

# OCR GCE A COMPUTER SCIENCE PROJECT H446-03

---

Name :                      Alexander Mills

Candidate Number :      <INSERT CANDIDATE NUMBER>

<Institution Name> :    <INSERT CENTRE NUMBER>

Title of Project :         <INSERT PROJECT TITLE>

# H446-03 – PROJECT CONTENTS

## Table of Contents

A. Analysis.....	6
Outline.....	6
Stakeholders.....	6
Game Research: Tetris .....	7
Game research: Hue.....	10
Game Research: World’s Hardest Game .....	13
Survey.....	14
Planning .....	14
Survey Response review .....	15
Proposed Feature List.....	18
Limitations and Scope.....	19
Why this Solution is Suited to a Computation Solution .....	20
Abstraction .....	20
Thinking Ahead .....	21
Thinking Procedurally .....	21
Thinking Logically.....	21
Thinking Concurrently .....	22
Hardware and Software Requirements.....	23
Success Criteria – designed for usability .....	23
Graphics - 1.....	23
Player design - a.....	23
Environment design - b.....	24
ENemy design - c.....	24

Objective design - d .....	25
User interface - 2 .....	25
Main menu - a .....	25
Pause menu - b .....	26
Options menus - c .....	26
GUI design - d .....	27
Sound - 3 .....	27
Sprite sounds - a .....	27
Level sounds - b .....	27
Background sounds - c .....	28
Level design - 4 .....	28
Maze layout - a .....	28
Maze population - b .....	29
Enemies - c .....	29
Checkpoints – d .....	29
Win criteria - e .....	30
Game mechanics - 4 .....	30
Player controller - a .....	30
Enemy controller - b .....	31
B. Design .....	32
User interface Design .....	32
User Interface Type .....	32
Layout .....	32
Navigation .....	36
Systems diagram .....	37
Overall program layout .....	38

Module Declarations .....	40
Module: Config .....	40
Config – Class: game_config .....	40
Config – testing .....	42
Module: Asset_loader .....	43
Asset_loader – Class: Img_loader .....	43
Asset_Loader – Class: Snd_Loader .....	44
Asset_loader – Class: Sprite_Sheet.....	44
Asset_Loader – Testing and Verification .....	45
Module: Main .....	46
Main - Class: Game .....	46
Main – testing .....	48
Module: maze_gen .....	49
Maze_GEN - Class: Maze .....	49
Maze_GEN - Algorithms.....	52
Maze_Gen – Testing: .....	55
Module: Sprites .....	57
Sprites – Class: Renderable_Sprite .....	57
sprites – Class: Player .....	57
Sprites – Class: Wall .....	60
Sprites – Class: Gateway .....	61
Sprites – Class: Block.....	61
Sprites – Class: Checkpoint .....	62
Sprites – Class: Key .....	63
Sprites – Class: Exit .....	63
Sprites – Class: Camera.....	64

Candidate Name: <INSERT NAME HERE>      Candidate Number: <NNNN>

Sprites – Class: Timer .....	64
Sprites – Testing.....	64
Module: Menu_Sprites .....	68
Menu_Sprites – Class: Text.....	68
Menu_Sprites – Class: Input_box .....	68
Menu_Sprites – Class: Toggle .....	69
Menu_Sprites – Class: Slider.....	70
Menu_Sprites – Class: Spinner .....	70
Menu_Sprites – Class: Button.....	71
Menu_Sprites – Testing .....	72
Module: Menu_System .....	74
Menu_System – Class: UI_Screen.....	74
Menu_System – Class: Main .....	75
Menu_System – Class: Start .....	76
Menu_System – Class: End .....	77
Menu_System – Class: Scoreboard.....	77
Menu_System – Class: Pause.....	78
Menu_System – Class: Options .....	79
Menu_System – Class: GFX_Options .....	80
Menu_System – Class: SND_Options.....	81
Menu_System – Class: Level.....	81
Menu_system - Testing .....	82
C. Developing the coded solution (“The development story”).....	84
D. Evaluation.....	84
Project Appendixes .....	85

## A. ANALYSIS

### OUTLINE

There is great academic pressure on students to perform to the best of their ability. To achieve this, students must study for longer, increasing stress levels and generating concern about whether time is being used effectively. There is a subsequent reduction in time spent on activities that don't tangibly benefit academic performance like gaming and other recreation. This has an adverse effect on mental health as it sets up a poor work life-balance and means there is no opportunity to de-stress, creating an unsustainable feedback loop which will hinder long term attainment.

To rectify this, I shall develop a game which 2d top-down tile game that heavily focusses on puzzle solving and systematic thinking. This will allow students to practice their problem solving and logical reasoning skills in a relaxed, enjoyable, and interactive game environment. This allows them to decompress, improving work-life balance due to a more sustainable method of practicing cognitive skills than studying. To successfully develop this solution, I will draw inspiration from other puzzle solving games such as Retro classics like Tetris(1984) and more modern examples like Portal(2007) and Hue(2016). This will allow me to evaluate existing solutions within this genre and which features are needed to ensure the game holds up to the stakeholders' expectations and meets their needs.

### STAKEHOLDERS

The target demographic of the game will be students in the age range of 15 to 18 who enjoy regular problem solving and logical thinking. This demographic covers a wide range of abilities; therefore, the game must have an array of tiered difficulty levels to ease beginners into the game while allowing advanced players to still enjoy it.

It is designed to be played after a study session to unwind, so the user will likely have a computer available, on which they play the game. This means the game doesn't need to be portable, so will be controlled by mouse and keyboard. As the game will be used to unwind and relax, it will have a simple, easy to understand control scheme; this will make it easier to learn and less taxing to use. To ensure that it is accessible to as many as possible, there will be very minimal text, having a symbol focused UI to overcome language barriers. The colour pallet of the game will use colours which are not too bright and have minimal blue; this will ensure it is pleasant on the eyes and not alarming, allowing the user to relax.

I have selected Benjamin Dodwell and Mate Fehevari to represent the target demographic. They are both 17 year old students who play videogames regularly. Their experience with similar games will allow them to give clear and well-judged feedback on my game, and how it compares to similar ones in the industry, allowing me to ensure my game meets the target demographics' needs effectively. They are also close contacts, so I will be able to regularly receive incremental feedback throughout the development process.

## GAME RESEARCH: TETRIS

Tetris is a 2d puzzle game where the player stacks blocks on a 10x20 grid. The square blocks come in groups of 4 called “tetrominos”, which can have many different shapes. They fall to the bottom of the board, and then stop falling, landing on top of any blocks that had previously fell. Should a full row be completed when the falling blocks are placed, this row is cleared, scoring the player some points. This makes for an engaging game where the player must organise a random stream of shapes into a compact pile at the bottom of the board, figuring out which shapes fit where to keep the board organised.

The game starts slowly, with the blocks falling slower. This allows inexperienced players to get used to the game mechanics . As more rows are cleared and more points are scored, the pieces fall faster, allowing the player less time to decide where to place the piece. This makes the game much more stressful and difficult for all but the most experienced players as even a small error can cause big problems, causing the blocks to pile up towards the top of the board, at which point the game is over.

To incentivise more advanced strategies, the game rewards clearing multiple lines at once, rewarding the user with more points. If they clear 4 lines in one go (the maximum possible), they score 8 times as many points as a single line. This leads to players risking building up larger piles so that they can clear more rows at once, earning more points more quickly.

