
Murus GDD

Scale - 2021



CONTENTS

CONTENTS	1
TEAM INFORMATION	4
FOREWORD	4
GAME OVERVIEW	5
Game Title	5
Game Description	5
Audience	6
Characters/Roles	6
Lore	6
Delivery	6
Writing	6
Value	7
LORE UPDATE	7
Environment	8
Theme	8
Gameplay	9

Visuals	9
GAMEPLAY/MECHANICS	10
Objectives/Goals	10
Perspective	11
Controls	11
Difficulty & Accessibility	12
flags & infinite time slow.	12
Reference Points/Originality	13
Platformers	13
Speedrunning	14
Custom Content	15
Narrative	15
TECHNICAL REQUIREMENTS	16
Platform	16
Development Environment	16
System Requirements	17
Resourcing/Capability	17
UPSKILLING REQUIREMENTS	17
Programming	17
Mathematics	17
Data Storage	19
File Sharing	20
Input Management	20
Digital Composition	21
DAW	21
VISUALS/ARTWORK/GRAPHICS	22
Importance of Art	22
Why Pixel Art?	22
Style	23
Clarity First	23
Inspiration	24
Distinctive Tools and Techniques	25
Artistic Process	26
Proof of Concept	26
References	28
In-game assets and UI	28
Animation	28
Backgrounds	29
Exporting	30
Font	30
User Interface	30
Focus on Diegetic UI	31

Examples of Other UI Classes	32
AUDIO/MUSIC/SOUNDSCAPE	32
Style	32
Compositional Process	32
FamiTracker	32
Loops	33
Timbre	34
Soundscape Development	34
BFXR	34
Audacity	34
CUSTOM CONTENT	35
Level Editor	35
How it works / File Structure	35
Scale / Move Interface	36
Multi-Selection	36
Quick Select Wheel	37
Ctrl+D	38
Imported Maps	38
UI Interface	38
Name Conflicts	39
Completion Verification	39
Level Manipulation & Security	39
LEVEL DESIGN	40
Campaign Structure	40
Design Process	43
TIMELINE	43
Deadline	43
Timeline	43
Prototyping	43
Structured Approach	44
Crunch	47
Responsibility	47
OTHER CONSIDERATIONS	48
Submission Guidelines	48
Judges	52
Bibliography	52

TEAM INFORMATION

Team Name: Soulcube

Team Number: **C3DA0289**

Team Member #1: Jade Short

Team Member #2: Maxwell Arch

FOREWORD

The Murus GDD functioned as a living document - ever evolving alongside the game. Here you will find outlined the entire project, from original pitch details of its inception, to more recent information & planning for recently created features. This game design document outlines all aspects of Murus's design & aims to give a comprehensive look into the core fundamentals of the project. Due to the living document nature of the GDD, lapses between present, past & future tense may be present - reflective of the time at which the information was appended to the document.



A game design document (often abbreviated GDD) is a **highly descriptive living software design document of the design for a video game**. A GDD is created and edited by the development team and it is primarily used in the video game industry to organize efforts within a development team.

GAME OVERVIEW

Game Title

Our game goes by the name of Murus - the latin word for “wall” - as the entire premise of the game is to scale walls, rather than to navigate using platforms. Canonically, each of the floating earth chunks are titled “Murus” - integrating the name into the game's lore. While most players

won't be aware of the etymology behind the word - it stands out as a memorable singular word that acts as something players to remember the game by, rather than a more literal convoluted title.

Game Description

Murus is an arcade platformer with a twist - it doesn't have platforms! Instead, players traverse a plethora of unique levels by scaling walls, with a "floor is lava" death system. Players must collect all the magical orbs as fast as they can in order to progress through the variety of stages.

Murus features a campaign packed with 200+ levels, featuring over 15 unique mechanics. It also has a robust level editor, where users can design their own stages full of custom levels - allowing levels to go above and beyond the base campaign content. These custom stages can be shared across machines - perfect for speedrunning competitions!

Murus is a customizable experience suitable for all - with configurable difficulty options & controls. Hardcore players can focus on grinding out the fastest times possible, casual players can simply enjoy the world, & creative players can craft anything they desire!

Audience

Murus is a suitable experience for those of all ages - and all demographics due to its G rating & accessible nature. It is mildly catered towards those with prior platforming experience, however younger, less experienced players can play with easier settings.

Characters/Roles

Murus features a suit of playable characters which are unlocked as campaign levels are completed. These characters are purely cosmetic, & do not give any players a mechanic advantage. To create more incentive designs for players to want to unlock - they also occasionally break the canonical cohesion, with cameos of non-fantasy characters. Because of the customizable nature of the player character, the lore itself cannot rely upon the visuals of the character for it's story telling.



Lore

While Murus is not reliant upon it's story, it does feature a loose narrative serving to contextualise the world of the game, which players can experience by playing through the campaign.

Delivery

Due to the open ended, high quantity, level based nature of Murus, telling a story in the traditional sense with character interactions, dialogue, & unique locations is not well suited. Instead, we had to innovate the story telling method - integrating traditional text based journal entries within the game world itself.

Writing

The writing is broken up into two styles - a traditional poetic style used for narration, & a journal entry style. The fragmented nature of the journal entries creates pieces to the broader narrative that can be experienced in a non-linear order, suitable for the campaign being playable in a (pseudo) non-linear order.

Value

The core ideas explored in the story are ones of reconciliation - how people deal with past conflicts, & how people's understanding & appreciation of difference are the key to a better future; otherwise a cycle of perpetual hatred ensues. This is expressed through the allegory of the world's origin, depicting a carer of the land - an age old witch - who is evicted by a new people (marble folk) when they destroy her natural homeland. She retaliates by cursing the sun to beat magic upon the now shadless ground - the grass now freezing those who touch it. In response the highest mages of the marble folk - the marble council - send tremors through the ground, rupturing the planet & sending shards of land airborne in order to provide shelter from the sun. The witch is forced to hide amongst the new people who she is begrudgen to, however after many decades pass she begins to see the marble folk appreciating the land more & learning to respect nature. She too begins to appreciate the splendour that some of the marble folks structures have when combined with the natural - a different kind of beauty. Eventually she decides that maybe there can be a future where their differences can be put aside & can co-exist - turning herself into the marble council & offering to revoke the sun's curse. However the marble council are not representative of the will of the wider marble folk & betray the witch - wishing to perpetuate the hatred as it keeps the marble folk united & under control. The marble council locks up the witch & declares that she was captured plotting an attack against the marble folk - reigniting the hate amongst those in the population. While locked up the witch meets all those who have been unjustly prisoned for also standing to oppose the order of the marble folks society. After many more decades rotting away in her cell, the witch decides to release all of her magical energy into the lands - dispersing it throughout all of Ale Moor (the region), in areas that

only she would know how to find - that is... unless they had a copy of her journal. It is this journal in which the diary entries the player is following are situated - passed down to the player character from someone who knew the witch when she was posing as human, & had stumbled across her hidden journal after she was betrayed by the marble council. The marble council has put out a request for heroes to collect these magical orbs & you play as one such hero - however as you are about to collect all the orbs, the marble council intervenes & locks you up too.

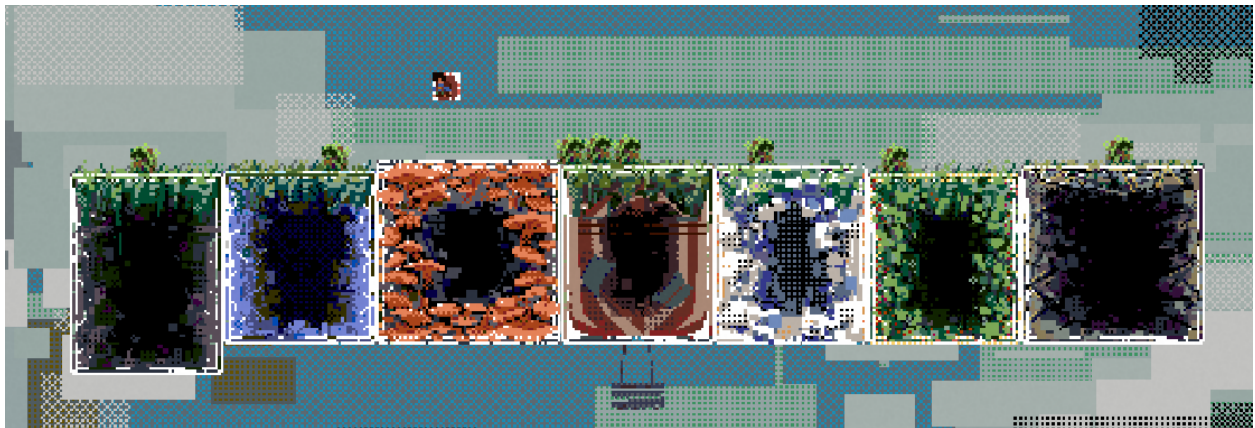
The allegory aims to frame an important message within a pleasant G rated world - so anyone of all ages can experience the moral message.

LORE UPDATE

We thought it would be worthwhile to keep the previous lore plans documented - but alas they did not make it into the final project due a revision of the competition guidelines. The themes presented, while masked in fantasy allegory and holding a positive message about reconciliation - do require some evocative imagery that begins to edge towards a PG rating - allusion to violence is still violence.

Because of this, the game's lore remains inaccessible in game & the in-game text feature was utilised for some handy tutorial popups instead.

Environment



The environment within Murus features a range of uniquely styled platforms against a variety of dynamic backdrops. Each tile type features their own distinctive look to reflect their effect and behavior. See 'visuals' for more.

Theme



Murus interpreted the word “scale” in its literal sense as a verb - to “scale” an object.

1. climb up or over (something high and steep).
"thieves scaled a high fence"

Similar:

Gameplay

The entire crux of Murus’s gameplay stems from the unique movement mechanic - the inability to touch the ground & instead, navigating levels by *scaling* walls. While this is a simple use of the theme, because it provides the foundation for the core mechanics, it is ever present within the gameplay experience. Due to the simple intuitive nature of the integration, there isn’t a

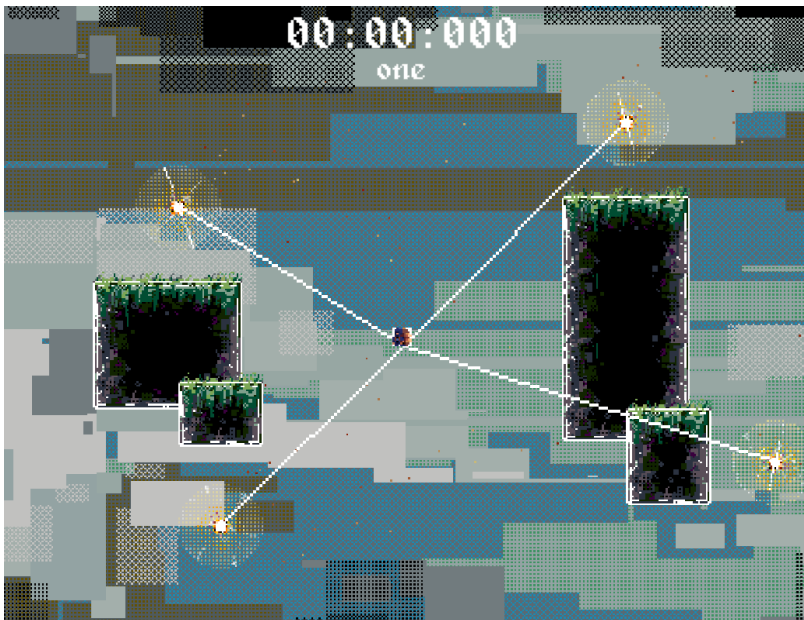


whole lot more to say about it's implementation.

Visuals

To make this task of “scaling” prominent within the experience, we have the player presented as very small in comparison to the viewport size. This makes it feel as if the player is climbing objects much greater than themselves - emulating the experience of “scaling” large objects, rather than just jumping around across objects. This juxtaposition of player size vs scalable object size is used in games like Shadow of the colossus, to emphasize the dichotomy in size between the player and the environment it's navigating.

GAMEPLAY/MECHANICS

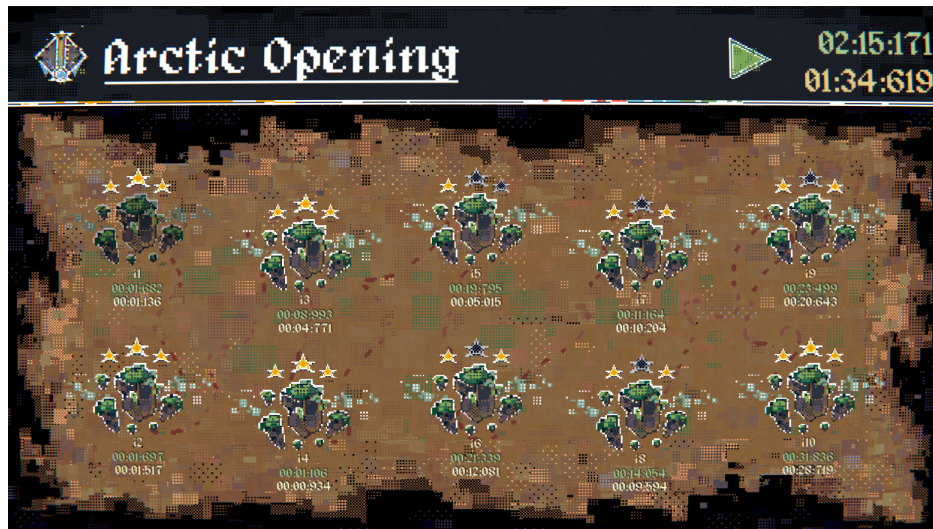


Objectives/Goals

Murus is a platformer with a speedrunning & custom content focus. Within each level, the player is aiming to collect all of the yellow orbs in the shortest amount of time possible. Stars are awarded based on how quickly players are able to complete the level. A second speedrunning category is also featured, known as the full map speedruns - these follow a continuous timer that does not reset upon dying or completing levels & concludes

when the player has completed all levels in a map in sequence.

