Version 1.12024/25

University of Staffordshire

[Zen Zone]

Technical Design Document

Adam Puttergill

P020640L@student.staffs.ac.uk

Contents

Project Introduction

Zen Zone will be a relaxing life sim game that includes caring for and nurturing a unique set of plants as they mature. The player can:

- Take different forms as needed in order to water and fertilize their plants, as well as gather resources
- Interact with the different elements of the world surrounding them
- Rebuild ruined landmarks and restore the lands

The game is inspired by Stardew Valley, as well as the Plants vs Zombies minigame, Zen Garden.

Project Goals

The overall goal of this project is to create a prototype for a 2D- Top-Down game using Unreal Engine's Paper2D and PaperZD plugins. It's a side of Unreal that I haven't yet explored within a genre I thoroughly enjoy. Further goals are to create a relaxing atmosphere through the use of VFX, SFX and Music.

Challenges and Risks

Potential challenges working within this style include a lack of support from Epic Games – there is only a very little amount of documentation related to the Paper2D & PaperZD plugins. Even unofficially, there are only small amounts of people sharing information on how to best use these plugins.

Additionally, animation is something I've never attempted to do properly and in such a full fashion as I plan to with this project.

Hardware Requirements

- Requires a 64-bit processor and operating system
- OS *: Windows® 7 SP1 / 8.1 / 10 64-bit
- Processor: AMD FX-4350 / Intel® Core™ i3-3210
- Memory: 4 GB RAM
- Graphics: AMD Radeon™ R7 260X (2GB VRAM) / NVIDIA® GeForce® GTX 750(2GB VRAM)
- DirectX: Version 11
- Storage: 1 GB available space
- Sound Card: DirectX Compatible Sound Card

Platforms

Target Platform

This prototype will be created for PC as the platform has a massive marketplace for 2D Top-Down games, both on Steam as well as Itch.io. Movement controls for this genre are also better suited for mouse & keyboard, although it is feasible to develop it for console platforms.

Engine Specific Specifications and Limitations

As the game is 2D, and in a pixel-art style, there shouldn't be any egregious hardware or disk-space limitations. However, there are limited resources available regarding the creation of 2D games in Unreal Engine, as mentioned previously.

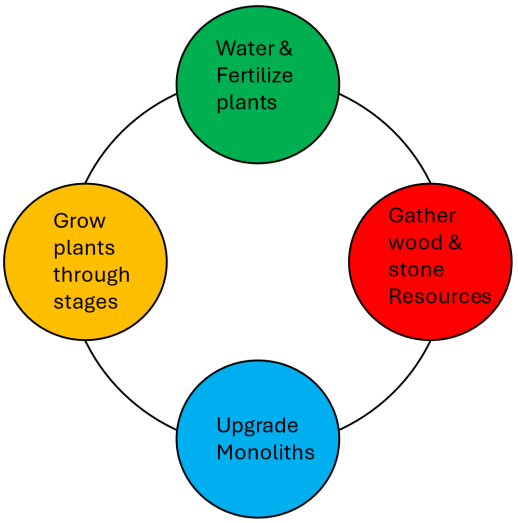
Engine Summary

This project is being developed within Unreal Engine 5.4.4, using the Paper2D & PaperZD plugins.