### Main menu:

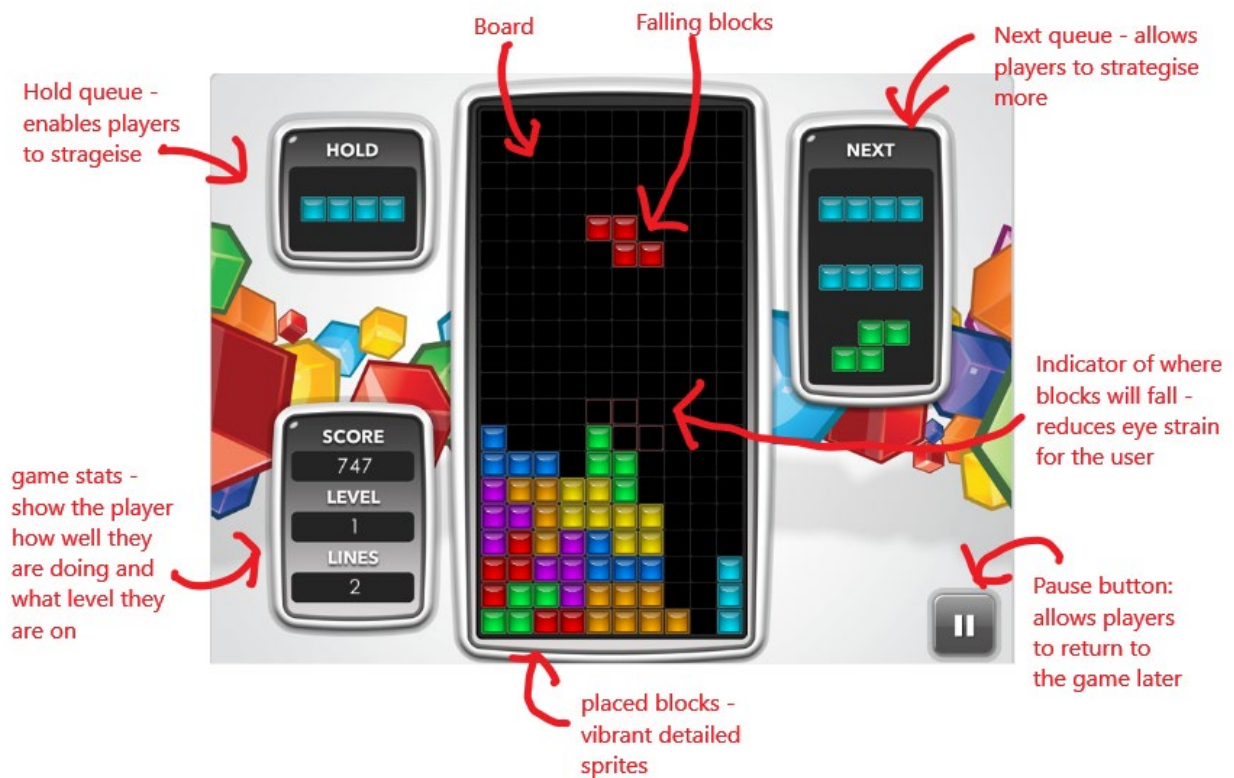


The game's main menu is the first thing that a potential player sees, therefore it is designed to introduce the players to the game, setting the colour scheme, theme, and branding. To help new players learn the game, there is a question mark button, which shows the controls, how to play the game and the language used to describe gameplay. My menu should contain all these features to make it usable and engaging.

The same UI "windows" are used in both the menu and the actual game. Hence the start menu has features that are blanked out, such as the "NEXT" and "HOLD" queues, which could be distracting or confusing for a new user. It also makes the UI over-crowded, so I will in my game I won't be re-using UI elements to reduce clutter.

### Gameplay:





The main game screen reuses the elements of the menu, so is familiar, though now all the elements are used. The bright colours on a dark background makes the game easier to look at, as well as distinguishing the individual sprites in the game and drawing the user's attention to the important features. The indicator of where the blocks will fall makes it easier for the user to see what the game will do next (where the block will land), reducing the chance of the user placing a block in the wrong place – this makes the game less annoying and therefore more enjoyable for the user; my game must also focus on this to meet the user's needs.

### Pause Menu:



The pause menu allows the user to stop the game and return to it later. This makes the game more convenient to play as the user can pick it up and put it down as they want. This will be less important in my game as each level will be played all in one session, though it will still be needed. The menu also offers a tutorial section for teaching inexperienced users and an options menu to allow the user to configure the game to their play style. My game should also have ample configurability to allow the user to have a comfortable gaming experience.

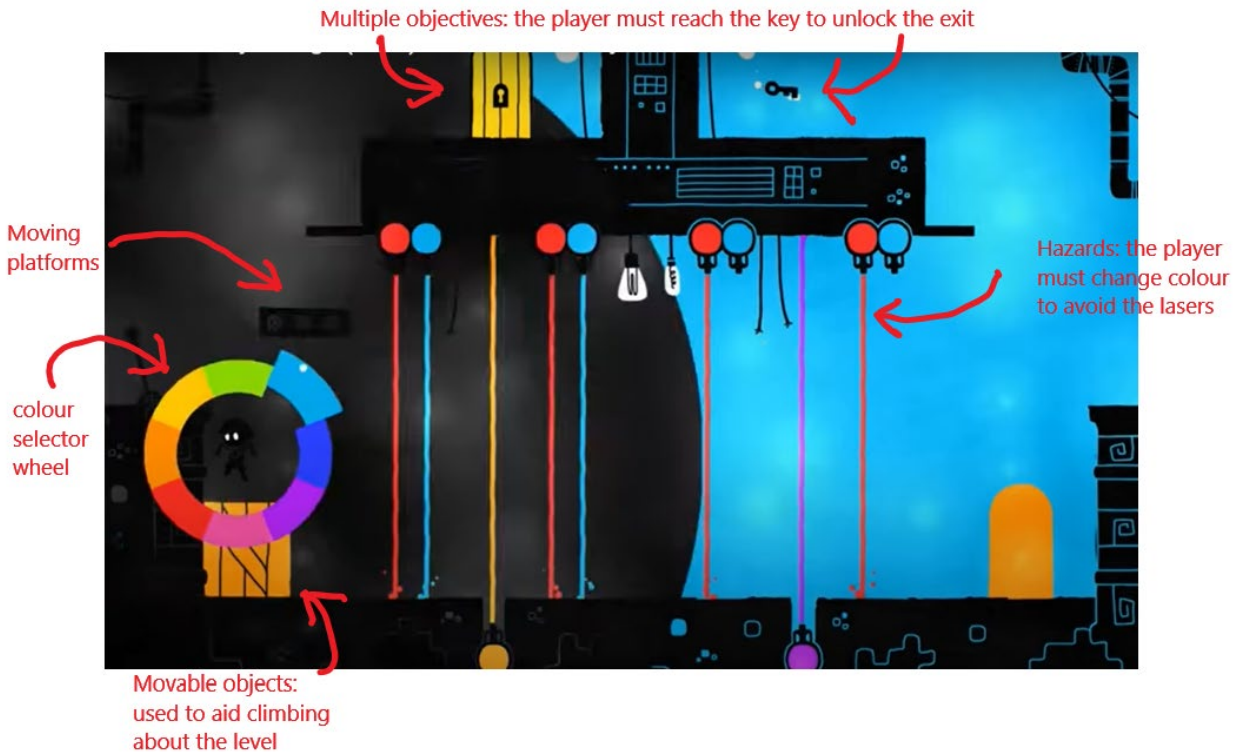
### GAME RESEARCH: HUE

Hue is a puzzle-based side scrolling adventure game with the goal of exploring the map and progressing the story line. The core game mechanic is that the player can change the colour of the background, making game objects of the same colour disappear, allowing the player to pass through them. With multiple colours, the puzzles become very intricate, requiring the player to carefully develop a strategy to deal with each new level, skilfully timing the switch between colours to avoid coloured hazards, move game objects around each other and traverse the coloured platforms to the exit. This mechanic makes for a more enjoyable and rewarding experience for the user as they must reason through how to make every move, and therefore I will implement a similar system for my game.

The game also makes strong use of a storyline developed by both narration and dialogue boxes from NPCs. The narration is triggered by the player finding letters, which are placed in longer, labyrinth style levels which are less challenging, allowing the player to absorb the story. The storyline adds depth and reason to the game, giving the player a reason to progress to the next area to further understand the

situation. This makes for a more immersive and engaging gaming experience, though a good story takes time to be written and will need narration, meaning this is out of the scope of my game.

### Typical level:



The colour scheme of the game is very focused around the 8 colours of the colour wheel, so they are a repeating theme throughout the whole game. The key game objects are in bright colours, which is both for the functionality and to highlight them to the player. The monochrome background complements the colours and is easy on the eyes, making it easier for the player to look at as it makes no use of bright or startling colours. I will make use of a similar colour scheme for my game, as it will make my game more relaxing to play, while still having visual interest.

The level design makes use of hazards, which the player must avoid by making use of the colour changing mechanic. These force the player to carefully time their inputs, making the game more challenging. The level also has multiple objectives: the player must acquire a key first before passing through the exit. This again facilitates more advanced puzzles. To make my puzzle game equally fun, I should incorporate all these level design queues. Each level has been manually designed, making them detailed, though I don't have time to design levels to this degree, so mine will have to be procedurally generated.