Both the time scored in the best full map speedrun, & the “golden time” (best time regardless of any resets during a full stage speedrun), are displayed on the map screen - acting as incentive to show players how much possible time they could shave off. In the below example, the player's best time for a mull map speedrun is 2:15 seconds, however the golden time shows that if they had gotten their best times on all the levels with no resets, they could bring that number down to 1:34!



Perspective

Murus is a 2D game with a side on perspective. This makes it easy for all relevant information about the contents of a level to be displayed on the screen at any single instance. If the game were 3D, players wouldn't be able to map out the route to nearly the same degree as the side on perspective allows. Top down also would simply be incompatible with the platforming genres - unable to express height.

Controls

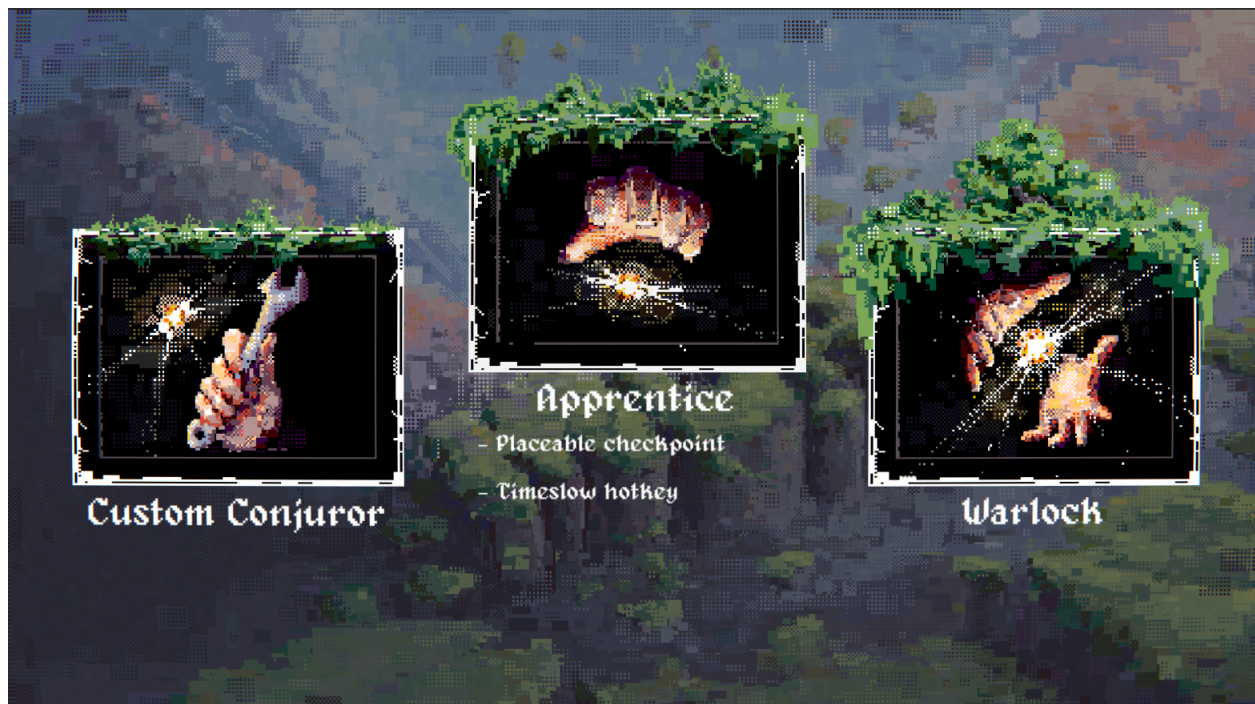
The base controls for Murus are extremely simple, following standard control conventions - WSAD & Space to jump, however almost full control customizability is offered. Custom keybinds can be set in the options menu, where players can customly bind all keys except for the horizontal movement axis - which only works with either A/D & LeftArrow/RightArrow. This is due to technical limitations within the game engine itself, though it is not a major issue as *everything* else can be re-bound. The game also can be binded to work with a controller, moving with the left joystick, & players choosing where the other buttons are mapped to. Other buttons are present within the game with E to exit & ESC to pause (customizable) and left mouse



button to place a flag (customizable) & right mouse button to slow down time (customizable).

Difficulty & Accessibility

Murus is designed to be a challenging platforming experience, however we understand that not all players enjoy playing games in this way, & we didn't want to inhibit them from being able to experience the game. Because of this, Murus features different difficulty options - a standard mode, an easy mode, & a custom mode. If players aren't enjoying playing with the standard difficulty, they can opt into easy mode which grants the players 1 placeable checkpoint per level, & also gives them a button to temporarily slow down time to pull off tricky platforming sections.



Because so many of the levels are puzzle based, we couldn't up the jump height or movement speed or anything like that, lest the "easy-mode" players have a watered down experience as many of the meticulously crafted levels could just be bypassed. The placeable checkpoints & timeslow are the perfect middle ground of making the game easier, but not altering the contents of how the levels are played. For those without the reaction time or co-ordination required to even play on easy mode, a third custom option is available that allows players to go wild with how easy the game is - able to grant infinite flags & infinite time slow.

Within the layout of the campaign map, we also had it follow a branching tree pattern so that players wouldn't get bottlenecked on a single map & rage quit the game - as you progress further into the campaign, players have more and more options for which levels they want to tackle.

To be inclusive towards those with colour blindness, all of the different elements have different visuals - whether that be different textures for the wall, or different particle animations for the orbs. This allows players to differentiate between things without having to rely on colour - making the game more accessible for a wider audience, something that wasn't present during the prototyping phase.

Reference Points/Originality

While the platformer genre may be decades old, & the isolated mechanics within Murus may be mostly commonplace (insta death blocks, single use blocks, bouncy blocks, sticky blocks etc.) - they are all given new meaning when the player navigates the world via scaling walls rather than walking along the tops of platforms. This core difference allows the game to use existing game tropes, but have them be fresh within the context of the different movement mechanic.



Platformers

While platformers are plentiful - the way each famous platformer feels varies quite substantially. Air control, jump height, stopping speed, friction movement speed & character size are all factors that vary across established platforming titles & Murus is no exception.

Murus's movement is most similar to that of Ori & the blind forest, where a lot of the platforming is spent airborne, utilising the environment to navigate levels. The player-determined checkpoint locations (flags in Murus) are also inspired by Ori's unique saving mechanic.

One of the biggest differences between platforming in established platformers like Ori & Celeste, is the lack of player abilities - all variation in platforms comes from interacting with the environment itself. This means that sharp, jerky, routes aren't prevalent within the Murus level design due to a lack of dash ability - with the only way to sharply change velocity being the jump button.

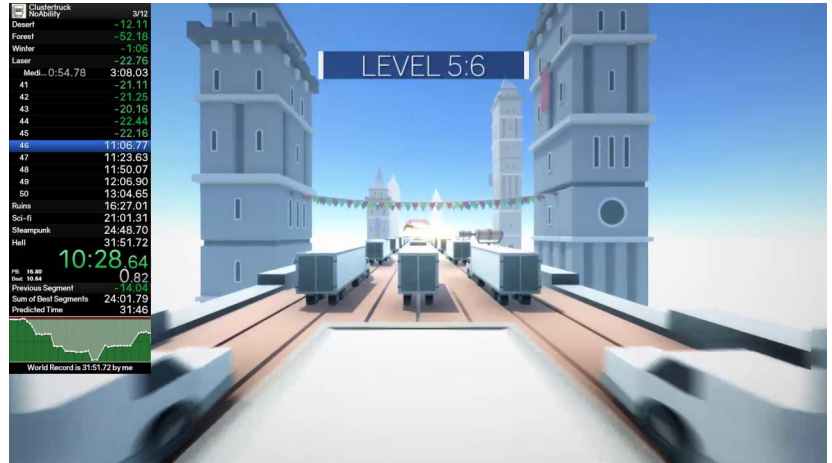
Wall jumping in these games also is handled greatly differently to Murus, in that pressing the jump key while on a wall in Ori/Celeste will propel the player off the wall, with an upward trajectory. However in Murus, the player has to control the arcing motion themselves, unable to jump while in contact with a wall (excluding slippery ice walls) . This creates a little sub game

within many of the campaign levels, in which the player has to come up with routes that allow the player to retain their jump.

Speedrunning

Early in Murus's development, we had to make a decision about the direction of the game - were we going to turn this movement mechanic into a way to traverse a large open world, or was it more suited to short level based gameplay. Upon testing the prototype of this short level based gameplay, a speedrunning scene began to develop itself & so riding off this feedback we decided to lean into that aspect of the game.

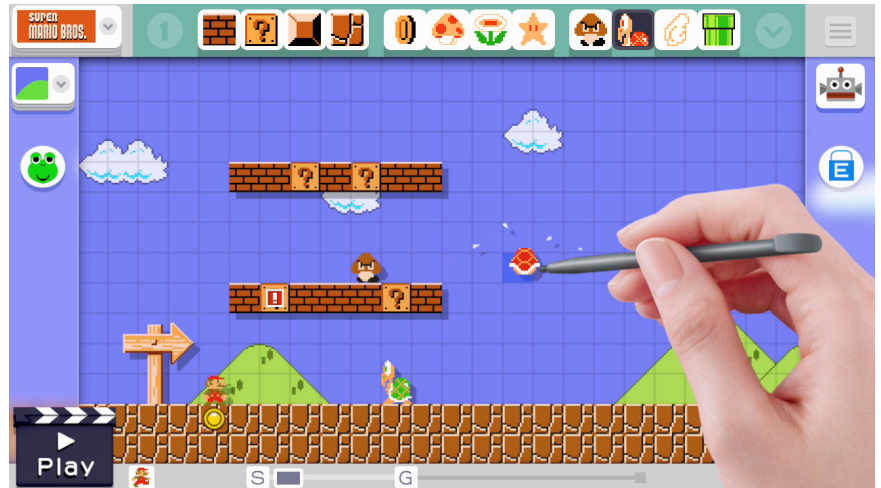
Our prior experience to speedrunning had largely been in the rogue-like genre, where certain games reward & boost players runs if they are adhering to certain time constraints - such as in games like Dead Cells & Risk of Rain. However, the same design philosophies could not be applied to Murus as the speedrunning serves a completely different purpose. In Murus, the speedrunning does not gate any progress throughout the campaign, nor make it easier for those players striving for quick times - it is simply there for those players who find enjoyment out of that competitive aspect of the game, & not something that those who don't enjoy it, need to be worrying about. Casual players can cruise through levels at a snail's pace, but those who are more invested in the game, have in-built support for higher level speedrunning. This type of speedrunning is more akin to something like Clustertruck, however Clustertruck lacks the in-built infrastructure for full map speedruns unlike Murus.



Custom Content

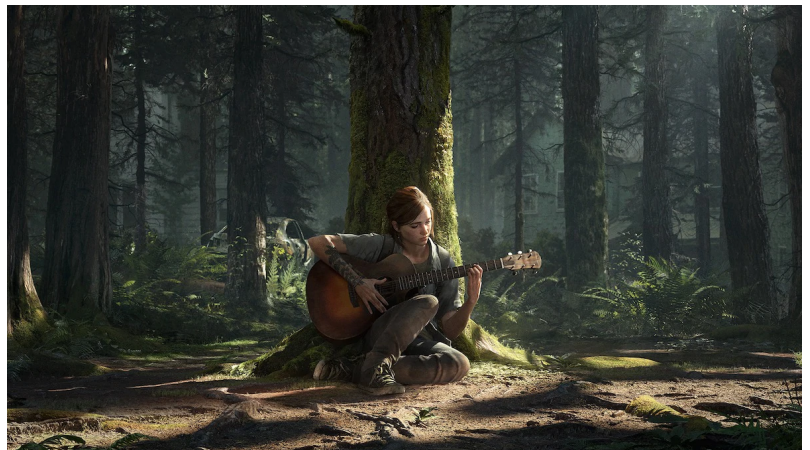
Murus's high skill ceiling and its simple geometric level layouts made it the perfect game for integrating custom level support. The game features a native level editor, which all of the campaign levels were made in - so anything that we are capable of making, the players can also make. This community fueled, custom aspect of the game also

allows players to develop & share levels of whatever challenge level they enjoy playing the game at - more casual players now have an infinite library of casual levels to be developed, & more hardcore players have an infinite library of hardcore levels. This organic way of providing content is similar to that of Super Mario Maker - taking a simple concept, & allowing the players to create things with it. While this aspect of the game is extremely reflective of Mario Maker, as stated previously - the way that traditional platformers (like Mario) play is extremely different to that of Murus, so it's not a competitor on the market.



Narrative

The Narrative present in Murus is a unique one, though it uses storytelling tropes to enhance its own meaning by subverting player expectations. It's quite common for villains nowadays to have some sort of "rational" motivation, so the characterisation of the Witch as a boogiemani villain isn't fooling anybody - everybody is aware there is something else going on. However this misdirection



has players waiting for the predictable twist that the Witch has rational behind her actions, so that when it is revealed that the Marble council are the true villains, audiences are caught off guard.

One of the biggest games that has its own take on a revenge cycle is "The Last of Us 2", which has a more personal perspective, rather than a broader societal one.

