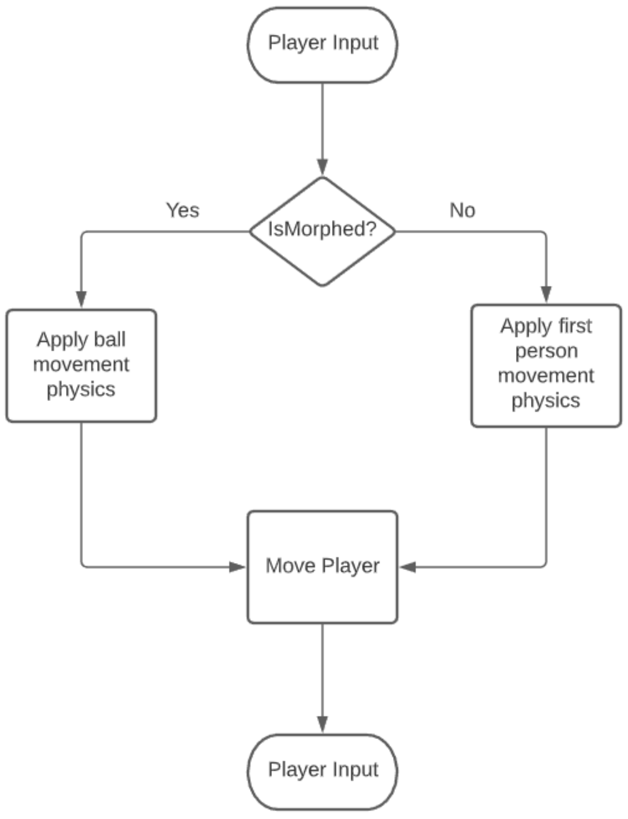
- Player Character
    - Shooting
    - Movement
- Lock on
- Health and Ammo
- Player Alt Mode
- Scanning System
- Logbook
- Enemy Type 1
- Enemy Type 2
- Mini Boss Enemy
- Level Streaming System
- Save and Load
- 3D Map System
- Powerups and Inventory
- Drops
- HUD

5.0 Player Movement

5.1 Overview

A physics-based controller that can function as both a modern first-person player and a third person rolling ball controller.

5.2 Planned Solution


Player Input
IsMorphed?
Yes
No
Apply ball movement physics
Apply first person movement physics
Move Player
Player Input

5.3 Implementation

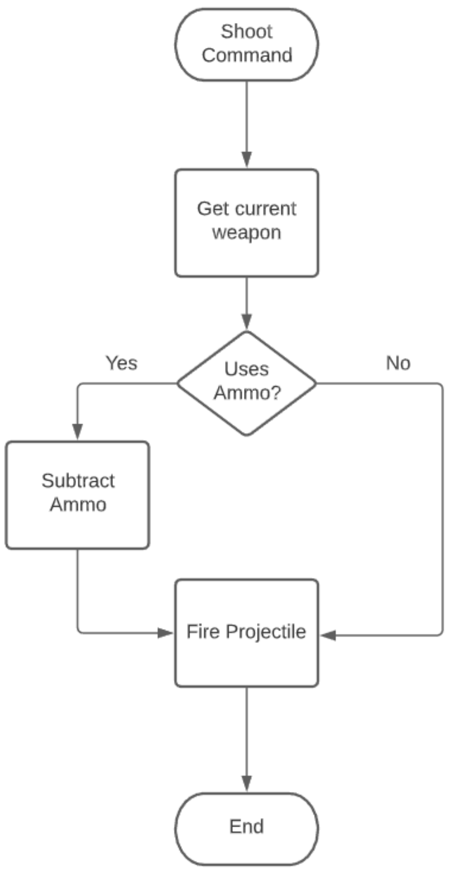
The player movement functions as planned with the game checking the morph state of the player and adjusting values to the physics movement such as max speed and friction based on whether the player is in first person or ball mode.

6.0 Player Shooting

6.1 Overview

Check the players selected weapon. If the weapon uses ammo, subtract the required amount from the players ammo supply. Fire the weapon.