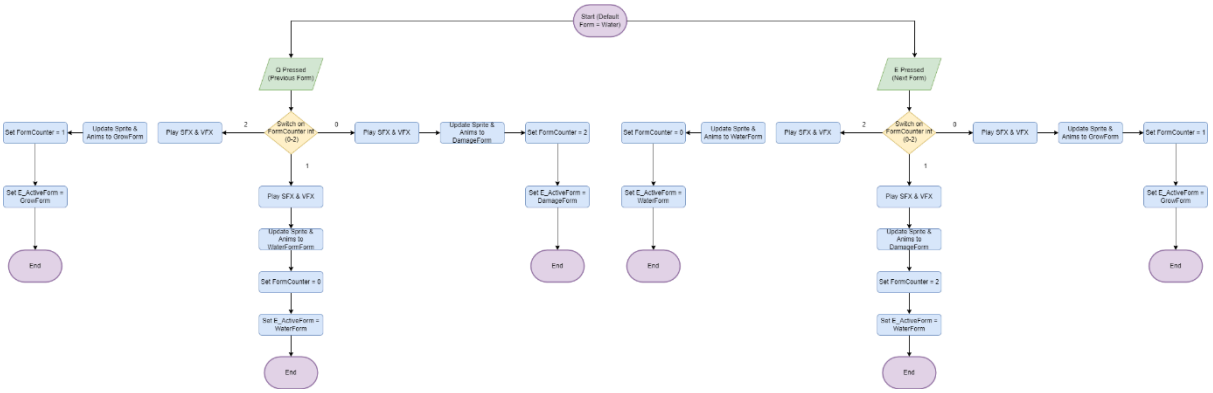
Systems and Diagrams

Gameplay Loop

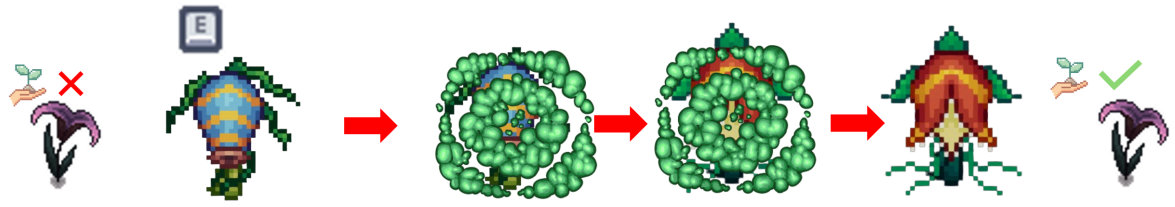
Gameplay Loop



Changeable Character Forms



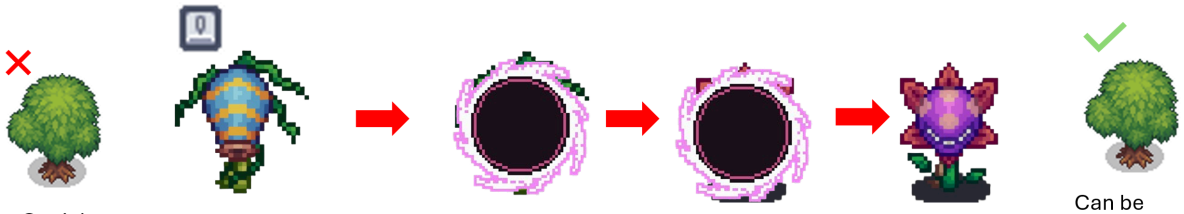
WaterForm to GrowForm



Can't be Fertilized

Can be Fertilized

WaterForm to Damage Form



Can't be Destroyed

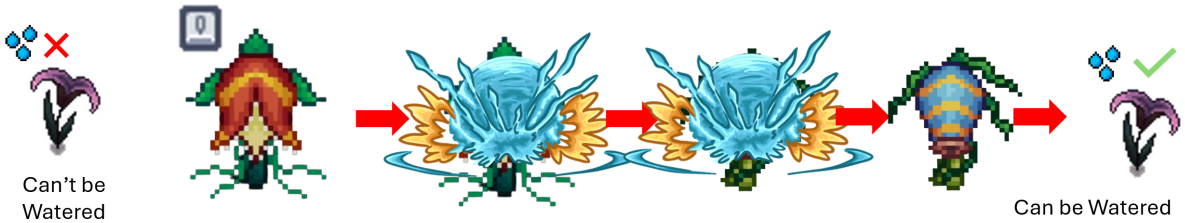
Can be Destroyed

GrowForm to DamageForm

Can't be Destroyed

Can be Destroyed

GrowForm to WaterForm



Can't be Watered

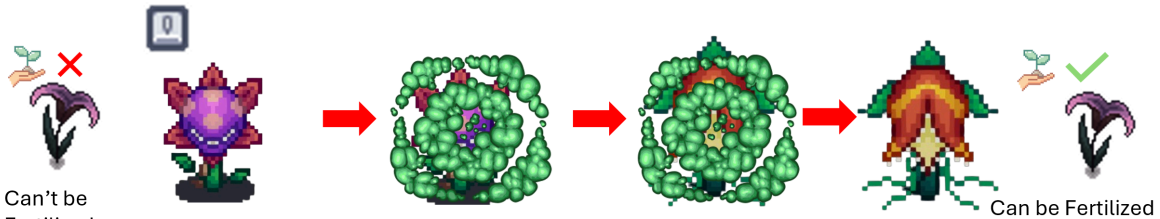
Can be Watered

DamageForm to WaterForm

Can't be Watered

Can be Watered

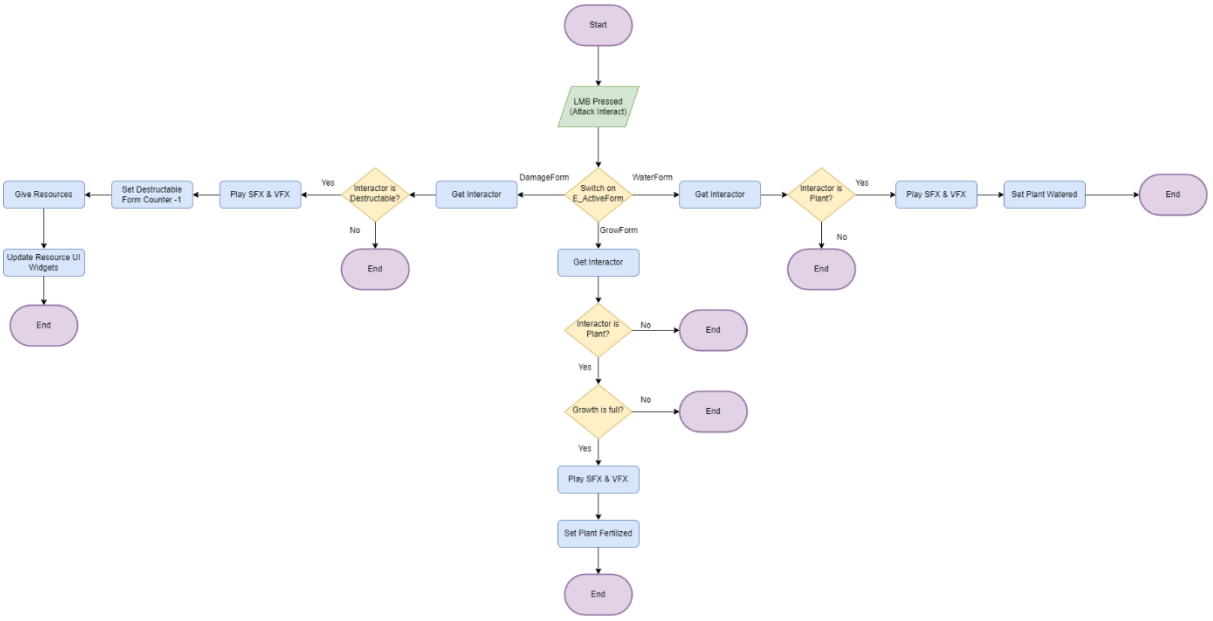