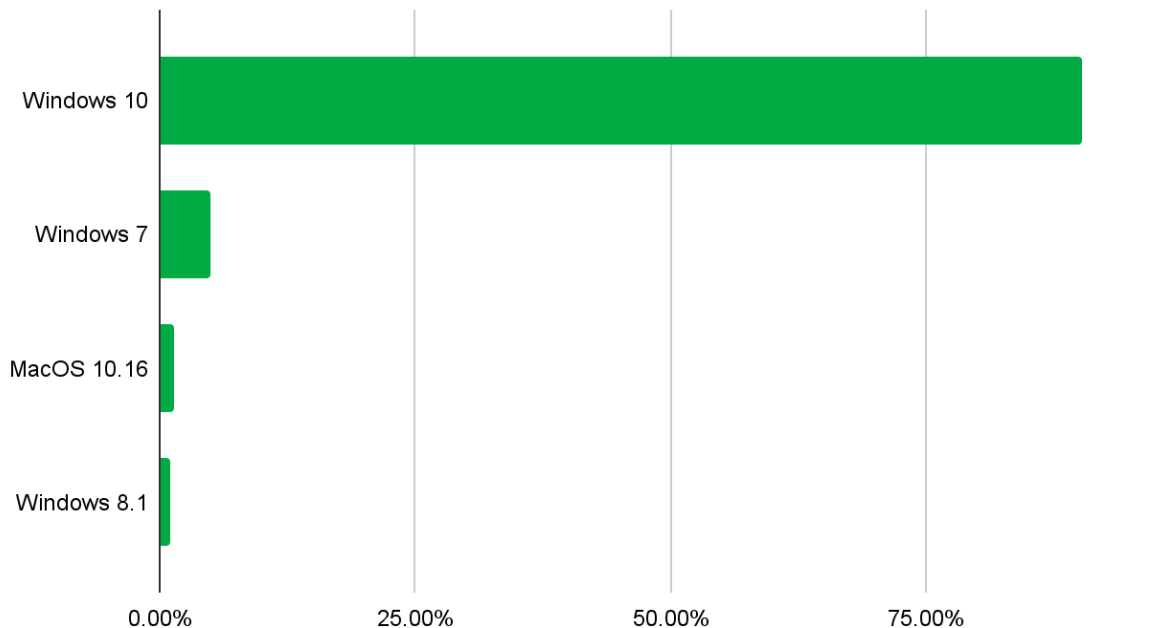
[OUT OF DATE DUE TO NARRATIVE BREACHING G RATING. LEFT IN TO SHOWCASE DEV PROCESS]

TECHNICAL REQUIREMENTS

Platform

The finished product will be able to run on most Windows 10 computers with fairly low specs. This is due to the manipulation of files & save data - which different operating systems handle differently & would require extra processes in place to ensure that everything works as intended across all operating systems. Windows 10 also makes up over 90% of all gaming on the premier storefront - Steam.

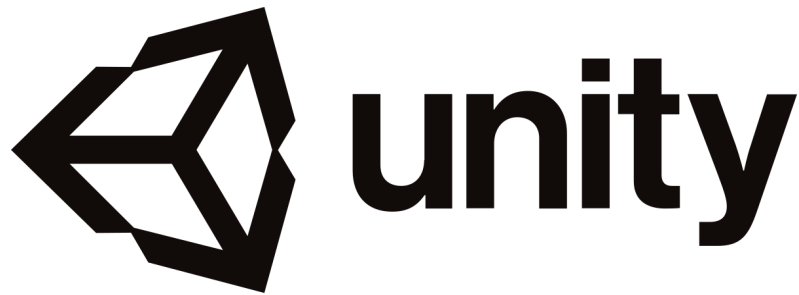
Distribution of Steam gaming platform users by operating System



Development Environment

Murus was developed using the unity game engine as it is one of the most versatile & accessible in the industry. We have prior experience working with it, & are aware of it's capabilities and limitations - so we were aware it was the most suitable engine to develop the game with. Some of its features, including its in-built particle editor, allowed us to easily procedurally generate visuals

that just can't be made by hand. It also comes with built-in JSON utility functions that makes reading & writing JSON to files super easy to integrate within the rest of the workflow.



System Requirements

Murus was developed on a standard gaming laptop - & as such, it can be run on most PC's, & most laptops with specs suitable for gaming. Even though the game only features 2d assets, some processes like rendering the procedurally generated pixel art lines, the parallax background & the sheer amount of particle effects in larger levels can slow down weaker computers. However if these features are causing performance issues, the user can disable them in the graphics settings.

Resourcing/Capability

In order to ensure that the technical requirements are met, we need to test the game across a variety of machines spanning a wide range of hardware. Jade took responsibility for managing the technical specifications as his programming directly correlates to the performance of the game on machines - if things need to be optimised, he knows what to fix. With our early wave of QA testing we had players experience severe infrequent lag spikes, and so decided to begin using the Unity Prolifer to manage which parts of the game are taking up the most hardware use.

UPSKILLING REQUIREMENTS

Programming

While we have had prior experience to game development using C# and the unity game engine - we had never dabbled in many aspects of the games features.

Mathematics

Many of the game's mechanics required some sort of mathematical underpinning - the most complicated of which being the exponential slow of the post-level timer. This problem comes in two parts - the first being that the timer needs to count up from 0 to an unspecified

number (the time it took to complete the level), but it needs to happen within a controlled time frame while also having the rate of increase slow exponentially. The second part is an additional feature that allows us to randomly pick a number from a defined range - allowing the exponential decrease to randomly either stop abruptly or begin to increase again, giving a false sense of hope. This randomness in the level completion screen engages the players, as they can never quite count themselves out as they watch their timer come to a stop.



To implement this, we had to learn how to evaluate definite integrals & use that knowledge to derive a formula that would work for any possible level completion time. This was done by defining our rate of change by the decreasing half of a parabola - emulating the same effect an exponential function ($y = e^{-x}$) would serve. This rate was mapped against time - beginning at 0, & ending at whatever random interval p was generated. By having p lie in a range on both sides of the parabola's turning point, so that we create the chance to abruptly stop the timer's rate of change (stopping before the turning point), or even begin to increase it again (stopping after the turning point). This parabola can be defined as $y = a(x - m)^2$ where y is the rate of change, x is the elapsed time, m is the time that the rate of change flips sign (turning point), and a is how steep the parabola is. To guarantee the area underneath the curve from 0 to p is equal to the time it took to complete the level - we have to adjust a to modify how steep the curve is. For example, a targetTime of 50 seconds is going to need a steeper gradient to reach within 3 seconds of the timer increasing, than a targetTime of 10 would.

We can derive a value for a using the following working:

$$c = \int_0^p a(x - m)^2 dx$$

$$c = \int_0^p \frac{1}{3}ax^3 - amx^2 + am^2x$$

$$c = [\frac{1}{3}ap^3 - amp + am^2p] - [0]$$

$$3c = ap^3 - 3amp + 3am^2p$$

$$3c = a(p^3 - 3mp + 3m^2p)$$

$$a = \frac{3c}{p^3 - 3mp + 3m^2p}$$

Using this, we were then able to translate it to code:

[Running every frame while the timer ticks up]

```
float x = realTimeElapsed;
float y = (a * Mathf.Pow((x - m), 2)) * frameTime;
currentTimer += y;
```

[Runs when level is completed & the timer begins]

```
c = targetTime;
p = Random.Range(2f, 3.2f);
m = 3;
a = (3 * c) / ((Mathf.Pow(p, 3)) - (m * Mathf.Pow(p, 2) * 3) + (Mathf.Pow(m, 2) * p * 3));
```

Data Storage

All our previous experience for data storage had been using binary, & we originally set out to use binary files for the Murus levels, however we ran into a few possible issues. The first of which being due to a lack of things we were able to serialize with binary, the file formatting was a

little whacky & prone to breaking the level completely if one piece of information was missing or entered incorrectly. For every element in the level, its type, position & size needed to be stored, however they had to be stored in separate arrays due to binary only being able to serialize the four base variable type - bool, int, float & string. This system worked fine, except if any value was missing from one of the arrays, everything else would be offset - rendering the entire level broken.

The second issue was that we had little control over what players could be putting in these binary files & we needed a way to prevent possible malicious code being run when the data was deserialized. Upon consultation with some software developers, we concluded that a text based file type was needed - so that no matter what was on the file, it could never be anything more than text.

We settled on JSON as it allowed us to serialize the previously unserializable classes as objects, solving our first issue. We had 0 prior experience with JSON, so we had to upskill a little bit - learning the ins & outs of how to read & write data with C# using JSON.

File Sharing

Game development gets a lot more complicated when external factors can begin to influence the contents of your game - in our case, the ability to load other people's levels into your own game. While we can guarantee that the ID's & names of levels are unique within one's local machine - we cannot guarantee that someone else may share those same values, which obviously causes some issues when importing other people's content. To manage this, when the player goes to import a custom map, before moving it into the maps folder, we have to rename it if its name is already taken, as having 2 folders with the same name causes issues when working with file paths.

We also had to encrypt the level files themselves as they contain a field titled "isBeaten", which is used to verify whether or not the custom level is actually beatable or not & players are not able to export their map unless all the levels have been beaten by them. The issue arises when users are able to go and edit this field in the JSON file manually - so we had to ensure that this was not possible by encrypting all of the custom files.

Input Management

Because Unity's default input manager doesn't allow for changing keybinds, we had to make our own input manager on top of the existing infrastructure. Unity has a "GetKey" function that takes in a keycode as a parameter. We grab this keycode from the settings screen when the player sets up their own custom keybinds - simply waiting for whatever input the game received after clicking on one of the rebind buttons & storing that keycode as a string to be retrieved from

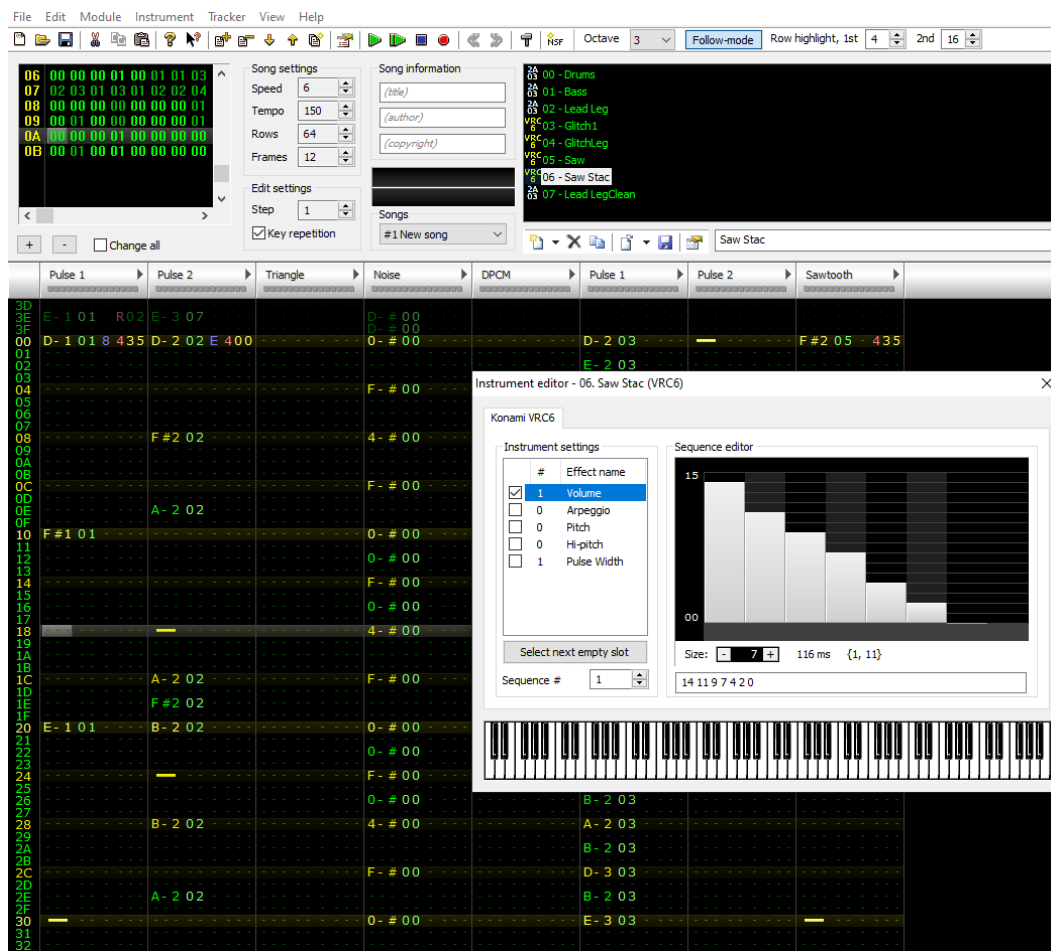
a save. We originally thought that this system wouldn't be suitable to implement due to incompatibility with controller input - though through further research we found that buttons on controllers are valid key codes, despite them not being on a keyboard.

Digital Composition

While we had some experience with physical musical composition - jamming in person with real instruments - we weren't very well versed with composing digitally at a quality suitable for a game soundtrack.