Pause Menu:



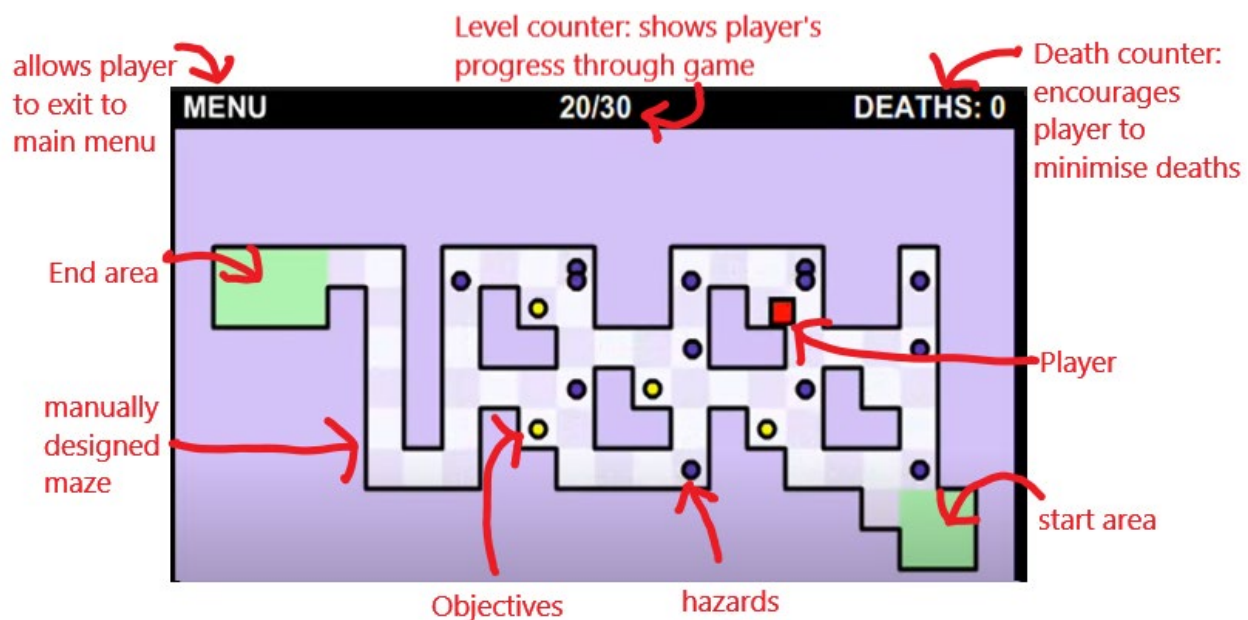
The pause menu allows the user to pause the game, allowing them to return to it later. It also provides some configuration menus for the user to tailor their experience to their needs. This includes a controls menu, where the user can learn the controls or configure them, a video menu where the user can configure the display resolution and full screen. It also has a colour-blind accessibility option, which is important as being able to distinguish colours is critical to the game, ensuring the game can be played by all potential stakeholders. The audio menu allows the user to control the volumes of different aspects of the game to their liking. These are all quality-of-life features, which enhance the rest of the user experience, and therefore will need to be a part of my game if it is to be enjoyable to play.

## GAME RESEARCH: WORLD'S HARDEST GAME

World's hardest game is a puzzle game where the player must navigate through mazes to the exit, collecting objectives before exiting. The mazes are 2d and are viewed from top down, so the player can immediately see all parts of the maze. This means that the player can heavily strategise how they are going to proceed through the level, but there is nothing to explore.

The core mechanic that makes the game much harder is the hazards moving about the maze. If the player touches one, they instantly die and return to the nearest checkpoint. They all follow pre-defined paths around the level but most move very quickly. The levels are designed such that all places in the maze baring a few have hazards moving over them, meaning the player must keep moving to stay alive, and as they are so close together, the player must perfectly time their inputs to move between them without hitting them, making the game very difficult. While this makes the game fun, it is also very stressful, something I want to avoid, so in my game there will be vastly fewer hazards and if they move, they will be much slower.

### Typical level:



The levels are all manually designed and have a standard structure: the checkpoints are green areas, the hazards are blue circles, objectives are yellow circles, and the player is a red square. This means the player knows exactly what they are doing each level, making the game intuitive to play. The maze has a checkerboard floor which clearly shows the game is tile based, allowing the player to judge the position and motion of the hazards. Manual layout makes for some clever and challenging level designs, though time must be invested to compose all the levels. As my game will need many levels, it will have to be procedural, but this will work well as it can generate a standardised colour scheme.



## SURVEY

## PLANNING

To gauge the needs of a larger group of potential stakeholders, I will use a survey to collect their opinion on how features of the game will be designed. This will allow me to make informed decisions about how the game should look and feel to play.

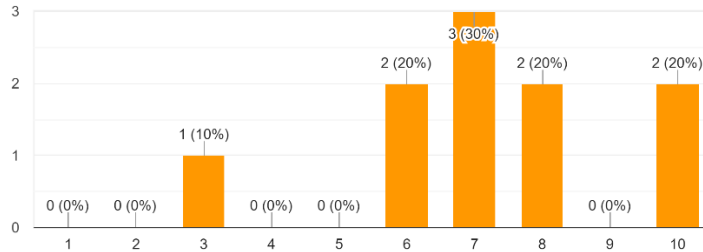
Question	Input type	Function
How important are graphics to make a puzzle game enjoyable?	Slider: 1 to 10 Comments box	Gauges how much work must be put into graphics to meet user needs
How much control over graphics is needed in the settings?	Multi choice: <ul style="list-style-type: none"> <li>No options</li> <li>basic options: resolutions, vsync, Fullscreen</li> <li>advanced: frame rate, rendering settings, toggleable visuals</li> <li>extensive: full colour scheme configurability, all rendering settings</li> </ul>	Allows me to develop a suitable graphics menu to make the game accessible for all users
How important are visual effects and animations to make a puzzle game enjoyable?	Slider 1 to 10 Comments box	Gauges how much work needs to be put into visual effects and animations
How important are Sound effects to make a puzzle game enjoyable?	Slider 1 to 10 Comments box	Gauges how much work needs to be put into the game's sound design
How much control over sound is needed in the settings?	Multi choice: <ul style="list-style-type: none"> <li>no options</li> <li>a slider for game volume, and a slider for music volume</li> <li>all game sounds have individual sliders</li> </ul>	Allows me to design suitable sound settings that will allow users to configure their game to their interests
How important is Background music to make a puzzle game enjoyable?	Slider 1 to 10 Comments box	Gauges how important background music is for the users to enjoy the game
How much time would you want to spend per level when playing a puzzle game?	Numerical input in minutes Comments box	Allows me to tune the level length so the game can be challenging for users but not enduring
How many times would you want to restart a level before completing it?	Numerical input Comments box	Allows me to adjust how many hazards there are in a level
Should the levels contain checkpoints?	Boolean Comments box	Determines if users want checkpoints or not, and thus determines if I will implement them
How should the game be titled?	Multi choice: <ul style="list-style-type: none"> <li>based on visual theme</li> <li>based on the style of puzzles</li> <li>based on a narrative</li> </ul>	Ensures that the title of the game conveys the theme and style of game to potential players well
Are there any other features which you would like to see in a puzzle game?	Comments box	Allows any other responses from the users, so they can input any other features they would like to see in the game

## SURVEY RESPONSE REVIEW

### GRAPHICS:

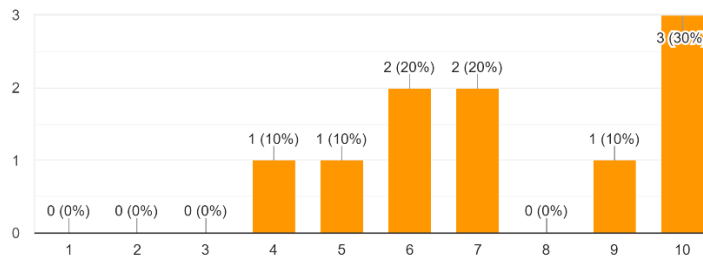
How important are Graphics to make a puzzle game enjoyable?