DamageForm to GrowForm



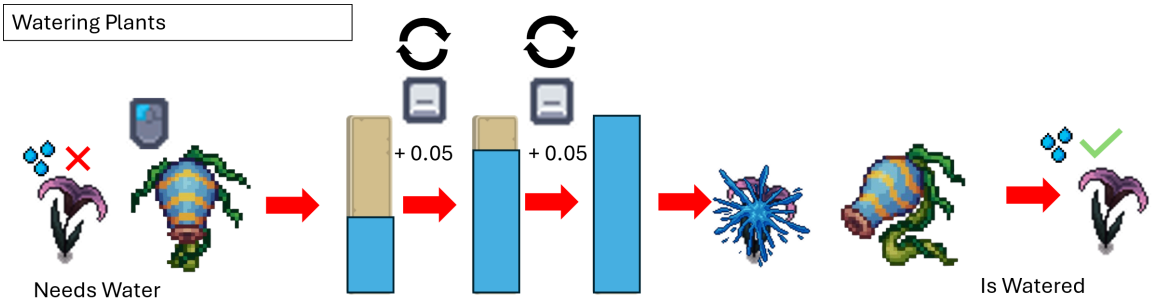
Can't be Fertilized

Can be Fertilized

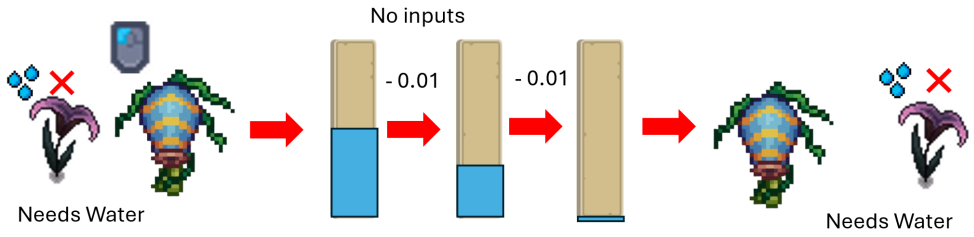
Form-Interactions



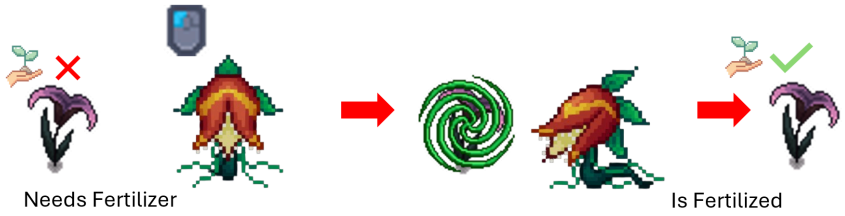
Watering Plants



Watering Plants



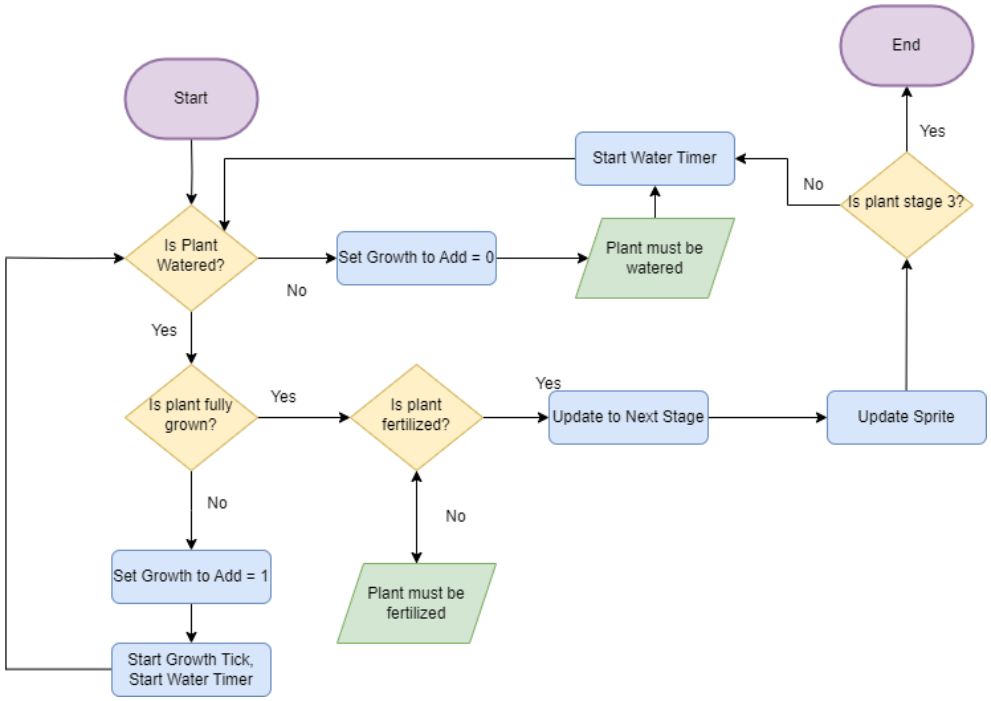
Fertilizing Plants



Destroying Trees

Destroying Rocks

Plants


Start
Is Plant Watered?
No
Set Growth to Add = 0
Plant must be watered
Start Water Timer
Yes
Is plant fully grown?
Yes
Is plant fertilized?
No
Plant must be fertilized
Yes
Update to Next Stage
Update Sprite
Is plant stage 3?
No
Yes
End
No
Set Growth to Add = 1
Start Growth Tick, Start Water Timer