DAW

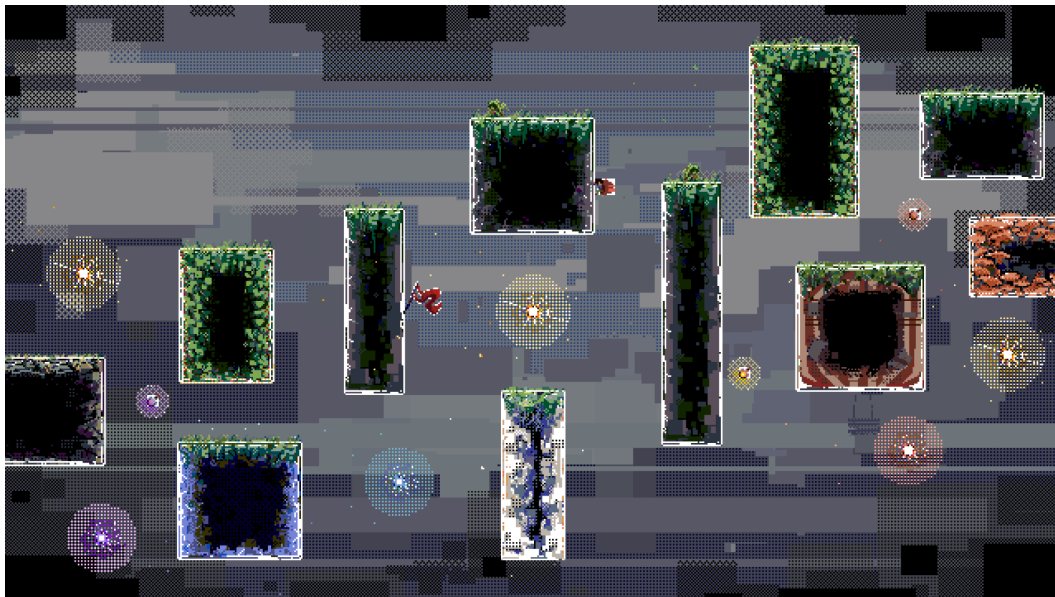
Most Digital Audio Workstations are not free, & those that are, are usually either very unintuitive, or very limiting in the quality of music you can create. We had decided on a chiptune style but the widely used “Bosca Ceoil” was quite limiting in the kinds of soundscape we could create, so instead we decided to learn how to use “FamiTracker”, an old school 4 channel tracker that can export for original NES cartridges! The interface was very different to that of traditional DAWs, so it took some learning to get a grasp on things.



VISUALS/ARTWORK/GRAPHICS

Importance of Art

An undeniable factor contributing to any game's experience is the art and visual components, so throughout the development of Murus we made sure to keep the graphics a priority. Many hours of conceiving, tuning and refining took place in order to add visual appeal and to tie the art in with everything else in the game.



Why Pixel Art?

From the possible digital (or photo scanned traditional) art mediums available, pixel art seemed to stand out as a prime candidate to fill the mold of the prototype for several reasons:

- Since pixel art fully embraces an entirely digital workflow, the process of asset creation is simplified by eliminating additional steps and obstacles that digitising traditional art brings. For instance, while photo scanning physically drawn assets could bring a unique and rare style, its added time consumption would limit and hinder the robustness of the final product.
- Within the range of entirely digital mediums, pixel art is one of the most simple and forgiving ones. The constraints of fitting information into squares along a grid make for a very accessible art experience with little room in terms of exceptionally fine detail

discrepancies such as an imperfect line or an off brushstroke. Even in the case of errors, rather than redoing an entire line or a stroke, parts can easily and precisely be tweaked with less chance of muddying a section of the work. The constraints add a strong sense of tight and purposeful decisiveness to nearly anything on the canvas. There is a largely held myth that this ease is brought with the compromise of visual fidelity which is blatantly wrong if used correctly, leading into the next point.

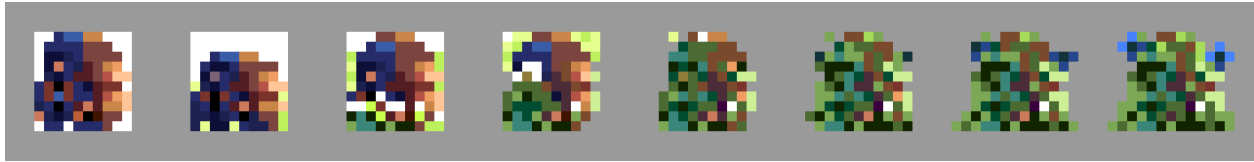
- With the majority of our previous projects being illustrated with a pixel art medium, it felt most natural to follow suit for Murus. Our familiarity with the medium allows us to push and challenge the boundaries in terms of style and technique while maintaining a comfortable 'at-home' grip in this stream of development.
- With the development of an arcadey, old school platformer that breaks many traditional rules and brings more to the table, we figured in addition to the aforementioned factors in support of the medium, pixel art to be the most appropriate one for the job. We went in hoping for the visuals to be mirroring the overall take on the genre; spawning from something old school, but breaking rules and bringing more to the table.

Style

A common misconception surrounding pixel art is its actual standing as a category within digital art, with many people mistaking it for an art style and not a medium. Within pixel art, there are countless styles and infinite possibilities for the manipulation of each aspect of the visuals just in the same fashion as any other medium, and with Murus we aimed to create a compelling and unique trademark look, spawning ideas from an amalgamation of design requirements and diverse inspiration.

Clarity First

Murus's initial prototype was entirely made from clean cut rectangles, allowing for the emphasis on precise gameplay to take centre stage. With this established, the art had to accommodate the tightness on the exactness of its style and adhere to the pinpoint hitboxes with little room for distracting vanity. This 'clarity first' approach certainly drove home the heavy emphasis on a squarish shape language enforced through the white outlining. As well as providing ample visual distinction, these imply a visual rule which states that all encompassed within the outline is a solid element. In other cases of visual aid, extra liberty was also granted with some assets that were unable to adhere to square boxes to reflect the hitbox, where the outline would fill in gaps to create a straight edge. The player character is the most apparent example of this.



[Above is a character sprite sheet for when the player is turned into a plant. Here the hitbox preservation is clearly shown, maintaining form even in the character's squishing motion (frame 2), and with the white fading away to signify the dissolution of the collision.]

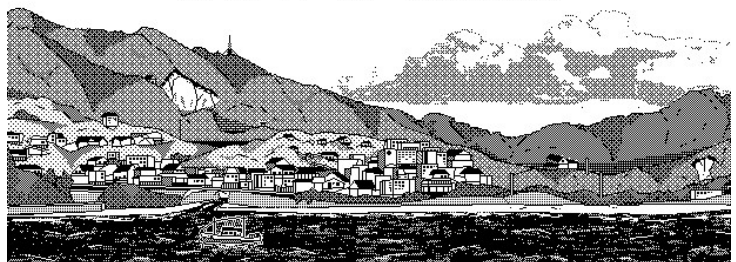
Inspiration



Inspiration for Murus's look came in two major forms: complete/holistic examples including games or artist styles; and content-specific examples such as photographs or single art pieces compiled in the form of boards. The former of the two served to steer the direction of the visuals, with the latter defining, broadening and refining them.

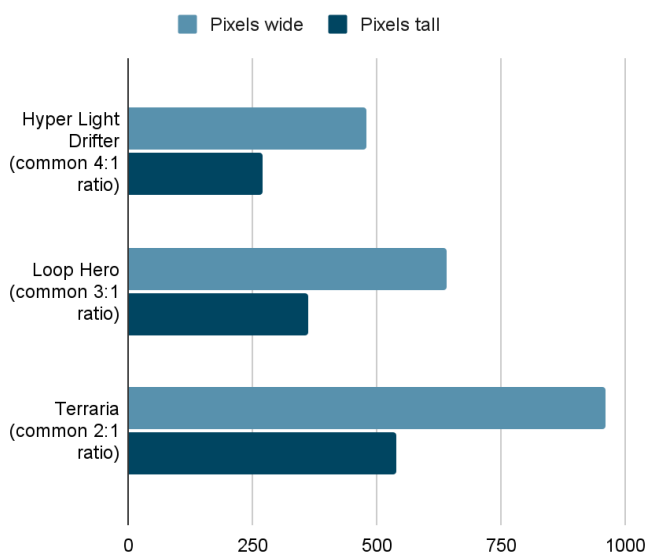
The rogue-like autobattler, Loop Hero (by Four-Quarters and published by Devolver Digital) certainly catalysed the intrigue of liberally incorporating rich colours to Murus's palette. This encouraged the recurring use of blacks, as well as the forceful and hefty dark tone belonging to each corner of the hue wheel. While they are rarer than the common softer colours, it helps to balance the game's visual tone and prevents it from toppling over in an overabundance of cheer and jolliness which may have occluded any serious elements of thematic value. Harking back to the arcadey origins, these tones are also reminiscent of old game console/cabinet palettes such as the NES, with the abundance of high saturation and low bit ranges. While we still don't have to work with the strictly limited palettes of these platforms, some colour conservatism was employed to link it back to retro origins as well as help tighten the consistency of the look. However just like these old limitations, to expand the range of colour available, dithering techniques were widely employed across assets.

恐怖の世界
WORLD OF HORROR



Loop Hero, along with roguelite RPG World of Horror (by Panstasz and published by Ysbryd Games), greatly encouraged the idea of dithering to interpolate colour and value while adding a degree of texture and charm to the art. These games notably brought to light the possibility of dithering based shading, which tends to prove an effective way to help tie in contrasting colours, aiding possible eyesores when it comes to an overload of varying hues and values with sharp cut-offs between each one.

Pixel Art Native Resolutions



In order to accommodate for details in smaller game assets that require higher visual fidelity, especially characters, the pixel resolution was set to establish a fairly high pixel density. Most games are built around a scale in which the x and y values divide into the x and y of a specific HD display and aspect ratio (such as 1920 x 1080). In the case of Murus, the native resolution was established to a 460 x 360 resolution, meaning 1 in-game pixel equates to 3x3 pixels on a 1080p screen (3:1 ratio). This resolution scheme was also inspired by

Loop Hero, as it seemed to strike a good balance of workload and achieved the required fidelity (though other games were taken into consideration for their density).

Distinctive Tools and Techniques

Due to the resolution being considerably large and the asset list being ambitiously expansive, many techniques were employed to assist in reducing the overall scope which ultimately aided in creating a distinct visual style for Murus.

Aseprite features many tools which help streamline or automate actions in a workflow, notably the rectangle tool and the custom brushes being the most effective for the project. Aseprite's custom brushes allow for the incorporation of dithering to freehand drawing (and other tools within the program), bypassing great degrees of manual hassle and saving great deals of time. The rectangle tool (similarly to other programs such as MS Paint or Photoshop) titularly easily paints rectangles, which served to be incalculably useful. By



applying patterns of a Bayer dithering matrix to a rectangle tool, significant portions of an artwork can be illustrated or blended extremely quickly.

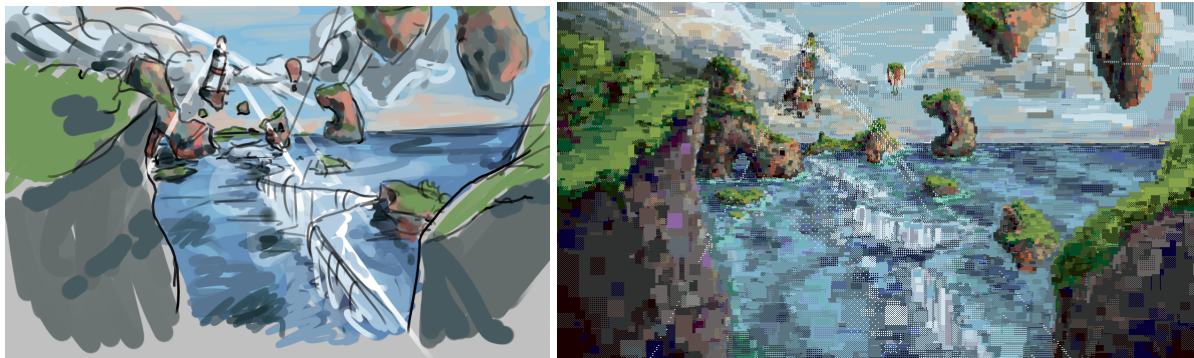
Increased usage of this stemmed from needing to quickly and effectively complete assets (seen mainly in the backgrounds) as crunch periods began to set in. This resulted in a style that can be interpreted to mimic the turbulent and lively nature of traditional paintings while adding a uniquely noisy, yet purposeful wash of texture to serve as a method of cohesion.

Artistic Process

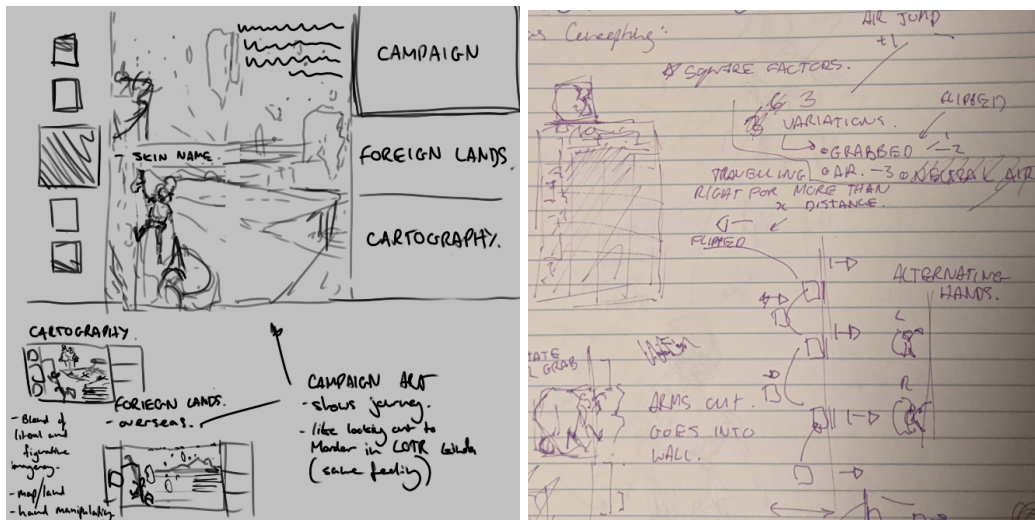
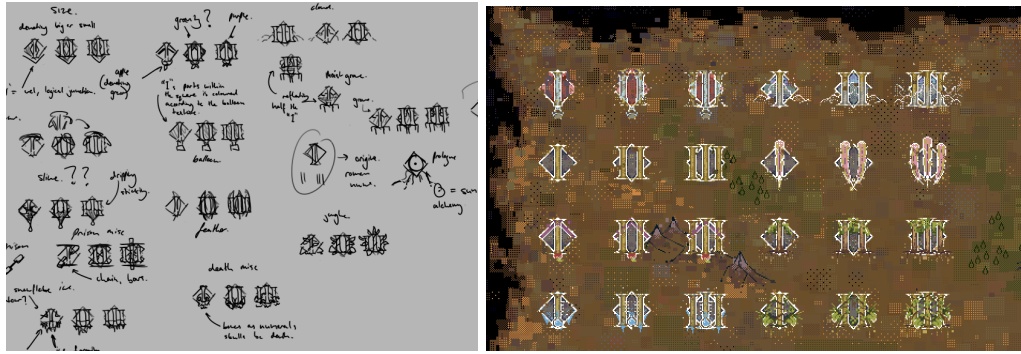
As the range of assets in Murus is considerably diverse, the process for creating the art was not a static method that ranged from piece to piece. While some steps remained nearly identical, many of them varied depending on what the art is for and how they're used in the game. As a result, the following is a generalised look at the art process in Murus.