6.2 Planned Solution



6.3 Implementation

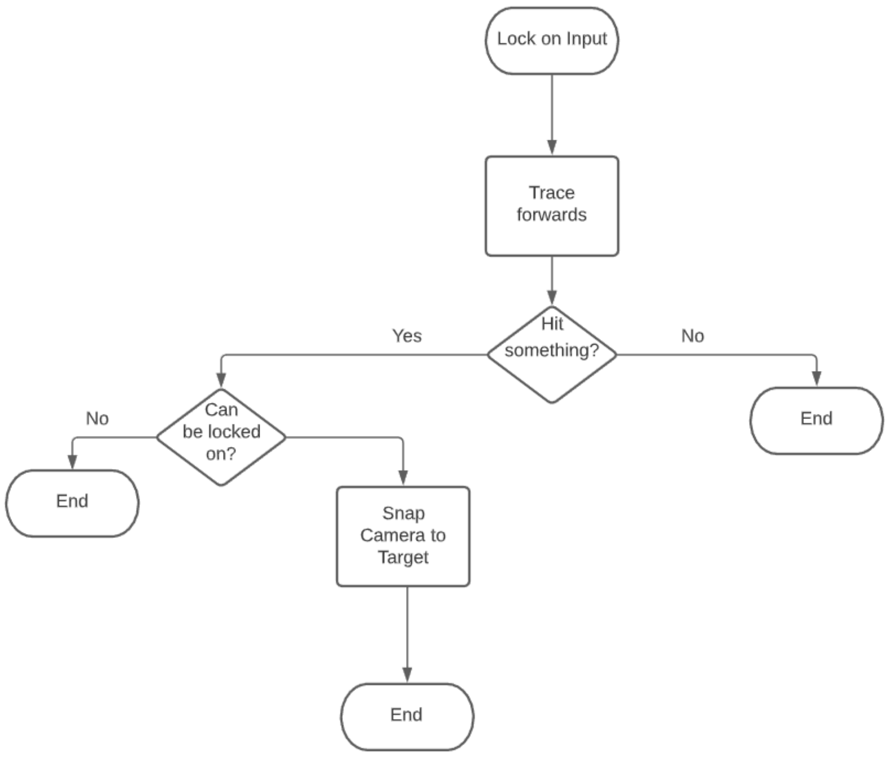
The shooting was implemented as planned with the function used to spawn the projectile being stored in a blueprint component that can be added to any actor I want to have shooting functionality.

7.0 Lock On

7.1 Overview

Player presses the lock on button and if a valid target is in range, then snap the players camera to the targeted object.

7.2 Planned Solution



7.3 Implementation

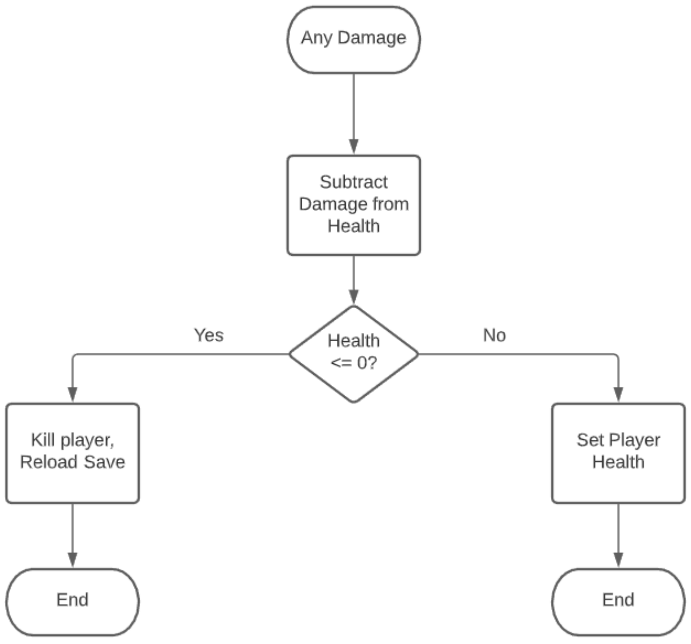
This functionality was implemented as planned but was modified to work on an input axis so that the camera would update its location with moving targets.

8.0 Health

8.1 Overview

Player is hit by a damaging object. Damage taken is subtracted from the players health and if the return value is below 0 then kill the player and reload their last save.