10 responses



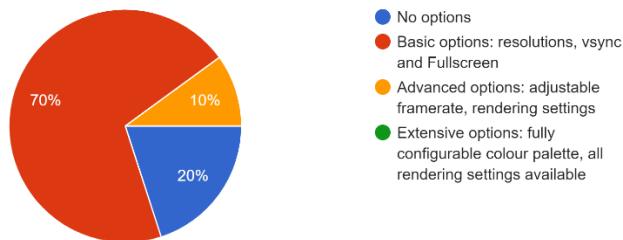
How important are Visual effects to make a puzzle game enjoyable?

10 responses



How much control over graphics and visuals is needed in the settings?

10 responses



Comments on Graphics and visual effects

2 responses

Visual effects should be used to make the game look polished and more appealing to players. Animations for keys or other sprites players can interact with would be a good addition to make the game more interactive, such as the animations used in Hue when a colour is obtained.

bru

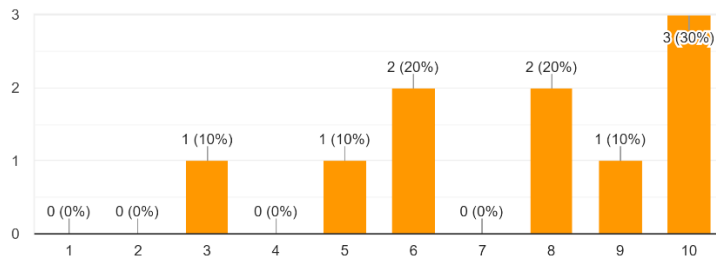
From the graphics part of the survey, it is evident that potential users prefer graphical fidelity over visual effects., though they are both very important. This means that I will have to spend more time on textures and sprites, ensuring they are high resolution with ample colour depth. I won't have time in this project to make them to the level required, so I will have to find some copyright free asset packs that work well together. These asset packs should also come with animations, allowing me to add some visual effects to the game quickly, though that isn't as important to the overall quality.

The users only need a simple settings menu which offers basic configuration for the game graphics, so I will implement a single graphics menu screen with configurable resolution and Fullscreen options.

## SOUND:

How important are Sound effects to make a puzzle game enjoyable?

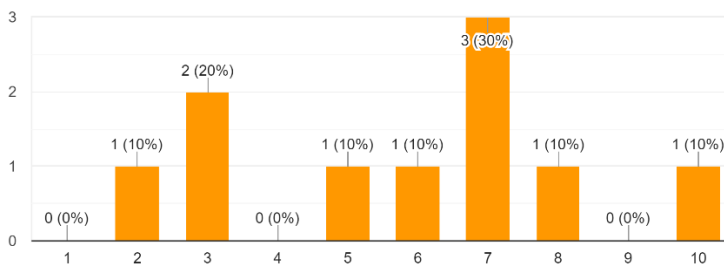
10 responses



By contrast, Sound is much less important for my game to meet user needs – it is still important, though less effort can be spent working on it. This means that I will spend minimal time designing sound effects so I will use copyright free ones or generate simple sounds from online tools. This will save time in the project so that I can spend more time on what is more important: the graphics and level design.

How important is Background Music to make a puzzle game enjoyable?

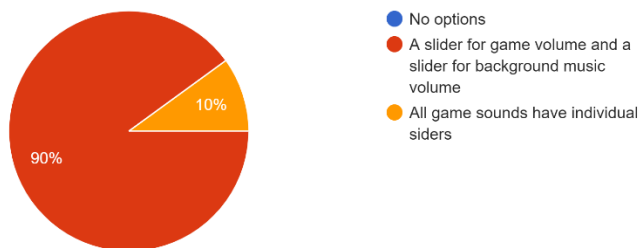
10 responses



The background music is again less important to the users, though it will strongly influence the feel of the game while playing it, so I will ensure to find some copyright free calming music to put for the background, as that will help the users relax while playing the game.

How much control over sound is needed in the settings?

10 responses



The sound menu will be very similar to the graphics menu: the users require no more than control over game and music volumes; this will fit easily into a single sound menu screen, which I will implement as part of the menu system

## Comments on Sound

2 responses

Calming music in a puzzle game is greatly appreciated

Sound effects should highlight key events in a game and should not be excessive. Major events in the game should include the collection of a sprite, the unlock of a door, etc.



## LEVEL DESIGN:

How much time would you want to spend per level in a puzzle game?

7 responses

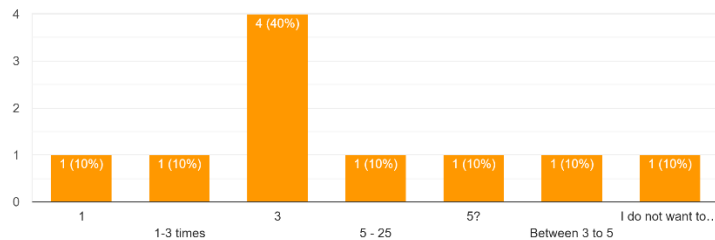
depending on the difficulty of the level, between 5 and 30
Depends on the difficulty. 1min-30mins
2-5 mins
2 - 10 min?
5-10 minutes
Up to 5 minutes (at least that's when it starts getting frustrating and I normally quit)
20 minutes

To ensure my levels are fun, engaging and challenging for all users, I need to identify key parameters that must be balanced to make the level accessible to all yet still difficult enough to be interesting.

None of the users want to be stuck on a single level for more than about 20 mins on average and 5 minutes looks like a good balance to ensure the levels remain enjoyable for all, and no one gets frustrated, though some are more patient and will happily play a level for up to half an hour. To meet all needs it would be good to make this variable, though this could take long to implement a system which creates balanced levels of vary sizes.

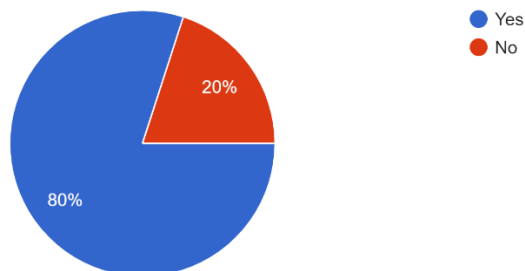
How many times do you want to restart a level before completing it?

10 responses



Should the levels contain checkpoints?

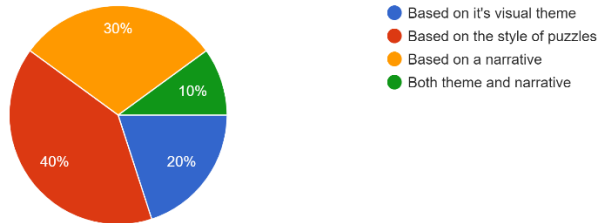
10 responses



The users want to have to try a level about 3 times before getting it, so they shouldn't be too heavy on hazards, though there should still be some to provide the correct level of challenge. The majority of users agree that checkpoints will make the level more playable, so those must be a feature to meet their needs.

## OTHER FEEDBACK:

How should the game be titled ?  
10 responses



game title suggestions  
4 responses

Colour Theory  
Saturation  
Hexa-Maze  
Fortnite

Are there any other features you want to see in the game?  
4 responses

Player customization  
Some form of dialogue + npcs to interact as your progress along each puzzle.  
Movable obstacles. Simple keybinds to control the state of the game, including pause menus and game instruction screens. Potentially animated start splash screen. Scoreboard to track progress, such as time to complete levels.  
Fortnite

The title of the game is the first thing a prospective user sees, so it must well represent the game. To accurately represent the game, it will be focused on it being a maze exploration game, as well as being linked to the visual theme of the game. That will entice potential players that are likely to enjoy the game.

Two of the features suggested (player customization and NPC driven story) are both not central to the gameplay, but make the game much more personal, giving each user the feeling of being emotionally connected to their character and their adventure making them more involved in the game.

These features may take a lot of time to implement, especially if they are to be done well, which likely puts them outside of the scope of what I can develop in this time frame.

A scoreboard is also a good idea to implement as that will allow timed competitive runs of the game, though this may be difficult to balance well with the procedural level generation.