Growing stage 1 to stage 2

Is Fertilized

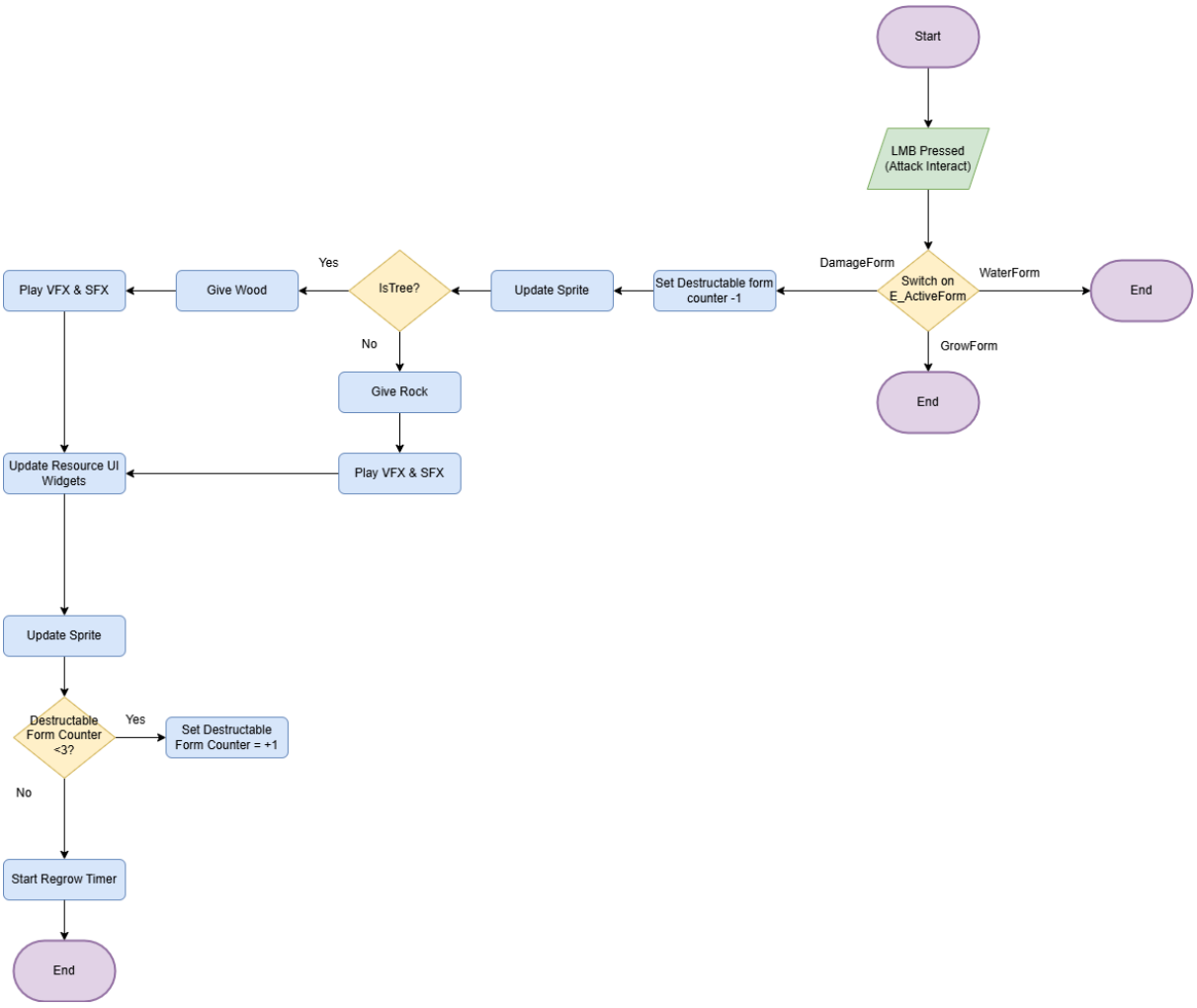
Needs Water

Growing stage 2 to stage 3

Is Fertilized

Needs Water

Trees/Rocks



Tree stages

*Stage updated each time it is attacked

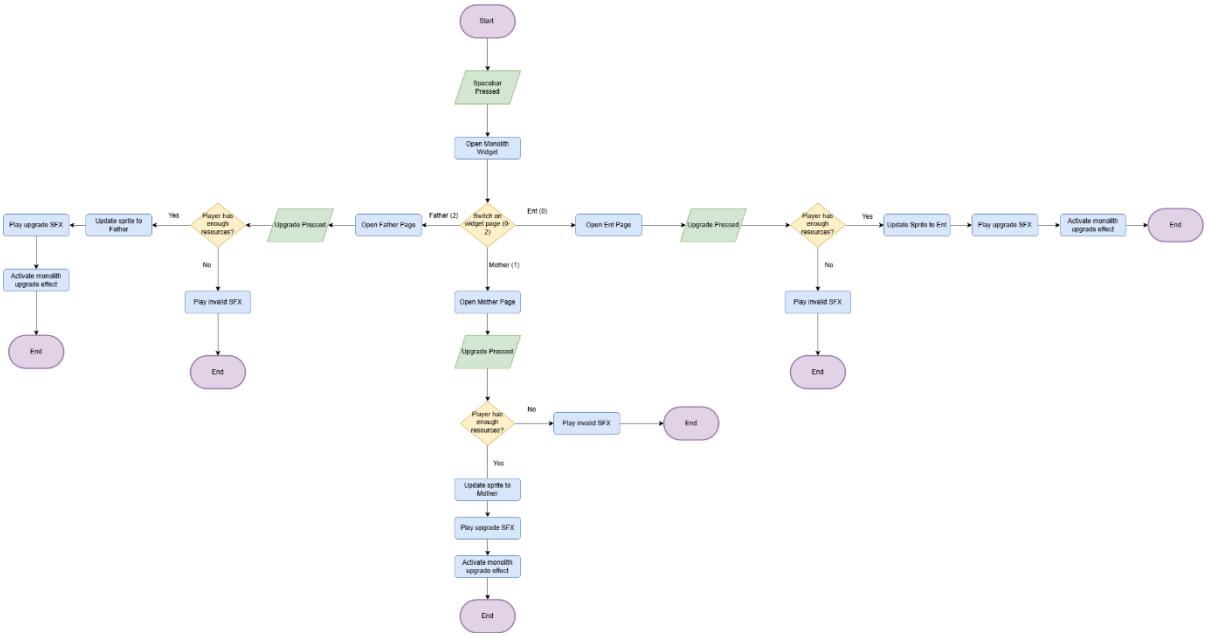
Stage 1 Stage 2 Stage 3 Stage 4 (Regrows after 30 seconds)

Destroying Rocks

*Stage updated each time it is attacked

Stage 1 Stage 2 Stage 3 Stage 4 (Regrows after 30 seconds)