8.2 Planned Solution



8.3 Implementation

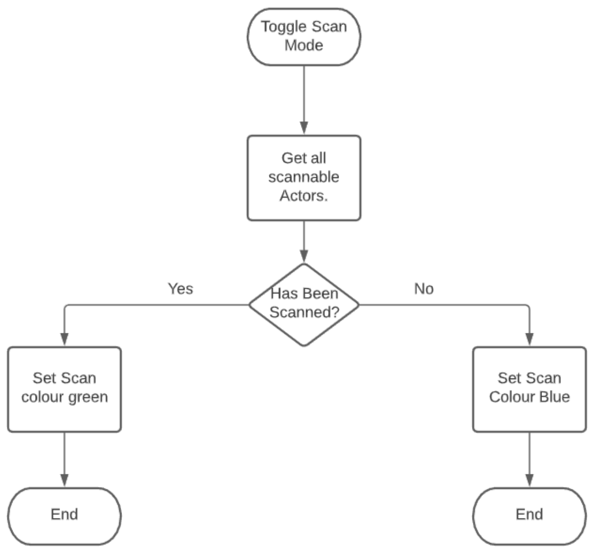
This functionality was implemented as planned with the player loading back to their most recent save on death.

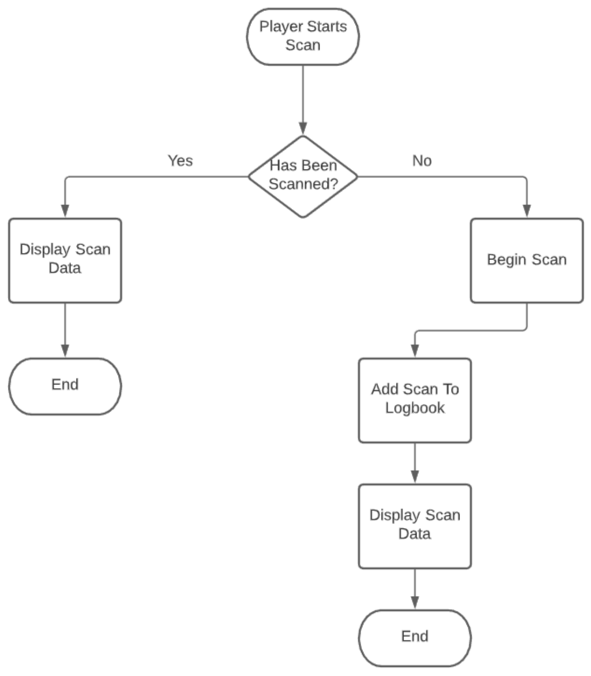
9.0 Scanning System

9.1 Overview

When the player toggles the scan mode scannable objects will be highlighted either blue for objects that the player has not scanned, or green for objects the player has scanned. When the player locks on to the object then the scan will start, then display the scan data for the object.

9.2 Planned Solution





9.3 Implementation

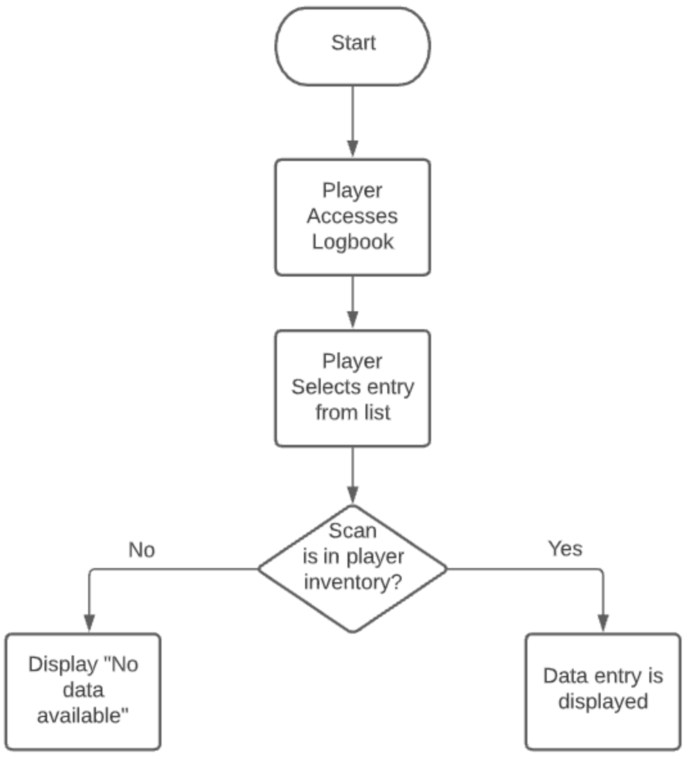
This functionality was implemented as planned. The original prototype was based on two materials however the version that will be used going forward uses a single material that lerps between two different colours.