## PROPOSED FEATURE LIST

Feature	Justification
Main menu which points to <ul style="list-style-type: none"> <li>Single player</li> <li>Settings</li> <li>Leaderboard</li> <li>EXIT</li> </ul>	Allows the user to quickly and easily navigate around all the games functionality
Procedurally generated mazes, populated with hazards and objectives automatically	Allows for infinite unique levels to keep the game new and enjoyable. Will take a lot less time to develop than manual levels

Ability to change player colour to navigate the maze	Makes the mazes more intricate and challenging to navigate
Ability to pick up and place down items to control elements of the maze	Makes the mazes more intricate and challenging to navigate
Enemies moving randomly around the maze	Makes the mazes harder to navigate as the player can't navigate about without considering where the hazards are going to go
Checkpoints in maze	Allow player to respawn at midway through solving a puzzle if they die
Settings menus for video and audio	Allows the user to configure the game as to make it optimally enjoyable for them
Menus must have simple, intuitive buttons and sliders	Enhances ease of use so users can focus on enjoying the game
Locally stored Scoreboard	Will allow the user to compete with themselves to beat their high score, making the game more challenging for those who want it
2d top-down camera perspective	Lends itself well to navigating and solving mazes
Limited field of view	Hides most of the maze from the user so they must explore it to discover the way out, making the game more challenging and in depth
Key game elements highlighted in functional colours	Makes the levels more intuitive as the user is automatically drawn to items and mechanics they need to use
Background elements must be relaxing, dark colours	Ensures the overall colour scheme of the game isn't too bright or startling, which is important to ensure the users can relax by playing the game
Ui during gameplay must be minimalistic	Keeps the screen free of clutter which will make it chaotic and stressful to look at.
simple animations for interacting with the maze and ui	Adds visual flare that makes the game feel more immersive, allowing the user to relax while playing the game
Simple sound effects for interacting with the maze and ui	Provides audible confirmation to the user about what they just did so they know it is important to beating the level
Relaxing, playful background music	Creates a calming, immersive atmosphere that ensures the user enjoys the game to full extent without distractions.

#### LIMITATIONS AND SCOPE

Limitations	How they would benefit the game	Reasons why they cannot be implemented.
Game can't be a 3d maze exploring puzzle game	A 3 <sup>rd</sup> dimension would allow the puzzles to be much more intricate, with many more hidden features and more alternate solutions	Im not familiar enough with 3d alternatives to pygame such as Ursina engine, which I don't have time to learn

There will be no narrative to the game	Narratives make games more enjoyable by telling an engaging, emotional story.	A well written and enjoyable story takes more time to come up with than I have for this project.
There won't be multiple level themes	More level themes would give the game more character, making it more immersive	Multiple level themes require more assets to be found or created, and then implemented, which I don't have time to do
There will be no player customisation	Player customisation would allow the user to feel more immersed in the game, making it more enjoyable	Configurable characters requires lots of assets for each part of the character, and a character config menu to be implemented, but I don't have time to implement this.
No local multiplayer	Would allow more difficult problems where the players must collaborate to solve the puzzle	Multiple player controllers would have to run together, as well the control scheme being more complex. It will also take more time to implement than I have available
No online multiplayer	Would allow players to solve puzzles with friends across larger geographic areas	Data would have to be sent across networks between clients and a host using socket, but I don't have time to learn how to implement this.

## WHY THIS SOLUTION IS SUITED TO A COMPUTATION SOLUTION

This game will have many complex features that must function correctly and interact with each other and the user seamlessly to produce an engaging, fun gaming experience. To do this I will employ computational methods

## ABSTRACTION

The player will walk around the maze, exploring the level, but navigating a real-life maze has a lot of complexities that are unnecessary and will make the game bulky, clunky, and difficult to play. Abstraction allows me to take away these annoying details while still retaining the original concept intact, but now much easier to interact with and use on a computer screen.

The gameplay will be built upon abstractions, for example, walking around a maze requires putting one foot in front of the other repeatedly to get around a 3d world, but controlling this directly will make the game hard to use and unintuitive, so instead the character controller will simply be the arrow keys which cause the player to move at a fixed rate in that direction on a 2d world. The inventory system will be heavily abstracted, just being a group of items, saving the user the trouble of trying to stuff many things inside a backpack to carry round.

The audio-visual design of the game will be abstracted, the textures being simpler than their real counterparts, with a less crowded colour palette and simpler shapes and less detail. The sound effects will be simpler, comprising of jingles rather than, for example the sound of actual keys being picked up. This serves to prevent viewing and

listening to the game from becoming overwhelming to the user, the simplicity making it much more relaxing to use.

Effective use of abstracted design is very important for my game to meet its users' needs as it allows the game to be intricate and engaging while not becoming overwhelming, laborious, and stressful, which is important while trying to relax and play a videogame.

---

## THINKING AHEAD

To ensure I meet the needs of the stakeholders as effectively as possible, I must carefully plan my game. This requires thinking ahead about how the game will be structured, planning out how it will be designed and how each part will function, reviewing how it should meet the requirements before being implemented.

The game will be planned extensively during the design phase, following a top-down design workflow, where the construction of each feature and how it will interact with all other features will be exactly detailed. This allows me to iteratively review the design to verify it still satisfies the success criteria all the way through development.

Without an effective plan, a project of this scale would quickly become incoherent, with each feature piling on top of the next, making the final solution a complex mess of inter dependent procedures, which would make the game impossible to effectively maintain or iterate on. This highlights how critical thinking ahead is to my game's success.

---

## THINKING PROCEDURALLY

During playing my game, many events will happen, such as receiving user input, loading assets, processing motion, rendering and animating sprites and displaying that to the screen. The events must be precisely timed to ensure the game behaves as I want it to, or it will become unpredictable.

To handle each sequence of events more easily, the game can be split into smaller, more manageable sub systems; this is Decomposition. There will be many smaller sub systems, such as:

- The game loop
- Asset loading systems
- Sprite rendering
- Maze generation
- Maze population
- Maze rendering
- Menu GUI

Each of these sub systems is a lot smaller and more specific than the game they will coalesce to form, meaning they are much simpler, each implementing only a few algorithms. Each one will be developed in isolation initially with a set of test programs to ensure they meet their functional requirements. This makes debugging much easier as the test programs will repeatably reproduce edge cases, allowing me to understand how my programs behave in tricky situations without struggling to reproduce those situations in the game itself.

---

## THINKING LOGICALLY

Candidate Name: <INSERT NAME HERE>      Candidate Number: <NNNN>

During gameplay, the user's decisions will impact what happens in the game next. This means that I will have to use logical thinking to ensure that certain gameplay paths are only unlocked under the correct conditions.

For example. The player will only be able to go through a door if they collect the correct key: This will require that upon approaching a door the code checks for if the corresponding key is in the player's inventory, and if it is, the door unlocks, removing its collider box, and if the key is not present, nothing happens

The player controller will require much logical thinking to design. The player must be able to move by taking in control inputs from the keyboard, where the player only move when a key is pressed, and it must decide which direction to move depending on which key it is. The player controller must also consider the environment, ensuring the player only walks on clear ground and never through walls, using conditions to check if there is a wall to the player's sides before moving, making sure to only move the player if there isn't a wall in that direction. The walls must also be checked to ensure that they are not the player's current colour, in which case what don't need to be collided with.

The main game loop will contain a litany of logic as it must consider what inputs are pressed and the game state to decide what to do with each input, such as checking what game state is currently active, then which parts of that game state have been unlocked, and then which parts of that state are currently being rendered on screen.

---

## THINKING CONCURRENTLY

There are many events that must happen all at the same time in the game; they must be processed concurrently. Concurrency is where the system switches very quickly between multiple processes to give the illusion that they are running in tandem: this will be used ubiquitously throughout my game.

The game loop must handle receiving inputs, updating each sprite, and drawing everything all at the same time as far as the user is concerned, but this can be achieved by checking the inputs, then updating each sprite one by one, then rendering each sprite one by one. This makes the game more playable and engaging than if each even happened one by one like in a text-based adventure game.

The audio system will also utilise concurrent processing as it will play dual channel audio from multiple sources at the same time, all while the game is also running. The background music will be playing from a file on loop in the background while events in game cause different sounds to be played and mixed over top of it.