Proof of Concept

All ideas for assets, whether it's a stage icon, tile design or a menu wallpaper, begin with a sketch; the proof of concept. This quickly and effectively places the idea in an accessible and malleable state which can be both tweaked during consultation between team members and rapidly reiterated to find the best expression of the idea. This stage is rarely done in pixel art, as high-resolution raster or physical sketches prove far quicker and more effective. The extensivity of this process is not fixed, and will depend on the complexity of the concept.



[Concept and final product of the 'Foreign Maps' background. As well as lines, colour is included in the concept due to it being a complicated piece.]

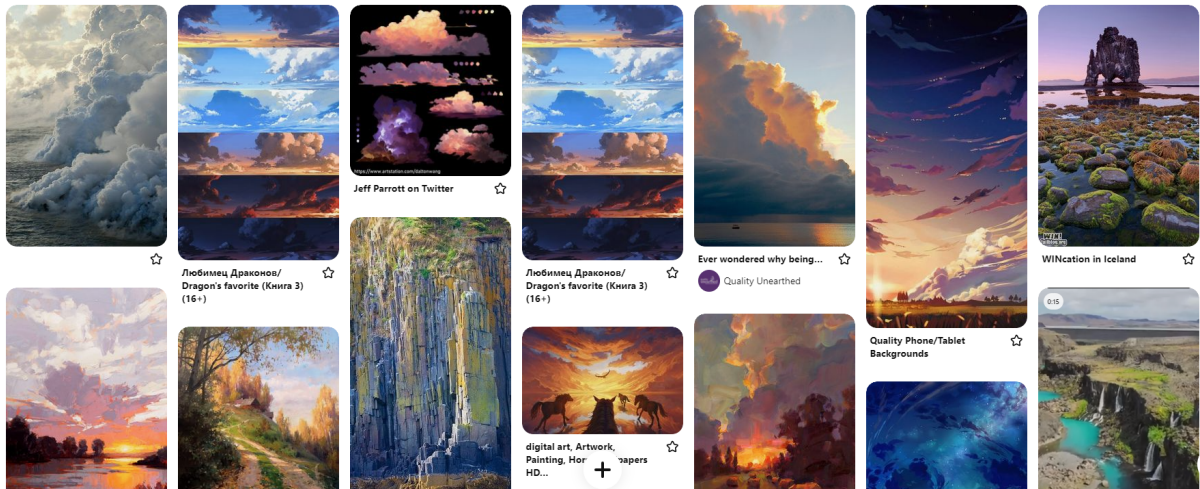


References



At all steps of the concept process, references are used to help keep the art on track, whether it be in terms of achieving realistic form or adhering to stylistic elements. These tie back into the game's inspiration, mainly in the content-specific form; focussing or giving insight into a certain subject. Pinterest and Artsation were used in finding reference photos and artworks, while Pinterest's board feature and Miro were used to help compile and organise them.

53 Pins



[Series of pins used to reference various parts of the campaign and main menu backgrounds.]

In-game assets and UI

To provide a sense of cohesion with the in-game visuals, most of the individual assets were drafted up and created in a single document designed to resemble a frame of gameplay or a complete menu. This side by side view of all the art helps to tweak visuals based not just on how they look individually, but within their respective visual contexts. This also creates a circular and recursive process with the art, where previously finished assets may be tweaked and revisited based on how it looks amongst new ones.

Animation

Complex animations are rarely created for the game, as the duration for making them mightn't be worth the time when a higher level of graphical fidelity could be achieved through a

more polished single frame. Although for the significant parts of the game which need more visual attention, such as the player, orbs, and checkpoint flags, very brief animations are made.

To reduce time consumption, the first asset of its category (such as an orb) will be used as a reference or template for the predecessors. This also creates more visual harmony, consistency and familiarity between each asset. For instance, the



same poses and movements are mostly preserved across all characters in the game, leading to a uniform feel for the player which makes swapping skins a much more accessible process.

Backgrounds

The backgrounds of Murus's UI were planned and created in a way similar to a painting, where theory on composition and colour value was crucial. Backgrounds for the 3 game modes share the same focal point: that being the sun in the 'campaign; piece, the lighthouse in the 'foreign maps' piece and the hand in the 'cartography' piece, creating unity between these 3 screens.



All backgrounds are taken into photoshop to undergo colour correction before being exported. Methods of correction may include hue shifting to rebalance the distribution and abundance of certain colours. During this process, some specific zones within the colour spectrum that appear on the image may be boosted or softened in terms of saturation. Image level sliders are used to

calibrate the tones of the piece, sometimes being applied to specific areas to control their brightness relative to adjacent sections.

Exporting

Different assets require a slightly different process for exporting. Single framed visuals are cut from their mockup stage and put into their own .ase (Aseprite) file, that is then exported as a .png. A collection of frames or multiple similar assets (such as a group of icons) are sequentially organised and compiled into a single image. This 'sprite-sheet' form can be easily cut and unpacked within the Unity editor.

Font

Our chosen font, Alargard, was picked for its medieval feel expressed in a pixel fashion. Some visual issues arose when being used within the game due to the font not being monospace (equal spacing between all letters). These problems were bypassed through coded solutions.

alargard.ttf

murus
murus

The `</mspace>` tag clears all monospace overrides.

Any font can become`<mspace=2.75em>` monospace, if you really want it.

Any font can become
monospace, if you
really want it.

Treating a font as monospace.

User Interface

Many game developers scratch their heads when it comes to designing the link between the player and the game itself, as there can be an overwhelming number of ways to achieve this, and its degree of effectiveness can be difficult to assess. Video game UI integration can be organised into 4 main classes shown in the following bifactorial matrix.

		Is the representation visualized in the 3D game space?	
		no	yes
Is the representation existing in the fictional game world?	no	non-diegetic representations	spatial representations
	yes	meta representations	diegetic representations

Murus's user interface contains elements belonging to most of these categories, with each one being selected for the UI based on what it's used for and how the player interacts with it. Despite this spread, diegetic components are seen to be more prominent in the menu areas.

Focus on Diegetic UI

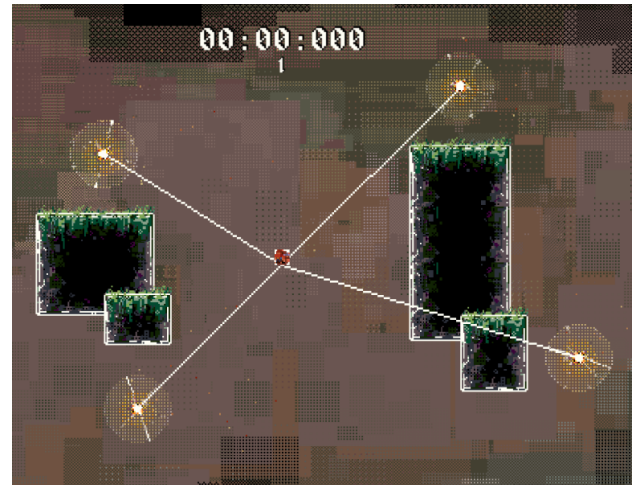
The choice to primarily incorporate the interface under the diegetic class was made for a range of integral reasons considered in favour of the player experience. Having interactable elements (which could all very well be plain buttons) expressed in a way which exist within the game's physical space and within the game's story, prompting both soft world-building, and a deeper connection/interest with the menus. The choice to make the stage selection menu into a physical map helps to do just this; developing the world while keeping it interesting.



Examples of Other UI Classes

The buttons on the main title screen and settings menu are classed as non-diegetic. Each interactable element does not exist within the game space or world.

The orb indication lines that preview at the start of each player's attempt is an example of spatial UI, with the lines directly linking with the objects of the world but remaining unseeable to the character.



In addition, the use of outlines can also be considered another example of spatial elements, where the player is informed of the solid boundaries through visuals that exist to the player but not to the character, also proving that the definition of UI is very broad and can include elements that are part of the game art.

AUDIO/MUSIC/SOUNDSCAPE

Style

The soundscape of Murus was designed to pair with the arcade style visuals - emulating the feel of old school NES games. This was achieved by a chiptune style, bitcrushing sound effects & composing music compatible with traditional 7 channel NES consoles. These constraints - similar to pixel art - holistically allowed the soundscape to fit the arcade style.

Compositional Process

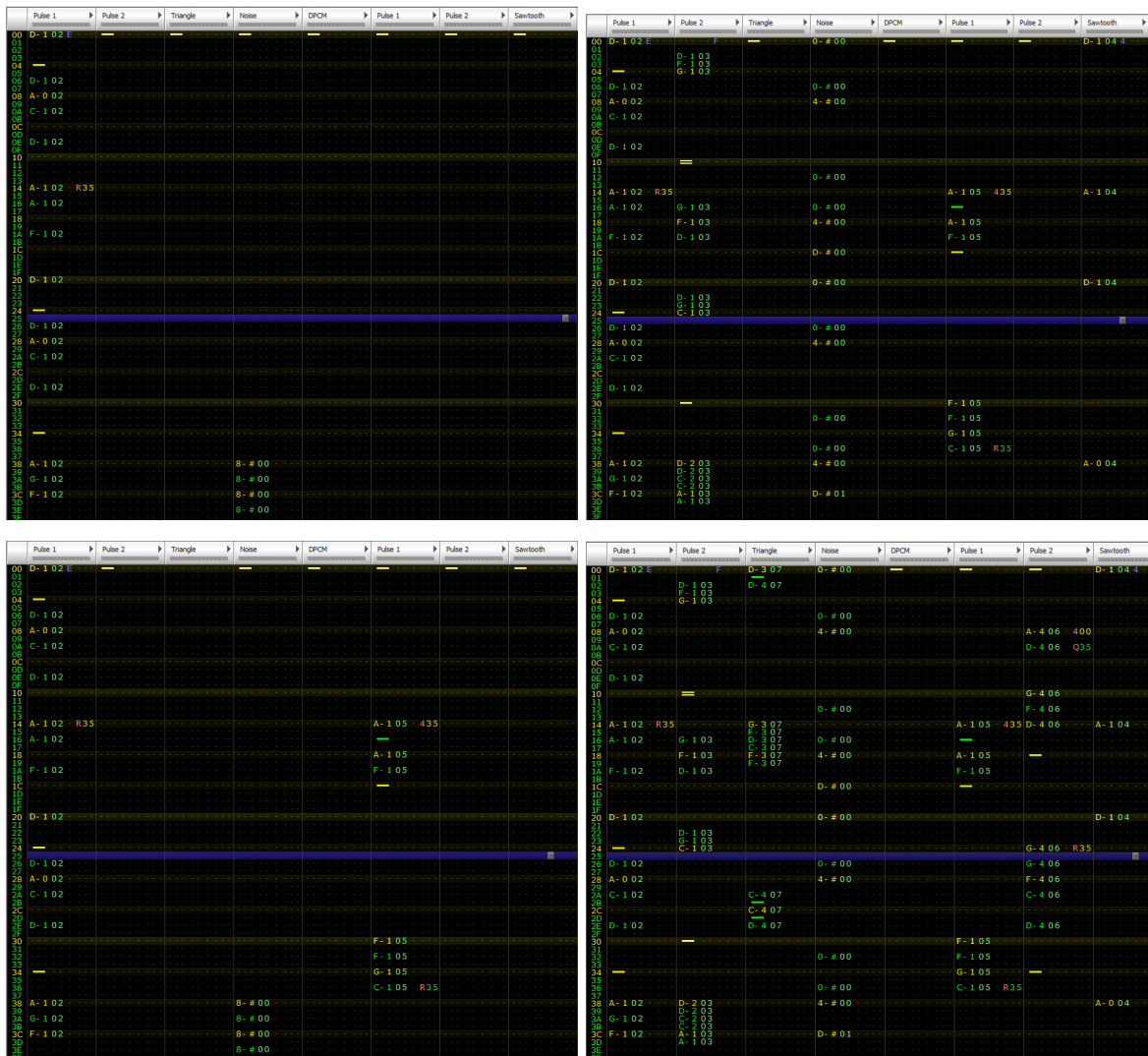
To retain consistency throughout the soundtrack, common processes had to be employed in composing each unique track.