10.0 Logbook

10.1 Overview

Give the player the ability to access an assets scan information from a logbook menu at any time after they have scanned the asset in question.

10.2 Planned Solution


Start
Player Accesses Logbook
Player Selects entry from list
Scan is in player inventory?
No
Yes
Display "No data available"
Data entry is displayed

10.3 Implementation

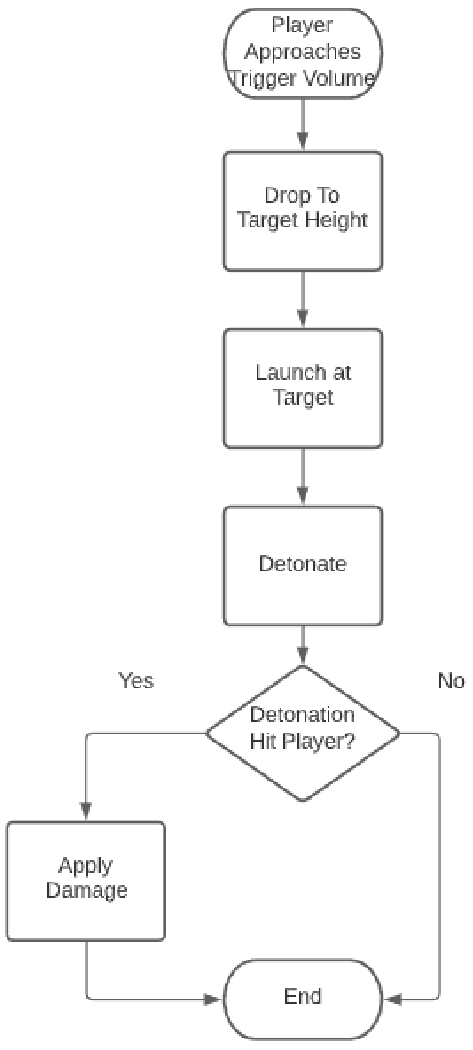
This functionality was integrated smoothly with only minor modifications to the player to give the UI easier access to the full scan database.

11.0 Enemy Type 1

11.1 Overview

Enemy will be suspended from the ceiling and when the player approaches the enemy will drop and launch at a target location causing damage on impact.

11.2 Planned Solution



11.3 Implementation

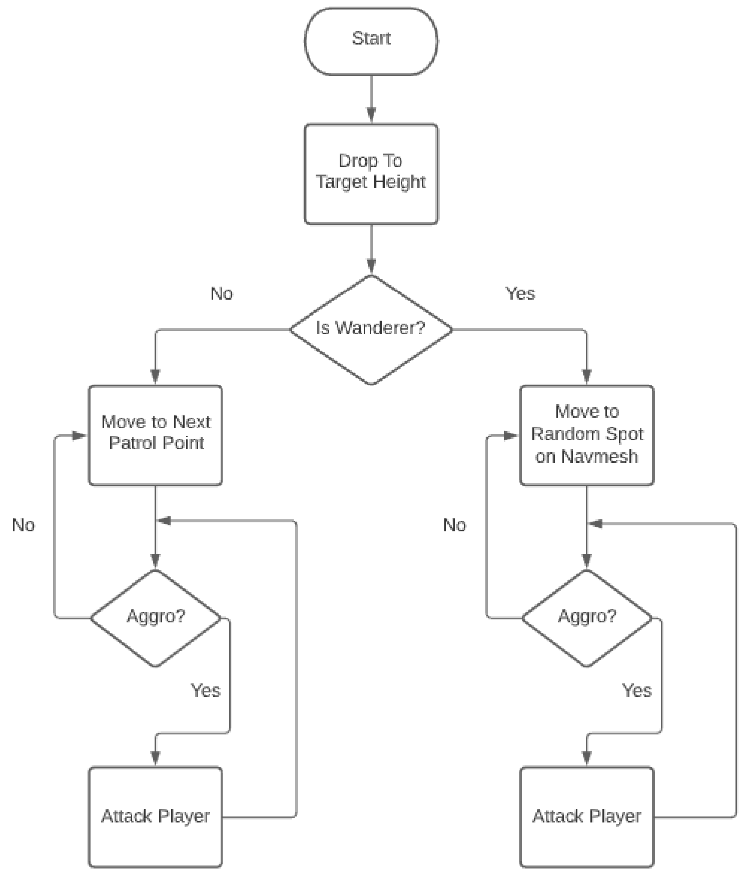
Functionality was implemented as planned

12.0 Enemy Type 2

12.1 Overview

Enemy will wander to a random point on the navmesh, or between set points, when the player comes near the enemy will attack the player.