Monoliths



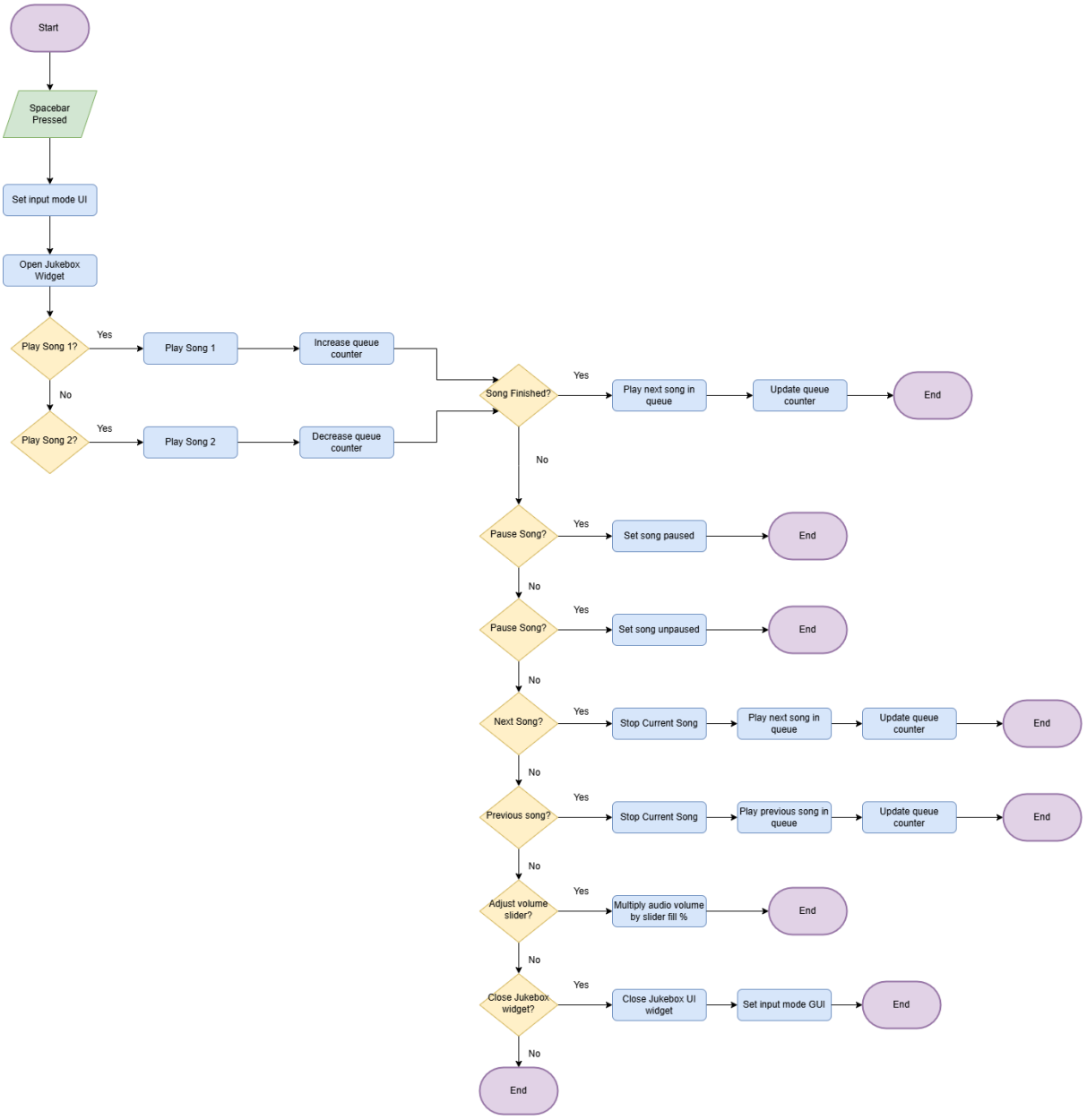
Mother Monolith Upgrade

Ent Monolith Upgrade

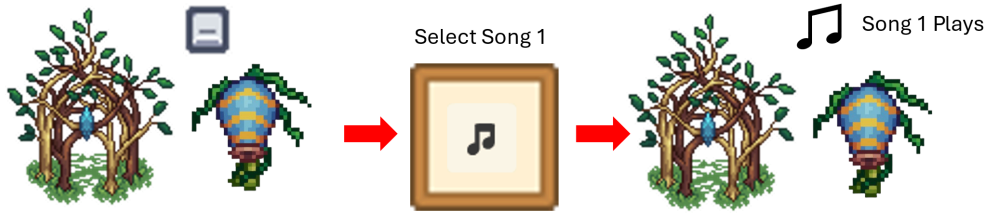
Ent Monolith Upgrade

Father Monolith Upgrade

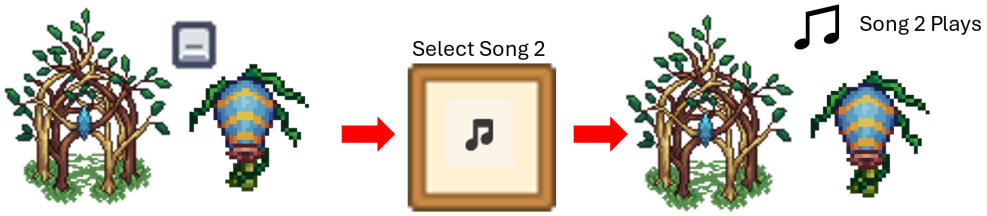
Jukebox


Start
Spacebar Pressed
Set input mode UI
Open Jukebox Widget
Play Song 1?
Yes
Play Song 1
Increase queue counter
No
Play Song 2?
Yes
Play Song 2
Decrease queue counter
Song Finished?
Yes
Play next song in queue
Update queue counter
End
No
Pause Song?
Yes
Set song paused
End
No
Pause Song?
Yes
Set song unpaused
End
No
Next Song?
Yes
Stop Current Song
Play next song in queue
Update queue counter
End
No
Previous song?
Yes
Stop Current Song
Play previous song in queue
Update queue counter
End
No
Adjust volume slider?
Yes
Multiply audio volume by slider fill %
End
No
Close Jukebox widget?
Yes
Close Jukebox UI widget
Set input mode GUI
End
No
End