## HARDWARE AND SOFTWARE REQUIREMENTS

<b>Processor: dual core x86 64bit @ 1GHz or better</b>	The game's code must be executed at a minimum rate to ensure it is fun to play
<b>Memory: 2 GB ddr3</b>	This will allow a minimal operating system build to run as well as the game, so long as it is the only thing running on the system
<b>Graphics: 256mb video memory, capable of rendering at 640x480</b>	The UI will depend on a minimum resolution to render properly and be readable, and this requires a minimum amount of video memory
<b>Storage: 500MB available space</b>	All the source code and assets use 500MB of free storage on the system
<b>Peripherals:</b> <ul style="list-style-type: none"> <li>• 40% or more qwerty keyboard</li> <li>• 2 button mouse or equivalent</li> </ul>	The gameplay requires wasdqe keys to play, which all qwerty keyboards bigger or equal to 40% will have. The UI menu system needs a mouse pointer to interact with, and a 2-button mouse will offer this.
<b>OS: 64 bit Microsoft Windows 10</b>	Windows is a modern and common operating system providing the required execution environment for the rest of the dependencies
<b>Python 3.10</b>	All my code will be written to be run by the python 3.10 interpreter, so to ensure all syntax is properly processed, python 3.10 is required
<b>Pygame 2.1.0</b>	My code will call pygame 2.1.0 functions, so to ensure that those functions run correctly, pygame 2.1.0 will be a requisite

## SUCCESS CRITERIA – DESIGNED FOR USABILITY

### GRAPHICS - 1

#### PLAYER DESIGN - A

Index	Requirement	Function	Source
1	Bright outstanding player colour scheme	Makes the player stand out from the rest of the game background	Tetris colour scheme
2	Player colour scheme reflects which of the 6 colours is currently selected	Allows user to tell what the current colour is to make puzzle solving easier	Hue game research
3	While walking, the player's feet animate	Makes the game much more immersive than the	User base survey

		player sliding across the ground	
4	When hurt, there is a visual indication they are hurt: they flash red	Tells the user that the character has been hurt, so they can be mindful of their lives	User base survey

### ENVIRONMENT DESIGN - B

Index	Requirement	Function	Source
1	Wall sprites are square	Makes wall sprites easy to procedurally tile, which the maze population engine needs	Proposed features: procedural maze generation
2	All types of wall sprites have the same texture	Indicates to the user that they cannot pass through this sprite	<ul style="list-style-type: none"> <li>Tetris game research: all blocks are the same texture</li> <li>Hardware limitations: reduces the number of textures loaded</li> </ul>
3	Wall sprites are of sufficient resolution to fit the theme	Ensures that the game has enjoyable, cohesive aesthetics	User base survey: good graphics are important
4	wall sprites have a dark colour	Makes the game more relaxing to look at	Hue game research: walls are darker colours
5	Gateway and block sprites have bright colours, which are randomly selected from 6 colours	Directs player attention to these walls, as they are interactive	Hue game research: objects critical to solving the puzzle are bright colours
6	Background environment colours are dark	Makes the game more relaxing to look at	Hue game research: environment around the game is dark.

### ENEMY DESIGN - C

Index	Requirement	Function	Source
1	Dangerous colour scheme: accents and highlights are red	Intuitively indicates this sprite is dangerous	User base survey: good graphics are important



2	Sprite is threatening: pointy angles, sharp shading	Intuitively indicates this sprite is dangerous	Hue: spikes have sharp angles to show that touching them is dangerous
3	Sprite clearly indicates what state it is in	Shows user if the enemy is attacking them or not	Proposed features: Intuitive gameplay

#### OBJECTIVE DESIGN - D

Index	Requirement	Function	Source
1	Enticing colour scheme: accents and highlights in gold	Draws the player towards them, so their importance is easy to understand	User base survey: graphics
2	Spaces where blocks can be placed to unlock new pathways are indicated	Indicates to the user that placing blocks here is needed to solve the level	Proposed features: changeable features of the maze
3	Blocks and the corresponding Gateways they open are colour coded	Allows user to pair together objectives while planning how to solve the level	Proposed features: changeable features of the maze

#### USER INTERFACE - 2

#### MAIN MENU - A

Index	Requirement	Function	Source
1	Background represents the game with an image of gameplay	Show the user what they are about to play, fits with graphical theme	Tetris game research
2	Start menu that opens a level	Allows the user to start playing a level	all researched games
3	When start button is pressed user is prompted to enter seed or allow a random seed	Allows a user to play the same level multiple times	Proposed feature list: scoreboard
4	Options button to open options menu	Allows user to configure game	User base study: settings
5	Scoreboard button that opens the locally stored scoreboard	Allows user to view previous high scores for each seed	World's Hardest Game research
6	Exit button that closes the game	Allows user to exit the game	All researched games

### PAUSE MENU - B

Index	Requirement	Function	Source
1	Can be opened by pressing escape	Minimises on screen UI	All games researched Proposed features: intuitive UI
2	Gameplay can be resumed by pressing resume button or ESC	Allows user to return to playing the game	All games researched Proposed features: intuitive UI
3	Button to access option menu	Allows user to change settings mid game	All games researched Proposed features: Settings menus
4	Button to restart level	Allows user to restart a level if they have made a mistake	Hue: pause menu's restart button is very useful
5	Exit to main menu button	Allows user to return to the main menu should they want to use it, eg to exit the game	All games researched

### OPTIONS MENUS - C

Index	Requirement	Function	Source
1	Buttons to open either graphics menu or sound menu	Separates different options to make menus easier to navigate	All games researched Proposed feature list: Settings menus
2	Graphics menu has buttons to toggle fullscreen and vsync	Allows user to tick which settings they want enabled	User base research: Graphics settings
3	Graphics menu has buttons to switch between available resolutions	Allows user to have the game at a good resolution for their screen	user base research: Graphics settings Hue game research: Graphics settings
4	Graphics menu has Apply button	User can change settings without the ui rescaling	All games researched Proposed features: Intuitive UI design
5	Sound menu has sliders for game sound and background music volume	Allows user to change the volumes of the game	User base research: Sound settings
6	Changes in sound menu take effect instantly	Allows user to gauge how loud it should be	User base research: Sound settings Hue game research: Sound settings menu

### GUI DESIGN - D

Index	Requirement	Function	Source
1	Buttons highlighted in bright colours	Allows user to clearly see and distinguish the menu functionalities	Hue game research: makes the menu easier to navigate
2	Buttons provide visual feedback when hovered over by changing texture	Shows user which button they are about to press	Hue game research: Menu system
3	Buttons provide visual feedback when pressed by darkening texture and moving	The user can see which buttons they are pressing	All games researched Proposed features: Intuitive UI
4	Buttons provide audible feedback when pressed	The user can hear which buttons they are pressing	All games researched Proposed features: Intuitive UI

### SOUND - 3

#### SPRITE SOUNDS - A

Index	Requirement	Function	Source
1	Walking sound	Indicates when the player is walking, making game more immersive	Hue game research Proposed features: Interacting with maze
2	Injury sound	Indicates when the player takes damage	Hue game research User survey: sound effects
3	Respawn sound	Indicates when the player has respawned	Proposed features: Interacting with maze

#### LEVEL SOUNDS - B

Index	Requirement	Function	Source
1	Block collection sound	Indicates to the user they have collected a block, so must be a positive sound	User base survey: sound effects Proposed features: Interacting with maze

2	Block placing sound	Indicates to the user they have placed a block	User base survey: sound effects Proposed features: Interacting with maze
3	Exit sound	Indicates to the user that the puzzle exit has been used, and they have finished the puzzle	User base survey: Sound effects Proposed features: Interacting with maze

### BACKGROUND SOUNDS - C

Index	Requirement	Function	Source
1	Relaxing Background music	Allows the user to relax while playing the game	User base survey: Comments on sound

### LEVEL DESIGN - 4

### MAZE LAYOUT - A

Index	Requirement	Function	Source
1	Maze has an entrance located on the edge	Acts as a starting place for the player to start from	Proposed features: maze generation
2	Maze has an exit located on the edge	Acts as a final objective for the player to navigate towards	Proposed features: maze generation
3	Maze is surrounded by walls on all sides	Stops the player from walking out of the maze, where the world isn't defined	Proposed features: maze generation
4	Internal walls are only placed on the inside of the maze	Ensures there are no useless walls as they would slow the game down	Proposed features: maze generation
5	There is a path from the entrance to the exit	Ensures the puzzle is solvable, otherwise the player will be frustrated	Proposed features: maze generation
6	All parts of the maze are connected	Makes sure enemies can navigate to the player, otherwise	Proposed features: maze generation

		some enemies will be useless, and will make the game slower unnecessarily	
7	Maze is well populated with walls	Ensures each level is challenging and not a strait forward corridor	Proposed features: maze generation