12.2 Planned Solution



12.3 Implementation

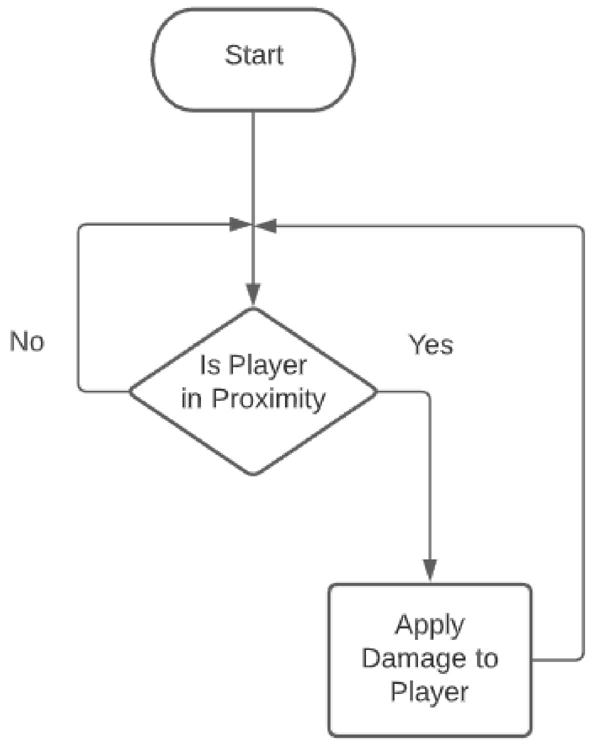
This enemy type was implemented as planned with the attack being refined to include a random timer between attacks.

13.0 Enemy Type 3

13.1 Overview

Enemy is static and will passively damage the player if they come within a certain proximity of the player