FamiTracker

The Murus OST was composed in a tracker style DAW - Famitracker - using a Konami VRC6 expansion chip, increasing the default 4 channels up to 7 - giving access to 2 square wave channels, a triangle wave channel, a noise channel, a DCM sample channel + the bonus channels of a single sawtooth channel & two PWM waves. This restriction made it that there are not enough channels capable of playing traditional chords - instead chords have to be emulated through fast arpeggiation.

Loops

Each of the loops within the OST follows a loosely similar structure in order to hold consistency over the soundscape. They begin with a single melodic line that is joined by bass & drums. Sections then last for 2 phrases each, usually phrases of 4 bars - adding new melodic layers to total a texture density of around 4-5 layers, now looping a polyphonic phrase, before dropping back off to a thinner, monophonic or homophonic section. After the drop off, the loop is usually followed by a sforzando where all the layers come back in at once, usually adding a couple more layers, capping out at the texture limit, before returning back to the loop.



Timbre

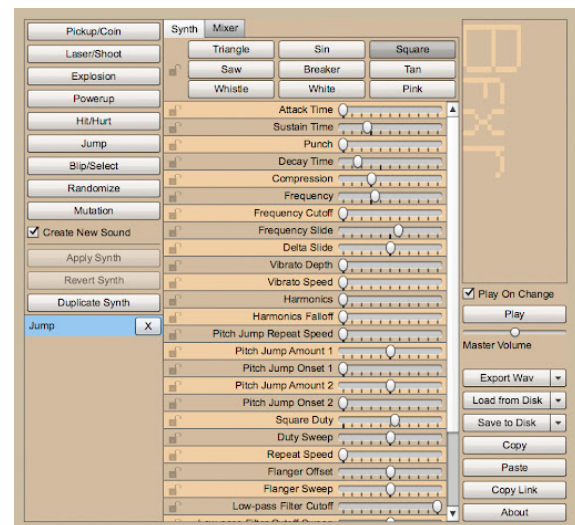
The OST features a set of recurring instruments, as the audio environment is developed using familiar tone colour. This gives the soundtrack a sense of cohesion, making each track feel like it belongs within the same game - with reflection of the map location reflected within the melodic ideas, rather than tone.

Soundscape Development

Similar to the soundtrack, we needed to make sure all of the sound effects had some sense of consistency - ensuring that the arcadey menu sounds were cohesive with the sounds of the player interacting within the game world.

BFXR

The artificial synth based sounds were used to create the majority of the menu sounds, & a large portion of the in-game sounds. Many of the sound effects consisted of a single wavelength, manipulated in a variety of ways, including wave type, compression, bit crush, frequency, attack etc. BFXR also features several parameters that allows settings to change throughout the playing of the effect itself, including a pitch jump sweep, flanger sweep & frequency slide - allowing sounds to feature even more variation. Due to the artificially synthesised nature of these sounds, within the Unity game engine itself, many of these effects have a randomised pitch offset to provide variation to many of the repetitive synth blips.



Audacity

All the sounds reflective of the real world were created with Audacity - a free tool that allows users to record sounds & manipulate them using a variety of effects - namely distortion & bitcrushing. This allowed us to make real word sounds feel like they belong within the arcade, chiptune environment.



CUSTOM CONTENT

Level Editor

The core feature that pushes Murus's content over the edge, is the inclusion of the in-built custom level editor. The level editor infrastructure makes up a large portion of the entire project's codebase itself. While the raw fundamentals of the editor were simple to create, adding commonly expected quality of life features consumed the majority of its development.

How it works / File Structure

Each element within Murus (wall/orb/potion), has an element class deriving from ScriptableObject containing an element ID, & a reference to the object to spawn into the level, as well as the object to spawn into the editor. When selecting an element from the toolbar, the game retrieves the element class based on the element id ascribed to the toolbar icon & instantiates the corresponding object in the scene - adjusting it's position & size relative to the user's input & appending it to a list of "spawnedObjects". When a user goes to playtest the level, all the objects in the spawnedObjects array are converted into data for the save file to read - converting their position, size, id etc. into serializable data. When the level is loaded within the game scene, a similar process is undergone - this time retrieving the serialized data & converting it into a list of the element classes - then looping over that list to instantiate each element in the scene & assign it's saved position & size values.

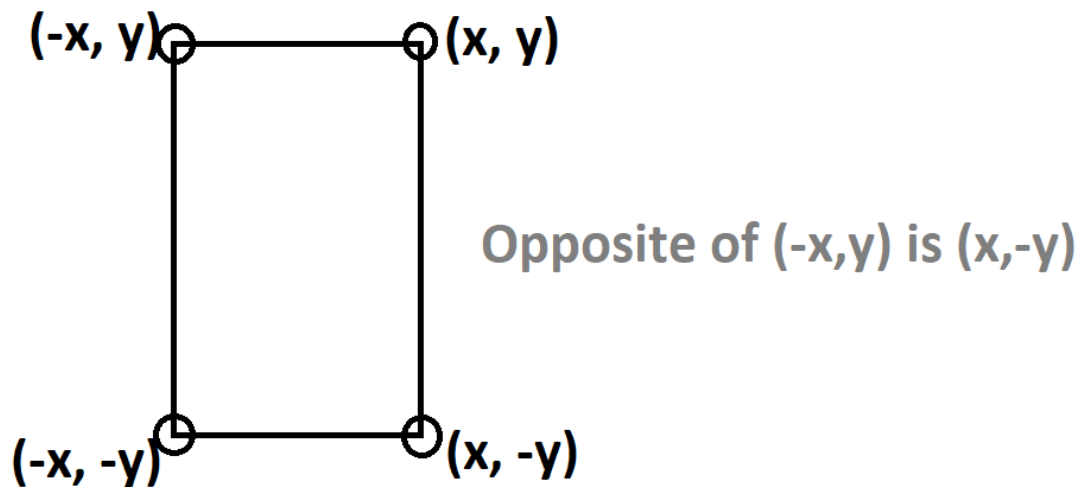
```
[CreateAssetMenu(fileName = "newElement")]  
@ Unity Script | 13 references  
public class Element : ScriptableObject  
{  
    public GameObject previewObject;  
    public GameObject spawnedObject;  
  
    public int id;  
    public string elementName;  
    public bool paintable;  
    public Sprite wheelIcon;  
    public Sprite previewIcon;  
}
```

```
[System.Serializable]  
9 references  
public class ElementData  
{  
    public int elementId;  
    public Vector2 pos;  
    public Vector2 size;  
  
    1 reference  
    public ElementData (int id, Vector2 p, Vector2 s)  
    {  
        elementId = id;  
        pos = p;  
        size = s;  
    }  
}
```

Scale / Move Interface

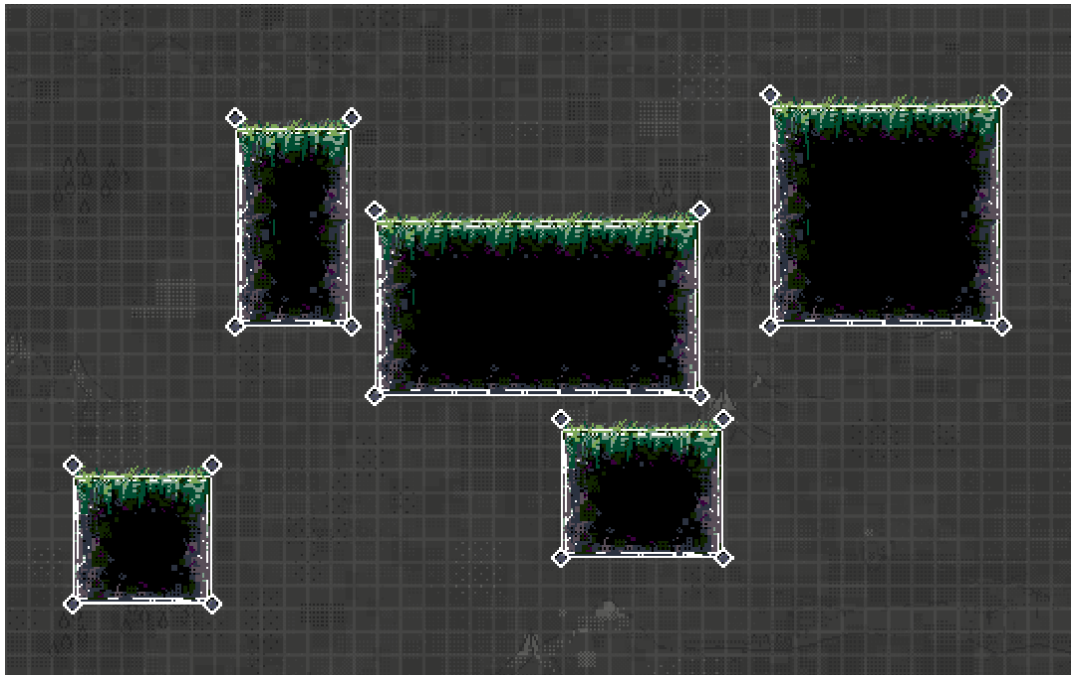


Players can select existing tiles to edit, as indicated by the corner nodes that appear when highlighting the tile. These nodes can be dragged to resize the tile after they've been placed - a hugely requested feature from the playtesters as it's a common practice. For the existing tile scaling code to function on pre-existing tiles, the position of the opposite corner was needed to act as the pivot point of the scale. This corner position could simply be calculated by flipping the signs of the clicked edge node's local position relative to the tile itself.



Multi-Selection

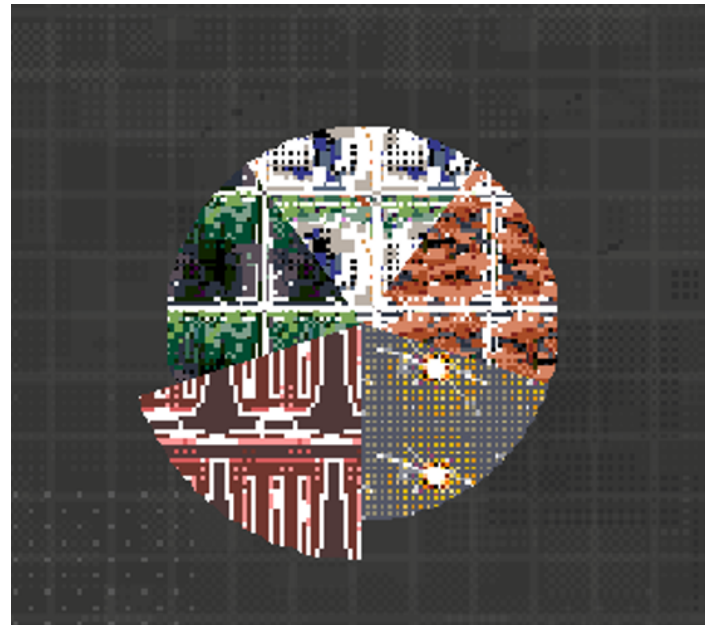
If users want to move larger sections of their levels at once, they can use a multi select feature simply by holding shift & selecting additional elements - emulating the workflow similar to most standard editor software. The selected groupings then all move as a whole when dragged into a new position.



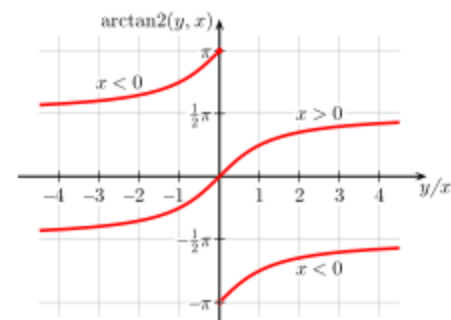
Quick Select Wheel

To further improve ergonomics, a quick select wheel allows for players to swiftly switch between tools. By default the wheel features the standard wall, the pickup orb, & the delete tool - however it is then populated with any other elements that the player is currently using in the scene, drawing from them like a palette.

Radial fill images are going to be used to divide the quick select into sectors - the angle from the circles origin to the player's mouse determining which sector to select. This can be calculated using the following code, utilising the Arctan2 function combined with an additional check to account for angles in all 4 quadrants.



```
float localAngle = Mathf.Rad2Deg * Mathf.Atan2(origin.x - mousePos.x, origin.y - mousePos.y);
if (localAngle < 0)
{
    localAngle += 360;
}
```



Ctrl+D

The final planned ergonomics feature is the Ctrl+D hotkey, allowing users to duplicate the exact properties of any existing element, making it easier to replicate certain sections of levels & create nice, symmetrical & even levels with ease. This is simply done by spawning in a copy of the currently selected object by using its element id to identify what to spawn - then copying over all the same size & position properties.

Imported Maps

Since custom content is such a core part of Murus, it's important that it is handled properly so that there aren't issues down the line. Playing other people's maps needs to be a secure, robust & intuitive experience & so there are several factors that need to be considered.

UI Interface

The time spent playing imported maps is going to far exceed the time actually spent importing new ones, so the user interface shouldn't clutter screen elements relating to importing. To address this, all the importing related UI is tucked away under a toggleable sidebar that can extend when the players want to import new maps, and otherwise be left aside. Players should also have a way to sort their large collections of imported maps, so we've allowed them to position them however they would like upon a huge canvas, similar to the campaign layout. Users can hit a snap to grid button if they would like the maps sorted in a traditional grid fashion, but

otherwise they can distribute them however they wish.