#### MAZE POPULATION - B

Index	Requirement	Function	Source
1	Maze is still solvable	Users need to have completable puzzles or they will get frustrated	Proposed feature list: maze generation User base survey: Desired level length
2	Blocks can be found before they must be used	Allows maze to be solvable	Proposed feature list: maze generation User base survey: Desired level length
3	Blocks are evenly distributed throughout the maze	Ensures the maze isn't too easy to solve	Proposed feature list: maze population User base survey: Desired level length
4	Enemies are evenly distributed throughout the maze	Stops the user from being overwhelmed by a group of enemies	Proposed feature list: maze population User base survey: Desired death count

#### ENEMIES - C

Index	Requirement	Function	Source
1	Hurts player on contact, dealing damage	Ensures the enemies are dangerous, making the player avoid them	Game research: World's Hardest Game
2	Pushes player back on contact	Makes the enemy attack more realistic	User base survey: visual effects
3	Has attack cooldown	Stops them from draining player health by attacking every frame	User base survey: Desired death count

#### CHECKPOINTS – D

Index	Requirement	Function	Source
1	When the player reaches a checkpoint, it is activated	Allows the checkpoint to detect when the player has reached it	User base research: Checkpoints Proposed feature list
2	When the user activates a checkpoint, other checkpoints are deactivated	Ensures only one checkpoint can be enabled at a time	User base research: Checkpoints Proposed feature list
3	When the player dies, they respawn at the nearest checkpoint	Allows user to restart the level from the last checkpoint when they die	User base research: Checkpoints Proposed feature list

#### WIN CRITERIA - E

Index	Requirement	Function	Source
1	When player reaches a key, they pick it up	Allows the player to achieve secondary objective to enable completing the level	Game Research: Hue, World's hardest game
2	When a player reaches an exit without all the keys, nothing happens	Ensures that the player must collect keys before trying to exit	Game Research: Hue, World's hardest game
3	When a player reaches an exit with all keys, they exit the level	Allows player to finish a level once they have all keys	Game Research: Hue, World's hardest game

#### GAME MECHANICS - 4

#### PLAYER CONTROLLER - A

Index	Requirement	Function	Source
1	When user presses arrow or wasd keys the player moves in that direction	Allows player to move around	All games researched Proposed features: intuitive controls
2	When a key is pressed, player accelerates, then has a constant velocity, then decelerates when key is released	Makes the player move more smoothly, making the game nicer to look at	All games researched
3	When a number key from 1 to 6 is pressed, the player colour is set to corresponding colour	Allows the user to control the colour of the player	Game research: Hue
4	When the player hits a wall, they stop moving in the axis of collision	Stops players moving through walls	All games researched

5	when q or e keys are pressed the player places one of their 2 collected blocks in front of them.	Allows the player to interact with the maze	game research: Tetris Proposed features: Intuitive controls
6	If there the user doesn't have any blocks in that slot when they try to place a block, no blocks are placed	Stops player placing blocks they don't have	game research: Tetris

#### ENEMY CONTROLLER - B

Index	Requirement	Function	Source
1	Defines a point in the maze to go towards	Gives the Enemy a place to go to	Game research: Hue Proposed features: Enemies
2	Path finds to get to the defined point.	Acts as a simple AI for the Enemy to follow	Proposed features: Enemies
3	When the player places a block, re-evaluates path	If a placed block obscures the path, the Enemy continues with a new path	Proposed features: Player can place blocks
4	Accelerates up to a constant speed when leaving an objective and decelerates when stopping at the next objective	Makes the Enemy's movement more fluid and predictable	Proposed features: Enemies

## B. DESIGN

### USER INTERFACE DESIGN

To make the game enjoyable and immersive to play, the user interface must be designed to be as easy to understand as possible, while still being capable enough to allow access to all the features. The user shouldn't be expected to have read instructions or done a tutorial on how to navigate around the menu system, therefore it must be designed to be instantly understandable, and this is best achieved by having it mirror user interfaces of many other similar games, then the user will start out familiar to the user interface.

#### USER INTERFACE TYPE

Most modern desktop games use the windows, icons, menus, and pointers (WIMPs) concepts for UI design. They are universal on the target platform; all computers with a mouse and keyboard have a menu system that makes use of these ideas. As well as its popularity and thus familiarity, I have also selected this kind of UI system as it is very visual, making excellent use of the resolution and colour depth available on modern displays to draw the user's attention towards critical features and indicators as well as directing them towards the intended navigation path using differently scaled and coloured interactive elements. By comparison, command line interfaces offer little interactivity and feedback to the user, so are only suitable for those who are experienced with not just the interface style, but also the particular application they are using, which makes them very unsuitable for games like mine.

As well as the user interface system being familiar to the user, each button's functionality and naming must be made as clear as possible so that a user knows what the button will do before they have pressed it. Otherwise, the UI will appear unpredictable and annoying, at which point it will have failed to meet the success criteria of the game being relaxing.

#### LAYOUT

The layout of each screen is also critical, as the user will search for certain buttons in certain locations on the screen before looking around for them, and the faster they can find the button they are looking for, the easier the UI will be for them to use. For example, while looking for the exit button they will check the buttons from the closest to the bottom and scan towards the top of the screen. The exact reverse is true for forward progressing buttons like start and resume; the user will check the top of the screen for those, so they should be placed there. Aesthetics are also critical for the menu layout as the game needs to be easy on the eyes. This is achieved through a symmetrical, well-organized layout where the elements are aligned in a grid, and thus they all have the same proportions, which should be dependent on a few simple ratios; this will reduce UI clutter prevent it from being a point of frustration for some users.



Main Menu Screen:

Colour Between The Lines

Start

Options

Scoreboard

Close

This is the screen that the game will start up to, and thus is the player's first impressions of the game. This screen will allow the user to navigate to all other parts of the game, so each button must be clear in what it will do and how it is a part of the flow of the gameplay. Thus, the buttons are vertically stacked in the order in which the user is likely to click them; this means that as the user's eyes scan down the list, they will likely see the more useful buttons first, which makes the UI much more usable.

Options Screen:

Options

Graphics

Sound

Exit

The options screen is what the user will use to navigate to other options screens. This screen will be accessed from multiple locations in the game, but then the exit button must cause the game to return to whichever screen led to the options screen in order to remain intuitive and predictable for the user. Thus, the screen switching will be implemented using a stack as this supports such functionality

Graphics Options Screen:

Graphics

< [resolution] >

Fullscreen ☐

Vsync ☐


Apply


Exit

The graphics menu is used to change graphics setting. The User can adjust these settings to maximize comfort when playing the game, such as ensuring the game fits in and makes good use of their computer screen. These setting must be saved across load cycles so that the user doesn't have to reconfigure them every time the game is loaded

Sound Options Screen:

### Sound

Game Volume: 

Music Volume: 

The sound menu follows a similar function to the graphics menu: it allows the user to configure their game to their comfort. Again, the settings chosen here must be maintained across load cycles so that the user doesn't have to re enter them. The music menu also has no apply button as the changes should take effect immediately to indicate the new volume to the player.

Scoreboard Screen:

### Scoreboard

n	Name	Time	width	height	seed
1	Name	Time	width	height	seed
2	Name	Time	width	height	seed
3	Name	Time	width	height	seed
4	Name	Time	width	height	seed
5	Name	Time	width	height	seed
6	Name	Time	width	height	seed
7	Name	Time	width	height	seed
8	Name	Time	width	height	seed
9	Name	Time	width	height	seed
10	Name	Time	width	height	seed

The Scoreboard screen saves player's scores along with their seeds; this allows the player to play the same exact maze multiple times should they want to improve their score. This also means that should they want to make it, this game can be competitive as users could try to speed-run the game, though this isn't in the intended use case

Pause Screen:

### Pause

The user sees this screen when they pause the game; this will be mid-level. This means that they can leave the game should they need to for any length of time and not impact the level timer. It also provides menus to allow for in game option changes or for the player to quit the level should they want to.

Candidate Name: <INSERT NAME HERE>      Candidate Number: <NNNN>

#### Start Screen

### Start Level

Maze Size:

Width:      Height:

X

Seed:

The start screen is what the user sees before they start playing the level – it allows them to set up the level to their interests. This means that they can change the size of the maze for longer or shorter puzzles. They can also specify a seed to replay a previous level should they want to. The start level button then allows them to actually start the level.