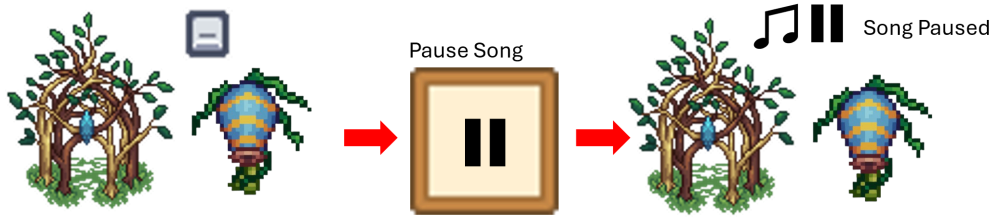
Jukebox – Select Song 1



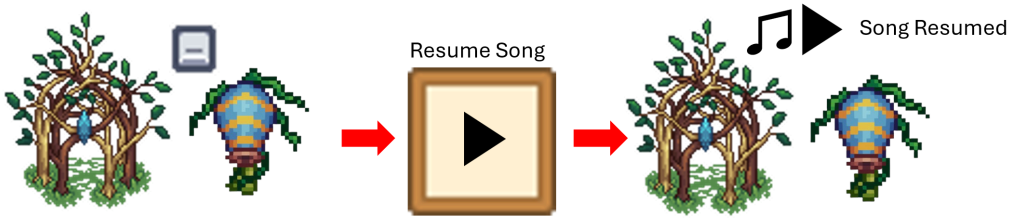
Jukebox – Select Song 2



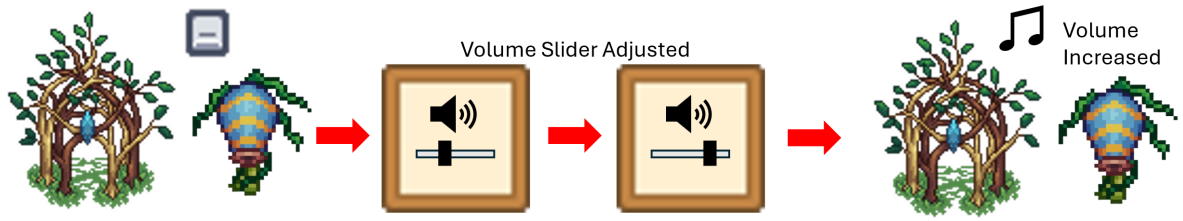
Jukebox – Pause Song



Jukebox – Resume Song



Jukebox – Adjust Volume

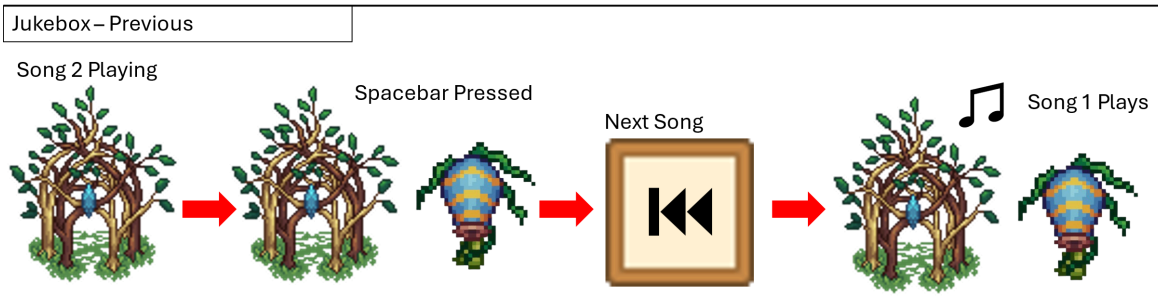


Jukebox – Next Song

Jukebox – Previous

Jukebox – Next Song

Jukebox – Previous



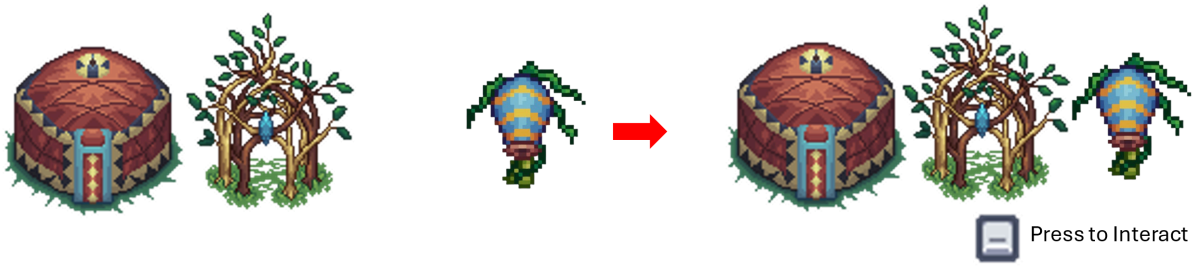
Zoom Adjustment

Zoom In

Zoom Out

Spatial Keybind Popups for Interactables

Jukebox & Monolith Hut

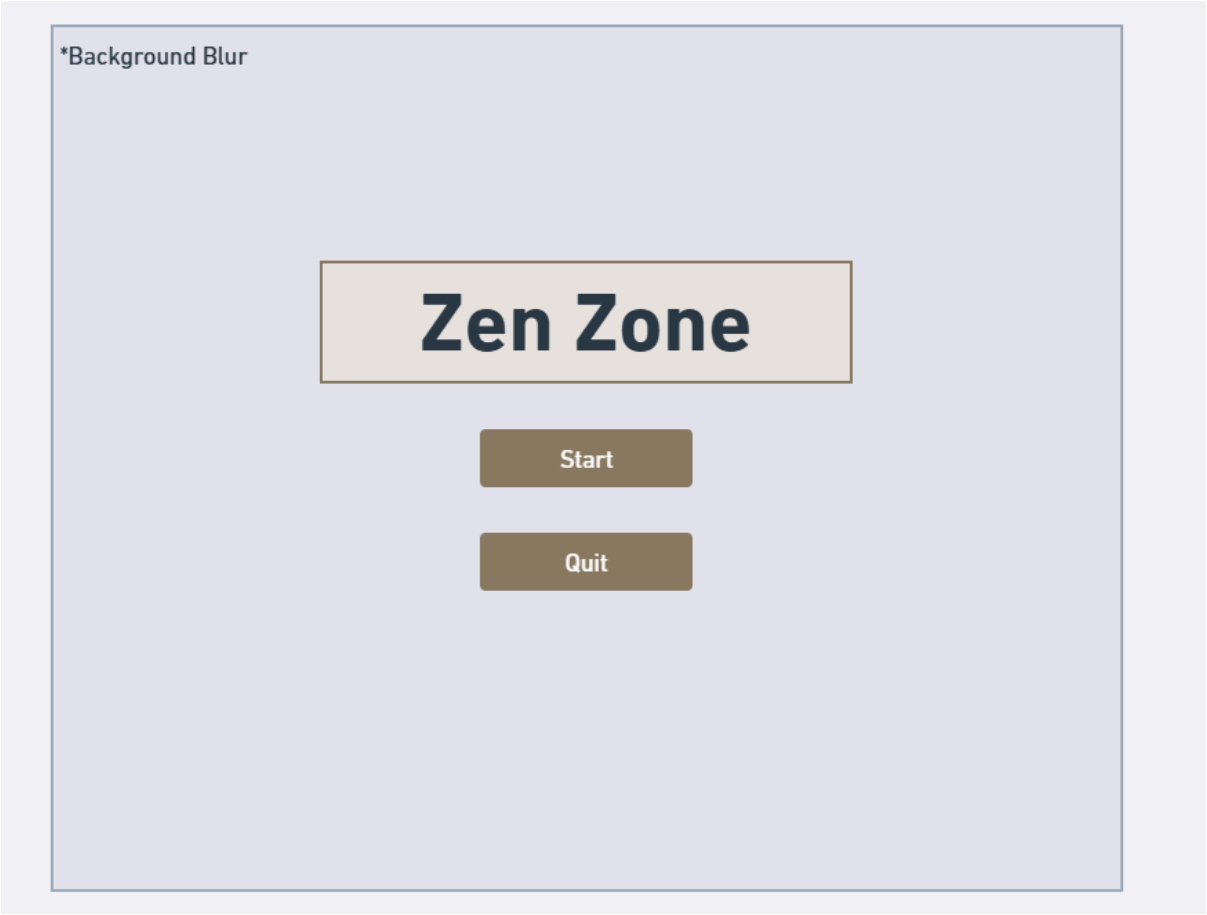


Plants & Destructables

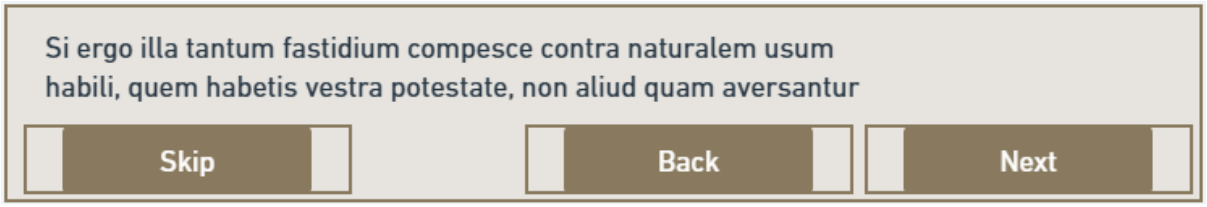
Wireframes

Main HUD

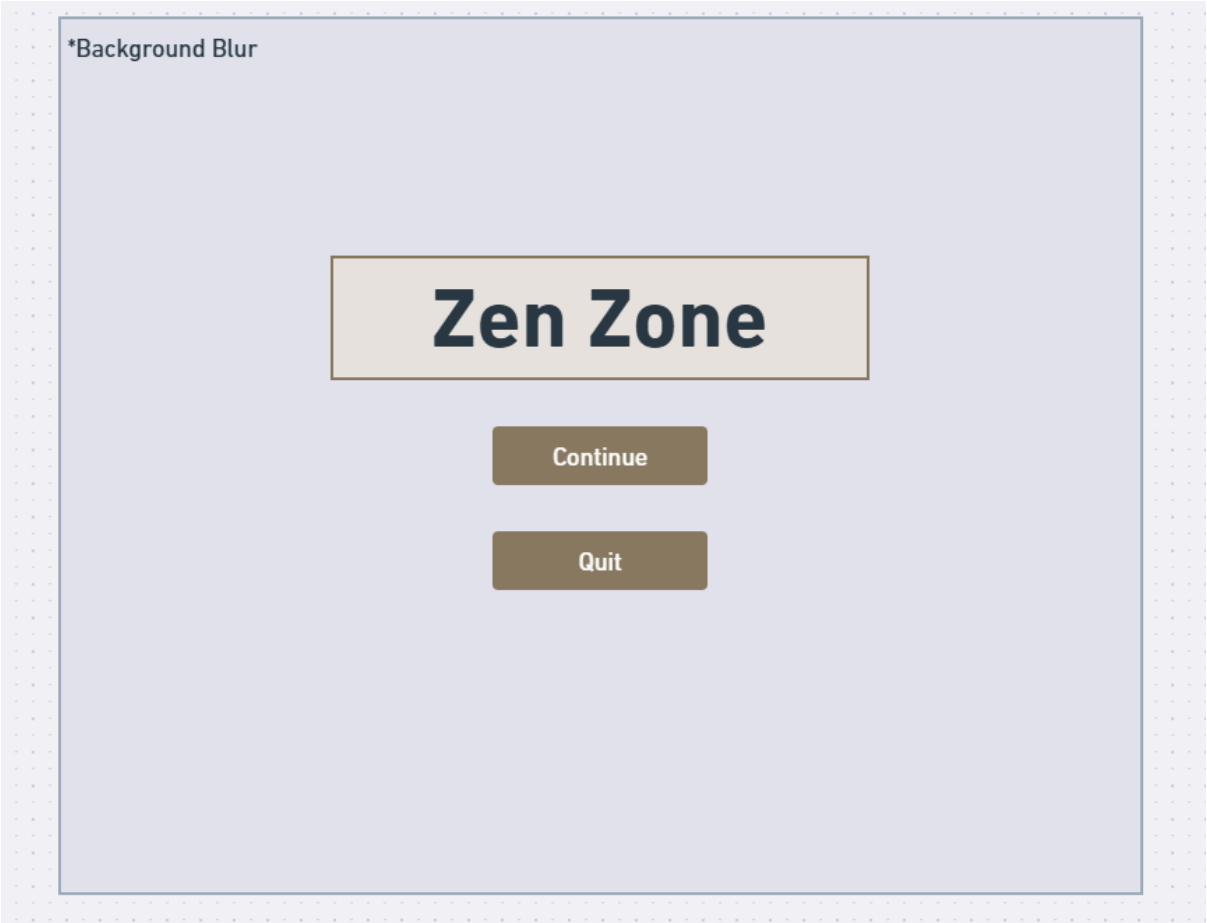
Main Menu



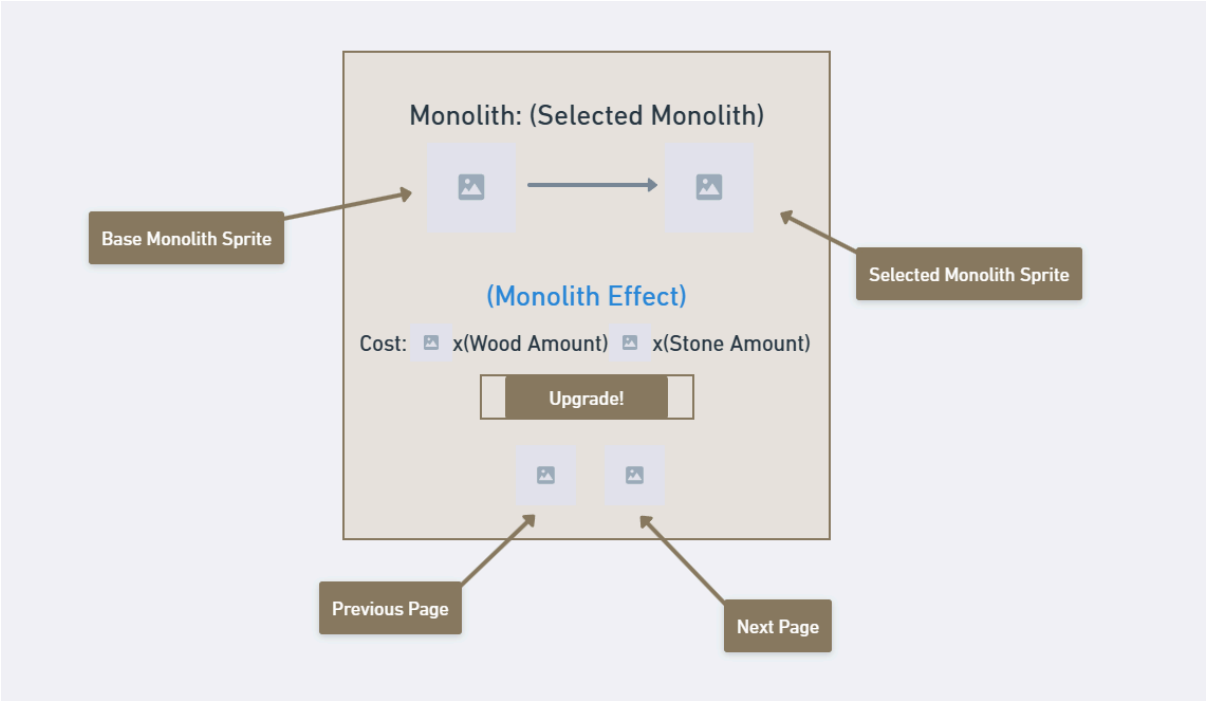
Tutorial Popup



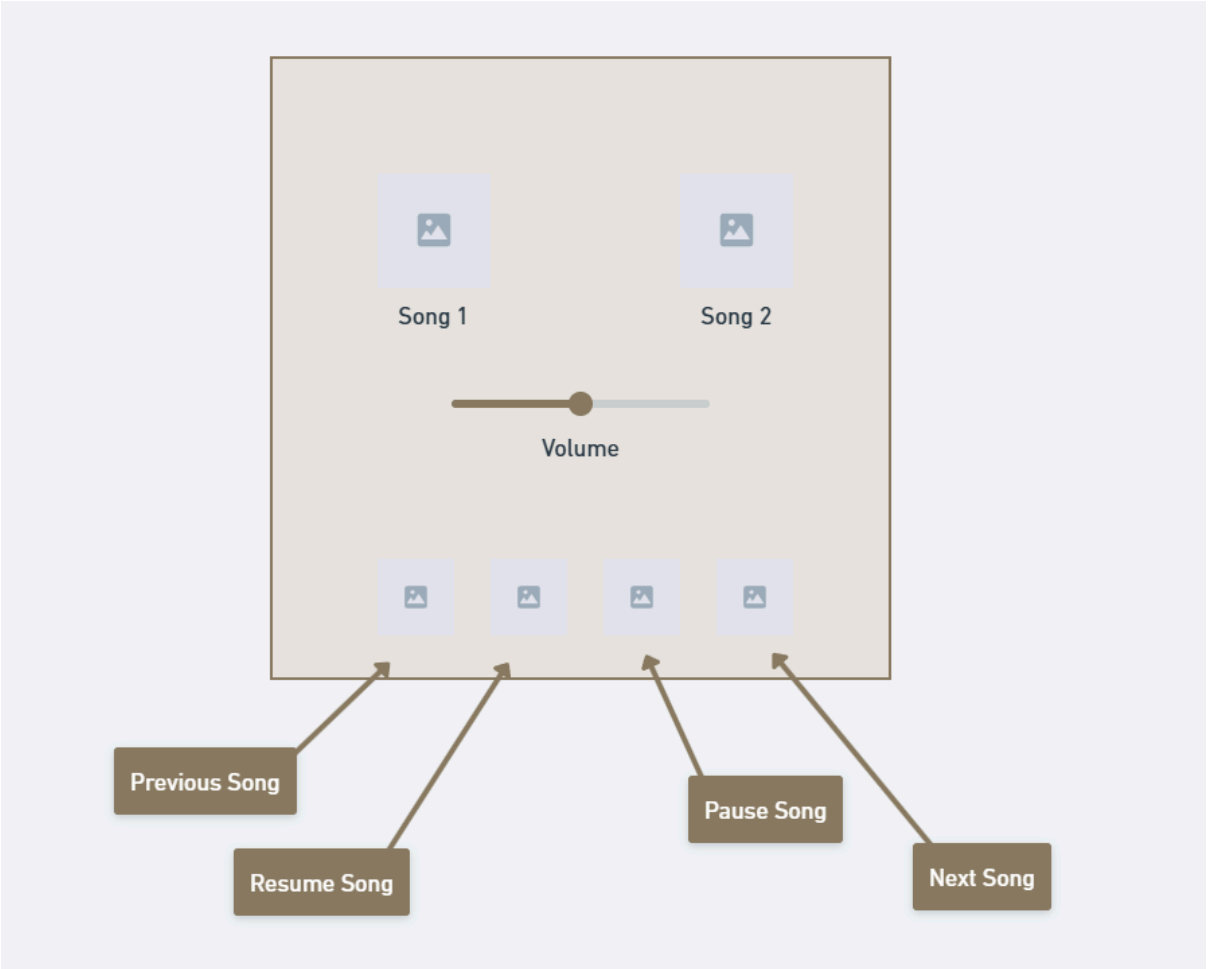
Pause Menu



Monolith Menu



Jukebox Menu



Optimisation and Profiling

Profiling Systems

Weekly systems reviews will be conducted to determine that the functionality of systems is working as intended, as well as ensuring that they have been programmed efficiently.

Additionally, regular playtesting sessions will be conducted throughout the project development timeline in order to stay on top of any bugs that may occur as well as having a second set of eyes to ensure that system functionality is as intended.

Profiling Graphics

In order to test graphical performance, I will test the in-engine performance of the project each time it is updated (1-3 times weekly). This will be done using Unreal Engine's 'Show Stats & FPS' tool.

Additionally, further details on graphic performance will be reviewed using the advanced details tab of this tool.

Coding Standards

Programming Standards

Conventional Unreal Engine 5 naming standards will be adhered to:

- Regular Blueprints: BP_(Name)
- Widget Blueprints: WBP_(Name)
- Animation Blueprints: ABP_(Name)
- Interface Blueprints: BPI_(Name)