Name Conflicts

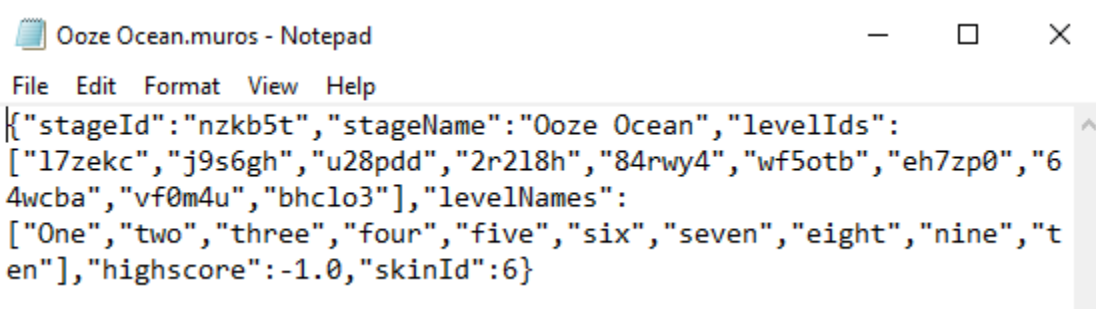
While the system can ensure that users aren't able to create maps with duplicate names, we cannot control the names players across devices use. Because of this, if two folders exist with the same name in the imported maps directory, issues can arise as the computer cannot differentiate between the identical paths. To address this issue, the map needs to be renamed before it is cloned over to the imported maps directory, simply appending a number to the end of the name - eg. "Mushroom Land" becomes "Mushroom Land (1)" which becomes "Mushroom Land (2)" etc.

Completion Verification

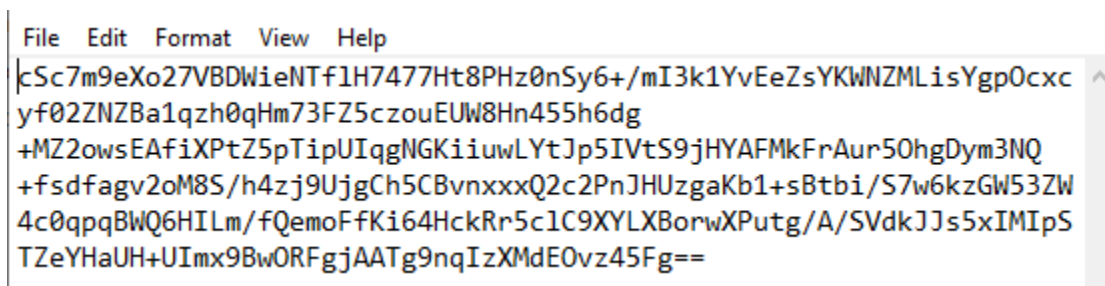
To ensure that players cannot import another player's map, only to find that it's literally unbeatable - players can only map if all the levels have been completed & thus verifying the map as beatable. If a verified map is altered, it must undergo verification once again before exporting.

Level Manipulation & Security

To stop players from tampering with fields such as the map ID field, highscore field & the isBeatable field - all the files are encrypted and thus unalterable by the users.



Unencrypted Map File



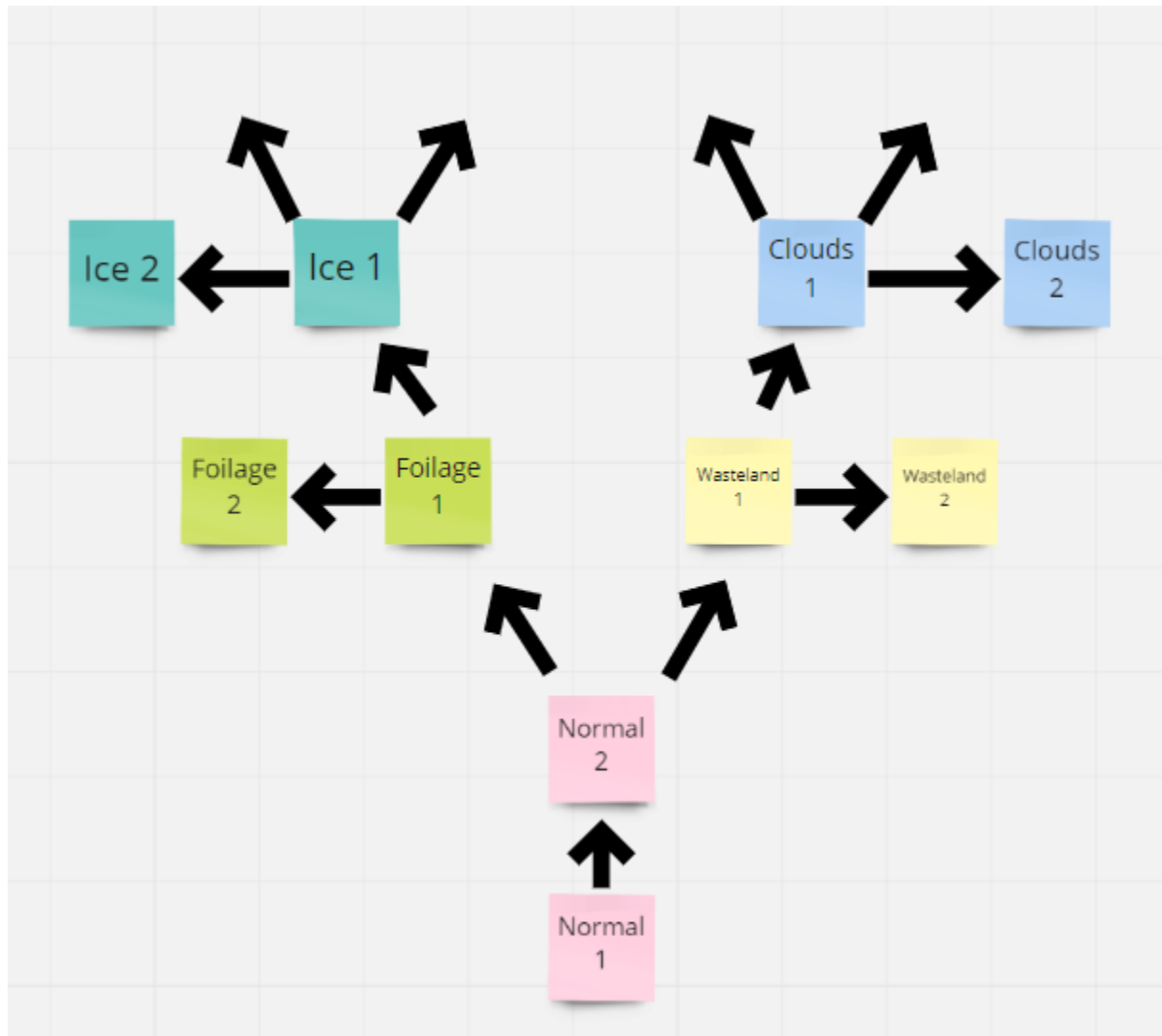
Encrypted Map File

LEVEL DESIGN

Level design can make or break Murus - interesting mechanics mean nothing unless they are paired equally as interested levels that take advantage of such mechanics. Each level is to be unique, & each map should have it's own thematic flair to it's design.

Campaign Structure

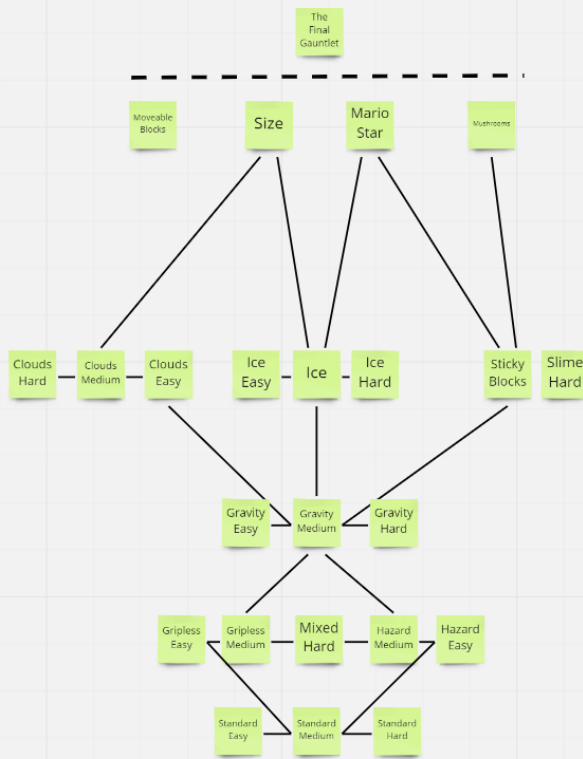
Within the broader structure of the whole campaign, we wanted to offer a variety of level designs, while also easing the player into mastering each of the many different mechanics present within the game. New mechanics are introduced in a branching tree pattern which means that new mechanics are introduced at an increasing rate, so that as the player becomes more skillful at the game, they are able to more adequately adapt to new mechanics. While each new branch introduces a mechanic in a map that must be completed to progress throughout the game - optional harder maps are provided for several of the earlier stages that expand upon the core concepts & them to their limits. We didn't want to discover new mechanics behind these challenging maps & so to provide some alternative incentive, many of the game's skins are unlockable by completing these maps.



A larger sum of maps was originally planned, with 3 tiers for each unique mechanic (hence development of 3 tier assets), however this had us stretching concepts too thin & diluting the experience with repetitive content.



Campaign Plan



Design Process

- Drop down some randomly scattered large landmarks
- Position objective orbs throughout
- Create dynamic pathways between the larger landmarks

TIMELINE

Deadline

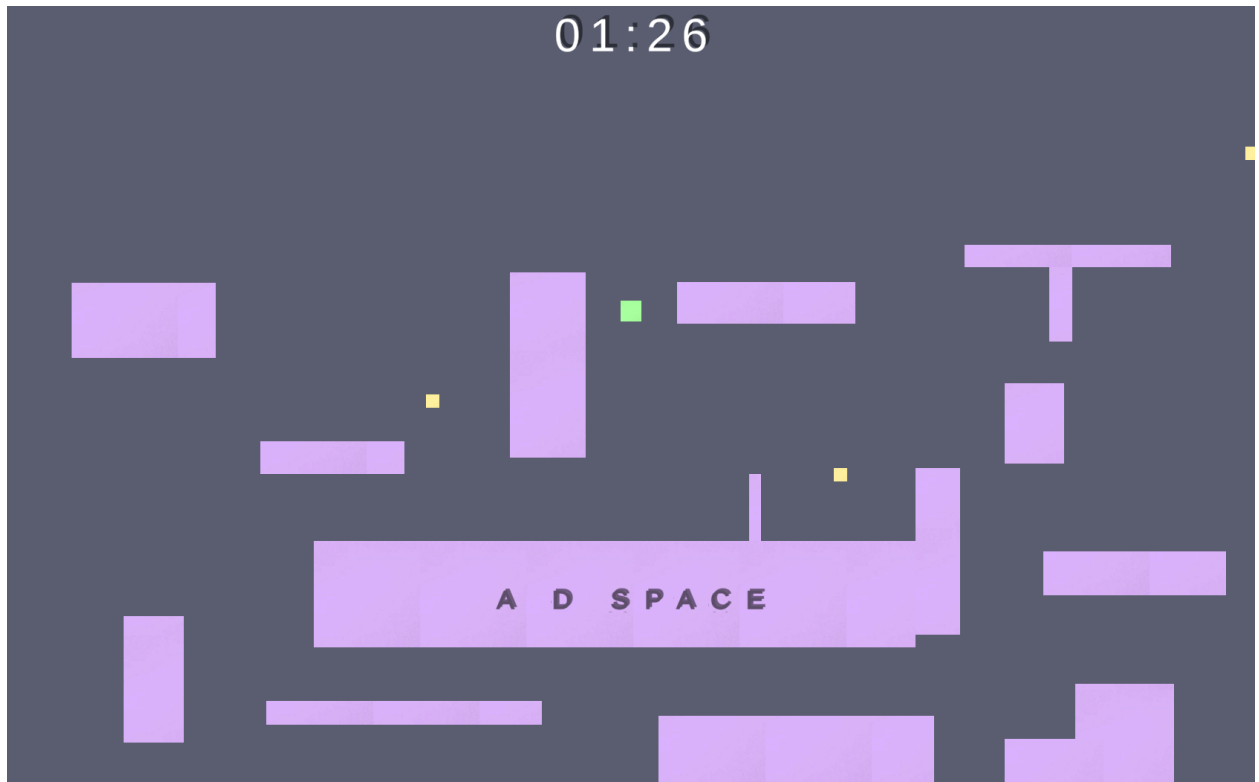
While the game needs to be submitted by the 5th of August, we aim to have a polished build out at least a fortnight prior, giving us adequate time to address any unforeseen issues & fix any last minute bugs. This also gives us time to revise this GDD document with updated information regarding how the game has developed since its first inception.

Timeline

The development timeline for Murus is a little all over the place due to the scattered nature of the project. Initially, the game started as a little tutorial project that we developed in a few hours to showcase platforming mechanics in unity to our students back in March this year - with no intention of entering the STEM games challenge. However after talking it out, we concluded that even though we would be set back by a year of development compared to other competitors, it was our last chance to give the challenge a shot.

Prototyping

The original prototype for Murus was developed in this time frame where we had no intention of taking the project further - & it was largely a solely project developed by our programmer. This prototype featured simple blocky “graphics” & laid down the core mechanics in a few different stages. The game had no visual theme & was simply “blocks that can’t touch the ground”.



In this early prototype, light orbs did not reset upon player death, instead, the timer was just added a penalty. This design was scrapped later down the line due to it restricting how creative we could be with level design due to a lack of control over the state the level could be in at any given time - players could make it $\frac{3}{4}$ through a level & restart, only to have it be impossible to get to that same point again without using the light orbs.