13.2 Planned Solution



13.3 Implementation

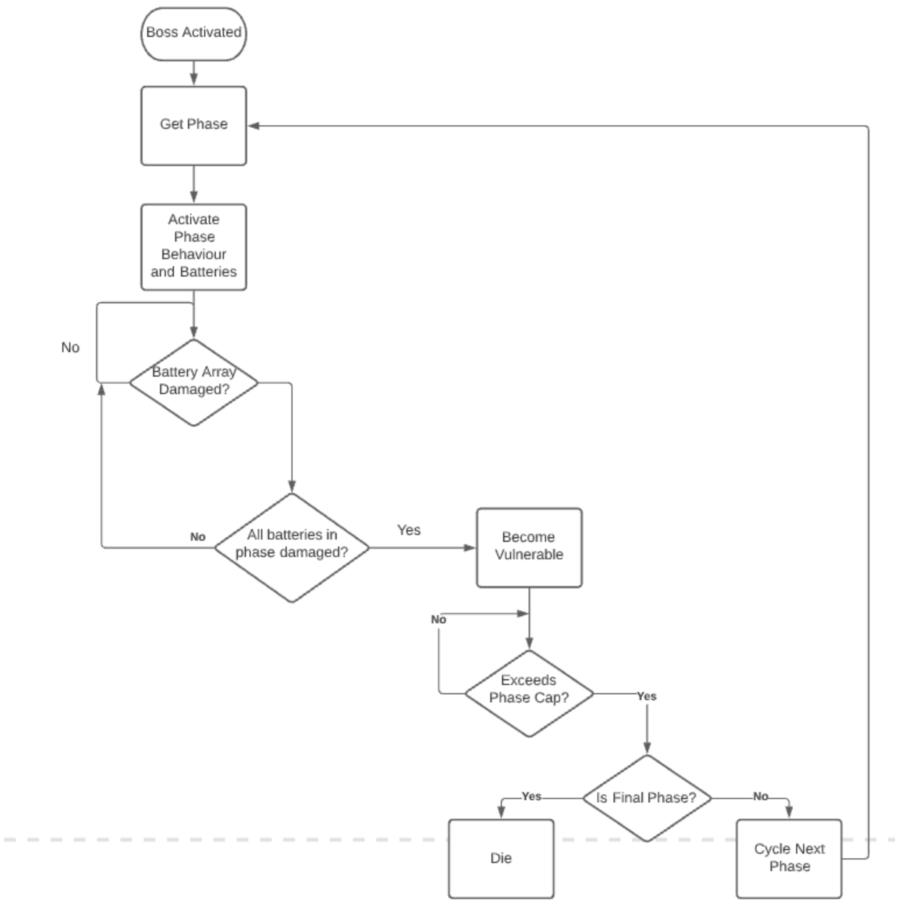
Enemy was implemented as planned

14.0 Mini Boss Enemy

14.1 Overview

A more complex enemy type with increased health and changing phases. Fight will only occur once and will serve as a key to progression.

14.2 Planned Solution



14.3 Implementation

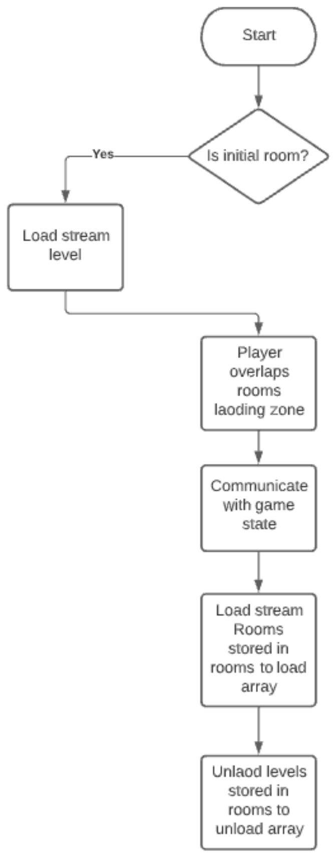
The boss enemy was implemented successfully and included varieties in behaviour such as random alternating attack patterns in the latter stages of the fight to increase difficulty.

15.0 Level Streaming System

15.1 Overview

Create a robust system for loading areas of the game level in and out to maximise performance whilst maintaining the illusion of a continuous game world.

15.2 Planned Solution


Start
Is initial room?
Yes
Load stream level
Player overlaps rooms laoding zone
Communicate with game state
Load stream Rooms stored in rooms to load array
Unlaod levels stored in rooms to unload array

15.3 Implementation

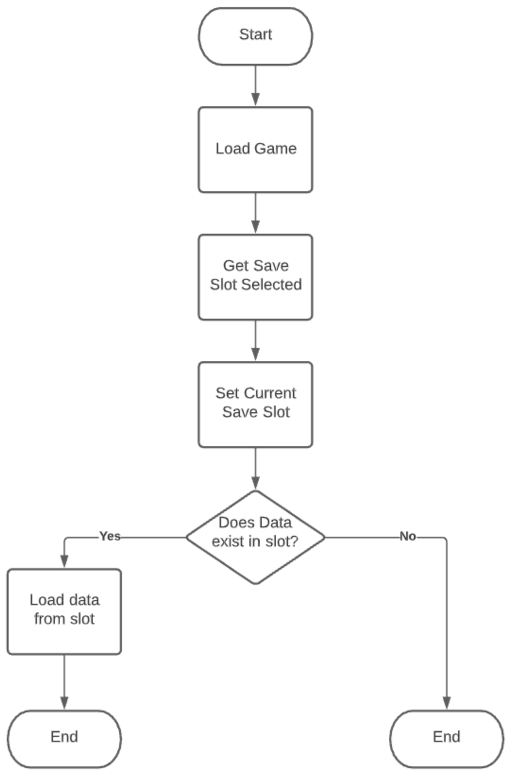
This functionality was implemented successfully and serves as a vast improvement on previous level streaming solutions used in the past.