- Game Modes: GM_(Name)
- Input Actions: IA_(Name)
- Enumerations: E_(Name)
- Tile Maps: TM_(Name)
- Textures: T_(Name)
- Sound Cues: SC_(Name)

There may be a few more I've yet to think of, but they will be named appropriately to ensure consistency across different blueprints in the project.

To optimize efficiency, several precautions will be considered when programming:

- Casts and bindings will be used strictly when necessary
- Event dispatchers will be used to communicate between different actors
- Is Valid? checks will be used to ensure any object retrieval works as intended
- Use of delays will be kept to an absolute minimum, and I will instead use Timers

- Any actors that implement the same functionality will be created as children of a base class so that repetition of code is minimized.

Style Guide

All blueprints will be neatened, aligned and commented once they are working as intended; this will help to ensure that they are as readable as possible for both myself (should I need to change anything) and for anyone else that may need to view the blueprints.

Additionally, events that need to run multiple times will simply be recalled as opposed to rewriting the event each time it's needed.

Commenting Rules

All major chunks of blueprinting will be commented in order to split them up based on their respective functions; this will help to ensure that any code that needs to be changed can be quickly identified as well as keeping the margin for error to a minimum.

To further the identifiability of different blueprint chunks, all comments will be colour-coded based on what their role within the blueprint class is. For example, chunks of blueprint relating to player inputs will be commented in light-blue, whilst chunks relating to gameplay functions will be commented in red. This style of colour-coding will be persistent throughout the different blueprints.