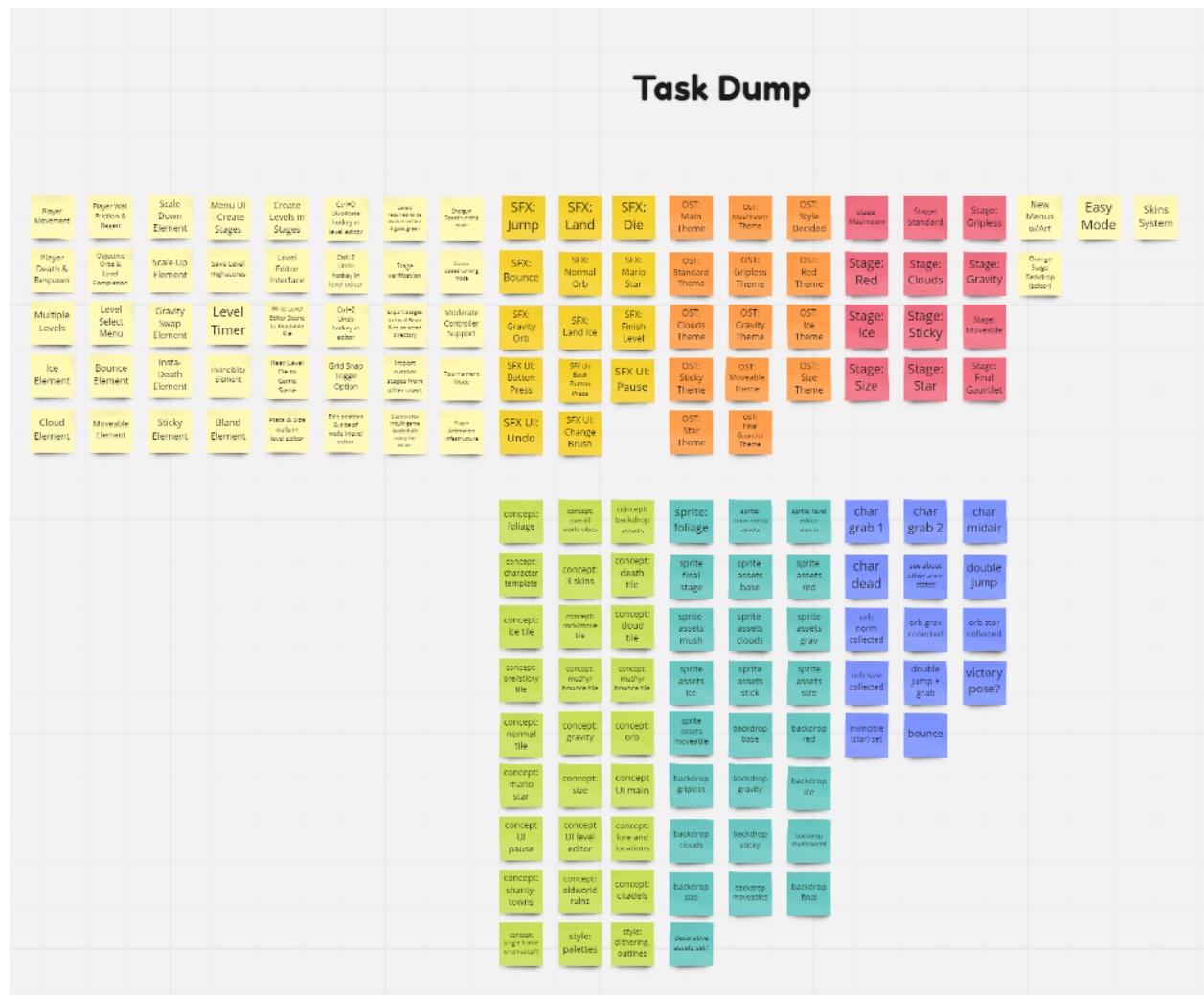
Structured Approach

The decision to pivot Murus's development into an entry for the STEM games challenge was quite an easy one considering how perfectly the mechanics already fit with the theme - all that was needed was to make it more visually clear that the character was "scaling" walls. The Solid State drive with Murus on it had recently died when making this decision, so we had just under 4 months to develop the game from scratch to meet the deadline.

We first focused on replicating everything that the previous prototype had, translating it over to a



After 2 months of re-constructing the game to resemble the prototype with a more robust framework, the concept for Murus's world & style began to develop. We set aside all the remaining tasks required to move the game from this perfectly playable prototype state into a playable polished product. We used the virtual whiteboard tool "Miro" to handle our development timeline-

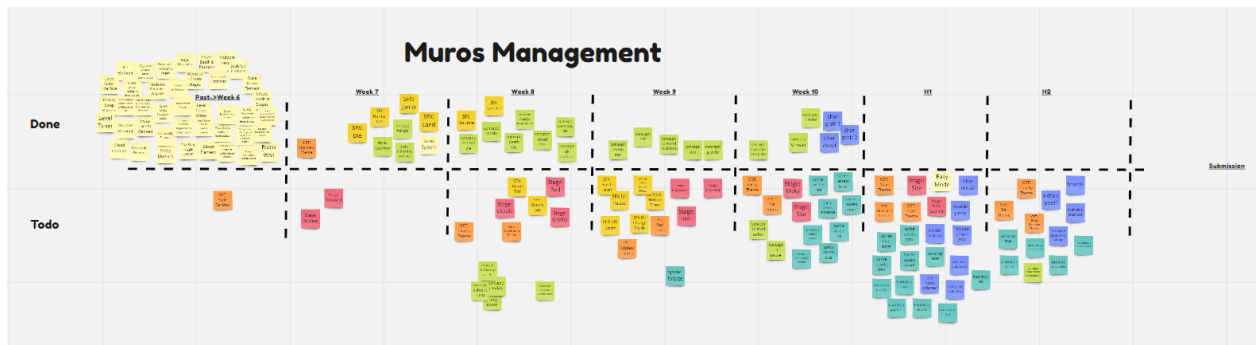
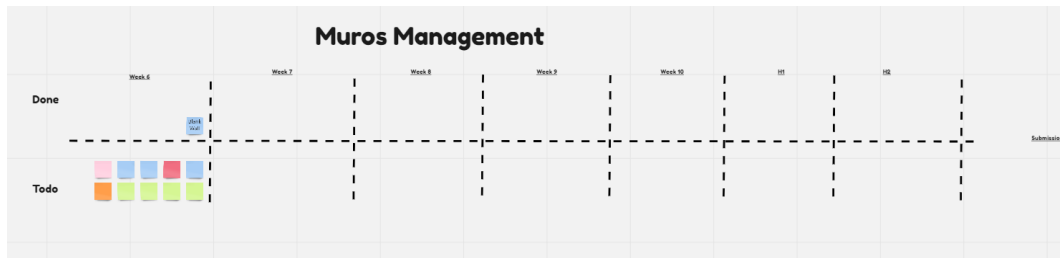


We gave ourselves until the end of the school holidays to develop a finished product - totalling a time span of 7 weeks to turn this-



Into this-





These tasks were spread out across a 7 week span, however we grossly underestimated how much there would be to do, & by the end of our self ascribed deadline, we didn't have anything close to fully polished. This forced us to cut into our playtesting time to continue development which we hope hasn't had a detrimental impact in identifying hidden bugs.

Crunch

This forced us into a crunch for the final month - spanning throughout the month of July. We had been prepared for this possibility since making the decision to enter the comp - knowing that trying to make a full game in 4 months was no easy task alongside school 5 days a week - & so Murus was designed to be very scalable in scope. To add more content, all we do is implement another element with some default & stages & just like that the base games content skyrockets exponentially, as each new element can be combined in levels with existing ones for an increasing amount of possibilities.

Responsibility

Since the development team is so small & both members of the team fill starkly different development roles, communication wasn't much of an issue during the organisational progress. Discord was used as the main means of communication between the team, exchanging simple DM's each weekend to recap on how development was progressing - & if any problems had arisen. Both team members took responsibility for completing each of their tasks individually, with no need for an



overarching project manager. This was doubly true due to the fact that we have worked together in smaller game jams for several years & aware of the collaborative workflow.

OTHER CONSIDERATIONS

Submission Guidelines

1. **All games entered must be suitable for play by all age groups, and must conform to the 'G' rating descriptor as issued by the Australian Classification Board, and the 'E' or 'Everyone' rating descriptor as issued by the Entertainment Software Rating Board.**

Considering Murus began as a game depicting flat coloured rectangles, the thematic content was clearly not integral to the overall gameplay loop - allowing us to safely develop a G rated game suitable for the STEM games competitions guidelines.

2. **In addition to conforming to the above 'E' and 'G' rating descriptors, all games entered must comply with the following specific expectations with regard to game content:**

i. Violence

Very mild, comical violence is acceptable but any violence depicted in the game must:

- have a low sense of threat or menace,
- contain no use of visible blood or gore,
- be justified by context, and;
- must not be realistic in nature, or imitate any real-life scenario.

No violence is present within the game - the player "death" animation is simply them transforming into a planet like the midas touch, with no threat/menace, no blood/gore, justified within the games world & the "floor is lava" mechanic, & is certainly not realistic in the slightest. The skull themed death counter was removed from the game as we didn't want to categorise "restarts" as "deaths", as the concept may be unsuitable for younger players.

ii. Sex

Games shall contain no references (implicit or implied) to sexual activity of any kind.

No sexual content

iii. Language

Games shall contain no profane, crude or coarse language of any kind. This includes colloquial terms, slang terms and profanity in languages other than English. Games must not contain any content that is defamatory.

All the writing in the game is suitable for all ages & can be labelled as none of the unpermitted things above.

iv. Drug Use

Games shall contain no references (implicit or implied) to drug use of any kind. Please note that this includes illicit and illegal substances, prescribed drugs and legalized drugs (such as caffeine, nicotine and alcohol).

The game contains no references to drug use. There is the single use of the word Fungal “Brewery” in the context of the name of the map where the bouncy potions are unlocked. This re-contextualises the word, which already contained no reference to drug *use* - simply the alluded *creation* of what is traditionally beer. Murus does not feature a single drug, nor reference to any.

2. make (tea or coffee) by mixing it with hot water.

"I've just brewed some coffee"

Similar:

prepare

infuse

make

mash

be in preparation

stew

mash

v. Nudity

Games shall contain no nudity of any kind.

The game contains no nudity of any kind.

vi. Themes

Games must not endorse, suggest or advocate for any of the following additional themes:

- Gambling (simulated or otherwise)
- Discrimination of any kind
- Illegal activity of any kind
- Impersonation of specific real life people, including public figures.

The game doesn't endorse, suggest or advocate for any of the banned themes.

3. Games submitted for judging in the Australian STEM Video Game Challenge:

- must not contain any intellectual property, including footage, images, artwork, programming or sounds that are not created by the Applicant unless such content is included as part of the Game Engine; and
- must be solely the Registrant(s) original work and must not be created in collaboration with any other individual or entity

All intellectual property within the game was created by us & as such belongs to us. We were unsure if this rule extended towards the use of open source packages required for certain application features within the engine, though as stated in this blurb, us using a free open source package to access the windows file explorer in unity will not disqualify or omit our submission.

Submitted games should refrain from the use of store-bought or purchased assets

The Australian STEM Video Game Challenge is intended to build and encourage capabilities in a number of areas. The development of characters, construction of environments and the realization of artwork is an important component of the game development process, and part of the challenge that exists in creating an original game.

While many game development platforms offer an ability to purchase/utilize store-bought assets, or assets developed by a broader community of users, the Australian STEM Video Game Challenge wishes to discourage the use of these assets for the purposes of the competition.

We do, however, recognize that in many cases participants in the Australian STEM Video Game Challenge are developing games for the first time, and in these cases the use of purchased assets may be advantageous to the building of skills and to the understanding of the game development process. Therefore, while we do not encourage the use of these assets, we understand their usefulness as vehicles for learning, and will not disqualify, or omit submissions that make use of them.



SrejonKhan/AnotherFileBrowser is licensed under the MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- ✓ Commercial use
- ✓ Modification
- ✓ Distribution
- ✓ Private use

1. **All submitted games must run in a Microsoft Windows operating system, or in an identified Internet browser.**
2. **All submitted games must utilize a keyboard and mouse based control system**
3. **All submitted games must function, first and foremost, as single-player games.**
4. **All submitted games must function/run independently of a need to download/install specific game development software, or additional software.**
5. **Submitted games should refrain from the use of store-bought or purchased assets**

- 1) As addressed in the technical requirements, our game was developed on & built for windows
- 2) The game functions with a keyboard & mouse based control system by default, even though it does have the additional capabilities of some mild controller support

-
- 3) Murus functions first & foremost as a single player game, even though it has the additional capabilities of importing other players levels
 - 4) No additional software is required to run Murus - everything is self contained.
 - 5) As previously addressed, our game features an open source, free for commercial use unity asset package which is necessary to access the file explorer to import other players levels (a feature that isn't even critical to the single player aspect of the game).

Judges

While Murus is designed for the average player, we didn't want to neglect the environment that Murus would be marked in for the competition - that being by a sole individual with varying levels of platforming experience & limited time. Because of this, we added tooltips to the first map informing the player of all the different difficulty & accessibility options the game has so that the Judge doesn't preemptively quit the game if they do not personally enjoy hardcore platformers. We really wanted the judge to experience the swath of content that we packed into the game, so making sure the judge actually was committed long enough was a major concern of ours, but we were hoping that the accessibility options make it so that no matter how inexperienced our randomly assigned judge is at video games, they can still fairly rate our game.

Bibliography

Stats on steam users operating systems:

<https://www.statista.com/statistics/265033/proportion-of-operating-systems-used-on-the-online-gaming-platform-steam/>

4 classes of game UI integration

https://www.gamasutra.com/view/feature/4286/game_ui_discoveries_what_players_php