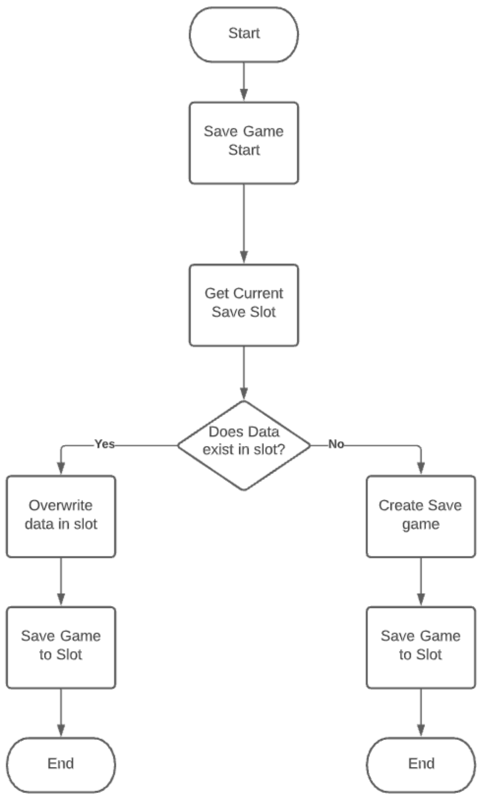
16.0 Save and Load

16.1 Overview

Give the player the ability to save their progress at designated points across the world to a specific and then load that progress next time they play the game, whist also allowing the player to start a new game in a separate save slot.

16.2 Planned Solution





16.3 Implementation

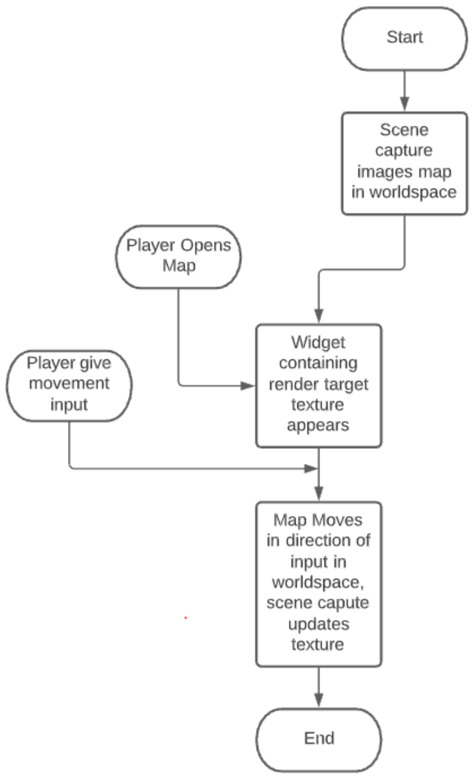
This system was implemented successfully and quickly with no complications.

17.0 3D Map system

17.1 Overview

A 3D map of the game level that highlights the room the player is currently in and can be moved around and rotated.

17.2 Planned Solution


Start
Scene capture images map in worldspace
Player Opens Map
Player give movement input
Widget containing render target texture appears
Map Moves in direction of input in worldspace, scene capute updates texture
End

17.3 Implementation

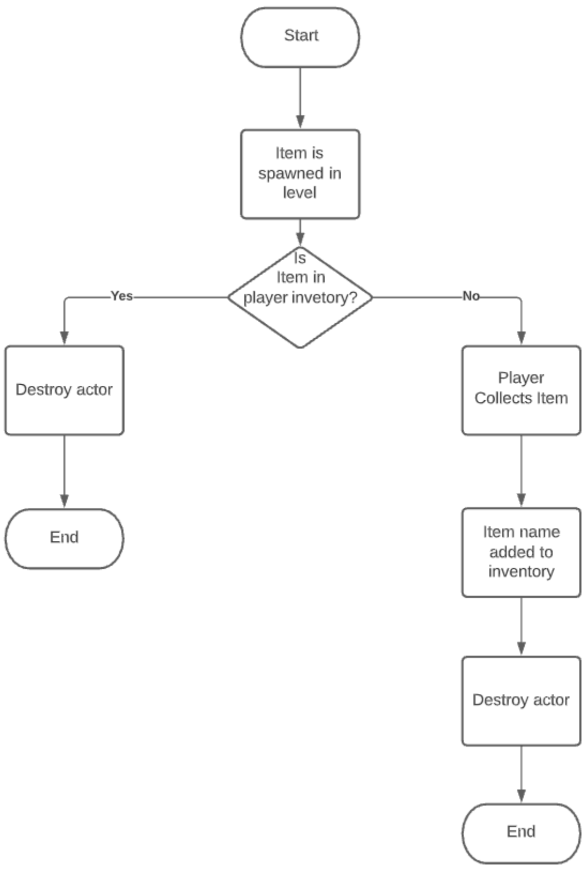
The map was implemented as planned however when viewing the map, the game sets global time dilation to 0 instead of pausing the game as locations cannot be updated when the game is paused.