Code Review Procedures

As mentioned previously, code will be reviewed 1-3 times weekly (based on how many updates have been made to the project) to ensure that correct programming practices have been adhered to as well as to check for any possible optimizations that can be made for efficiency's sake.

Production Overview

Moscow

Must	Should	Could	Won't
Changeable character forms	Resource collection (wood & stone)	Unique environmental interactions	AI-based non-playable characters
Watering & Fertilizing plants	Rebuildable monoliths	Simple Tutorial with suitable animations	Objects/Stimuli that interact with the plants

3 unique growth stages for each plant 2 unique songs	Jukebox where the player can select songs Suitable VFX & SFX	A progress bar-based mini-game when watering plants Adjustable camera distance	

Timeline

Task	Start Date	Completion Date
Creating the base map, Character, Controls & Camera	18/12/2024	25/12/2024
Changeable character forms	01/01/2025	08/01/2025
Watering & Fertilizing Plants	01/01/2025	08/01/2025
Creating 2 unique songs	01/01/2025	25/01/2025
Resource Collection	08/01/2025	17/01/2025
VFX & SFX	08/01/2025	17/01/2025
Plant Growth Stages	08/01/2025	17/01/2025
Tutorialization	17/01/2025	24/01/2025
Player Testing	24/01/2025	24/01/2025
Jukebox	24/01/2025	02/02/2025
Monoliths	02/02/2025	08/02/2025
Bug Fixing	02/02/2025	08/02/2025
Camera Zoom	07/02/2025	08/02/2025
Watering Mini-Game	08/02/2025	12/02/2025
Spatial Input Prompts	12/02/2025	14/02/2025
Polish	12/02/2025	14/02/2025

Budgeting

The only money that will be spent on this project is for a 1-month subscription to CraftPix (£15) in order to gather the necessary 2D art assets for the project. Besides this, the project will cost nothing as Unreal Engine 5 is free to use.

In terms of man hours, the project will take roughly 150 hours' worth of time over the span of 8 weeks.