18.0 Items and Inventory

18.1 Overview

Permanent items that the player can collect that grant powers or expansions to existing powers.

18.2 Planned Solution



18.3 Implementation

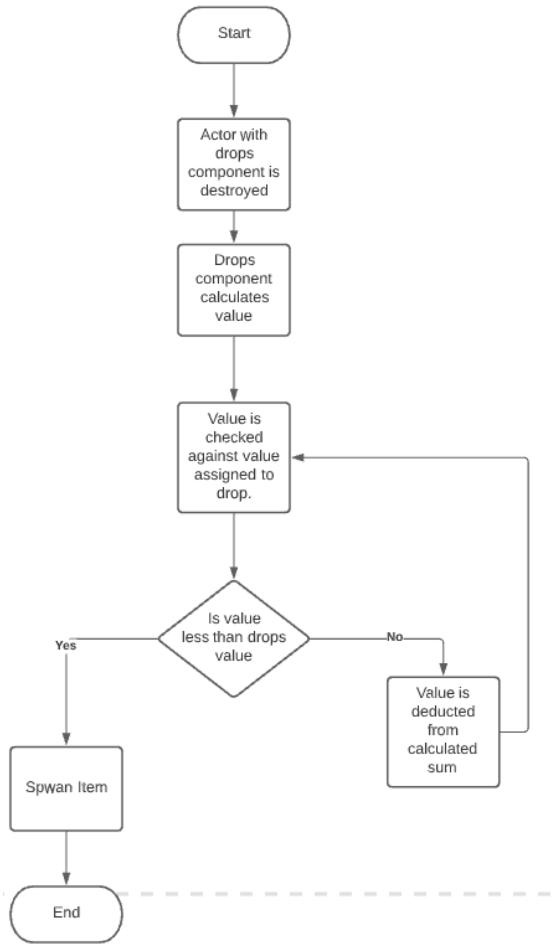
This system was implemented as planned.

19.0 Drops

19.1 Overview

Health and ammo drops that are spawned when specified actors are destroyed, with a weighing system that makes smaller health and ammo units more common than larger denominations.

19.2 Planned Solution


Start
Actor with drops component is destroyed
Drops component calculates value
Value is checked against value assigned to drop.
Is value less than drops value
Yes
No
Value is deducted from calculated sum
Spwan Item
End

19.3 Implementation

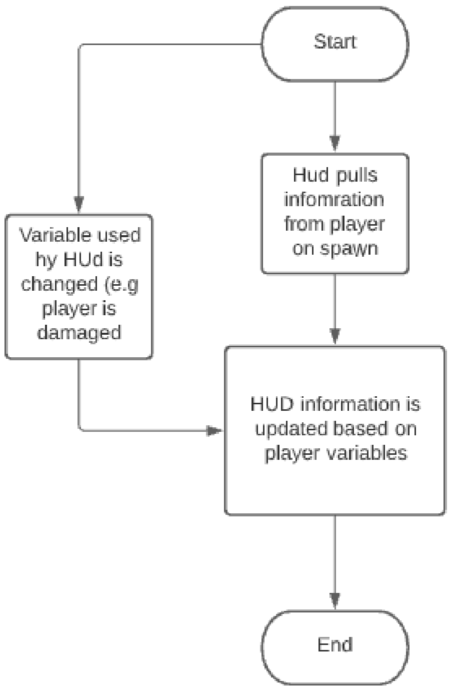
This functionality was implemented as planned with drops being chosen based on a ticket system and the calculation of a random value that is used to select drops.

20.0 HUD

20.1 Overview

A diegetic heads-up display that displays the player's current and max health, current and max ammo, current weapon, mini map display, and the currently equipped visor.

20.2 Planned Solution


Start
Hud pulls infomration from player on spawn
Variable used hy HUd is changed (e.g player is damaged
HUD information is updated based on player variables
End

20.3 Implementation

This functionality was implemented as planned. However, I broke the HUD down into two sections, a visor display for health, map, and current visor. And a holographic display on the player weapon that displays the players current weapon and ammo.