#### End Screen

### Level Complete!

X

Seed:

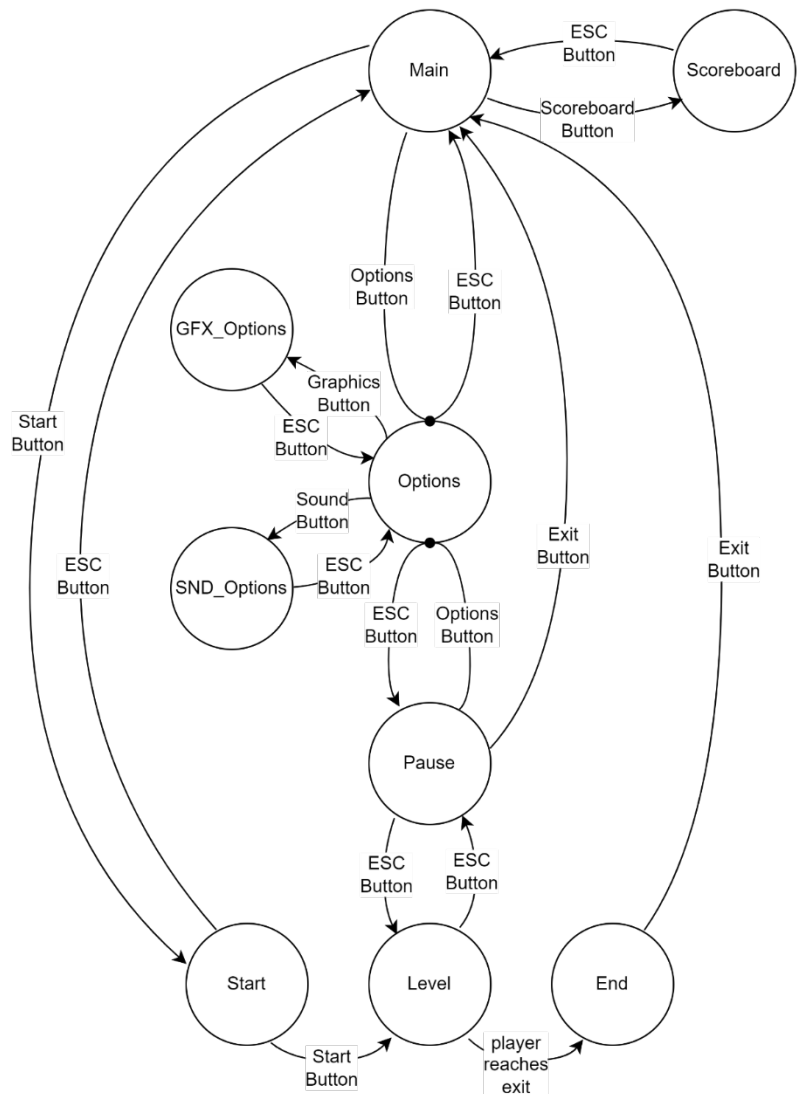
Time:

Put Your Name on the Scoreboard:

When the player finishes a level, they will be met with this screen; it congratulates them. It also tells them some statistics about the level they just completed. The user can then append their score to the scoreboard under their name. the return to main menu button then allows the user to return to the main menu, from which they can close the game, or start another level

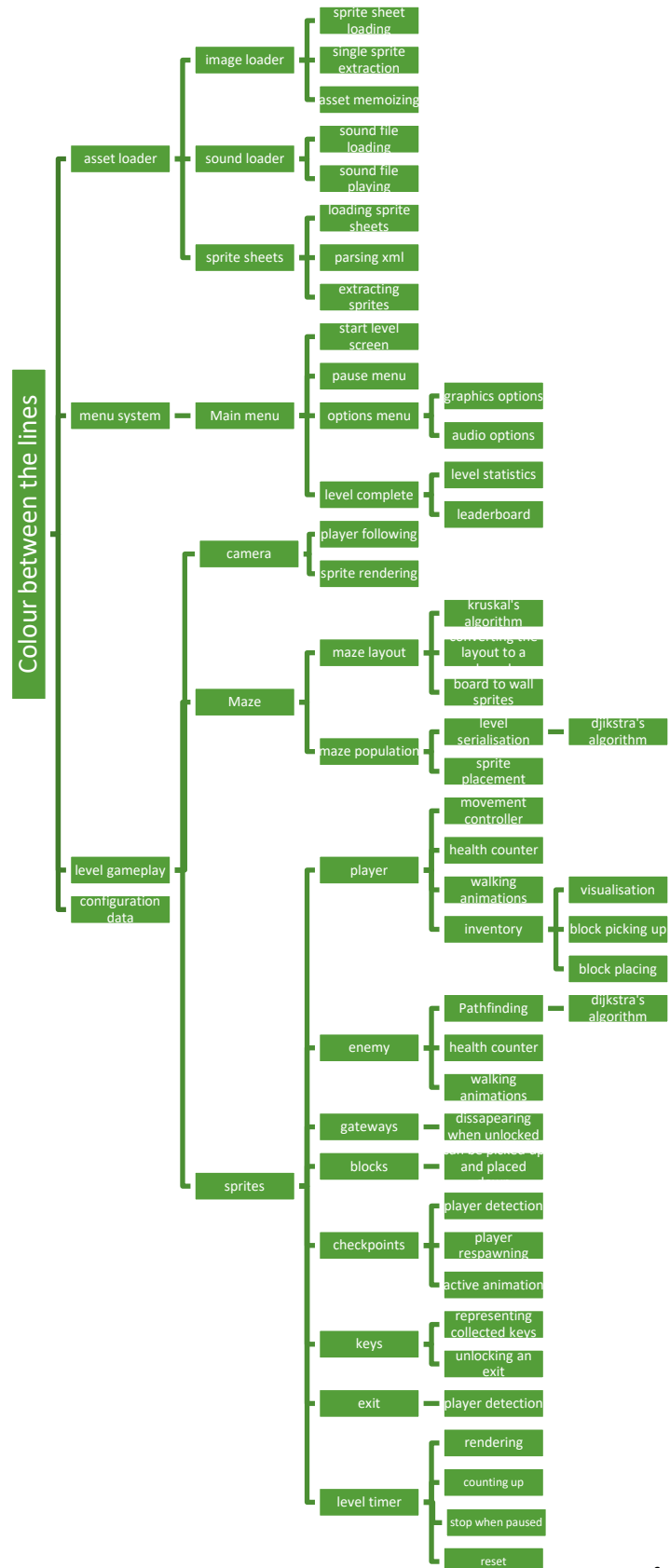
## NAVIGATION

The navigation between multiple menus is critical to the flow of playing the game, so the user must be capable of getting to any menu they need quickly, so that they can, for example, find a setting they want to change. Should this be complicated or convoluted, the user will get fed up with trying to find the setting and will just avoid configuring the game to what fits their situation best, which impairs the game's ability to be relaxing and enjoyable for all users. As such, I have taken into careful consideration what menus are needed to present the necessary information and how the user may flow between them. As well as providing enough flexibility for fast and intuitive navigation, the UI must also not be overwhelming, as this will also unsettle the user as they don't know where to go. As such, I have split out and categorized the menus in a hierarchical structure, such that the options menu leads to sub menus for different options, which means that it is faster and easier to find the exact setting wanted as well as reducing on screen clutter. To visualize and understand how the user will flow between the different game states, I have constructed a state relationship diagram for all the game states, and what events lead to the transition from one state to the next. This makes it clear how the user will progress through menus, and where they are restricted into following the flow of the game to ensure that it is easier to follow as there are less erroneous pathways.



## SYSTEMS DIAGRAM

The systems diagram shows that I have chosen a top-down approach to constructing my game. This means the game is composed of several independent modules which can be designed, developed, tested, and debugged individually. This decomposes the game down into manageable sub-units, each of which is small enough to easily comprehend while designing it. Each module will have a standardized interface, which allows other game modules to interact with it and make use of its services. Each module will also get its own test programs that make use of their interface so I can quickly and visually test each module's functionality, verifying it meets its success criteria before implementing it into the game. Once all modules have been independently developed, they will be integrated into the game, and then they will be holistically tested to determine if the game meets all its success criteria, identifying any shortcomings and patching them until the game meets all success criteria.



## OVERALL PROGRAM LAYOUT

For organization, my game will be split into modules, each of which will be placed in a separate file. With this system, objects and functions will be grouped by functionality. This will make development easier as code all modules are independent, so can be tested individually, which means that should a bug be discovered, there isn't much code that needs to be traced to understand how it got to an erroring state, making development faster. Code readability and therefore maintainability will also be improved with this layout.

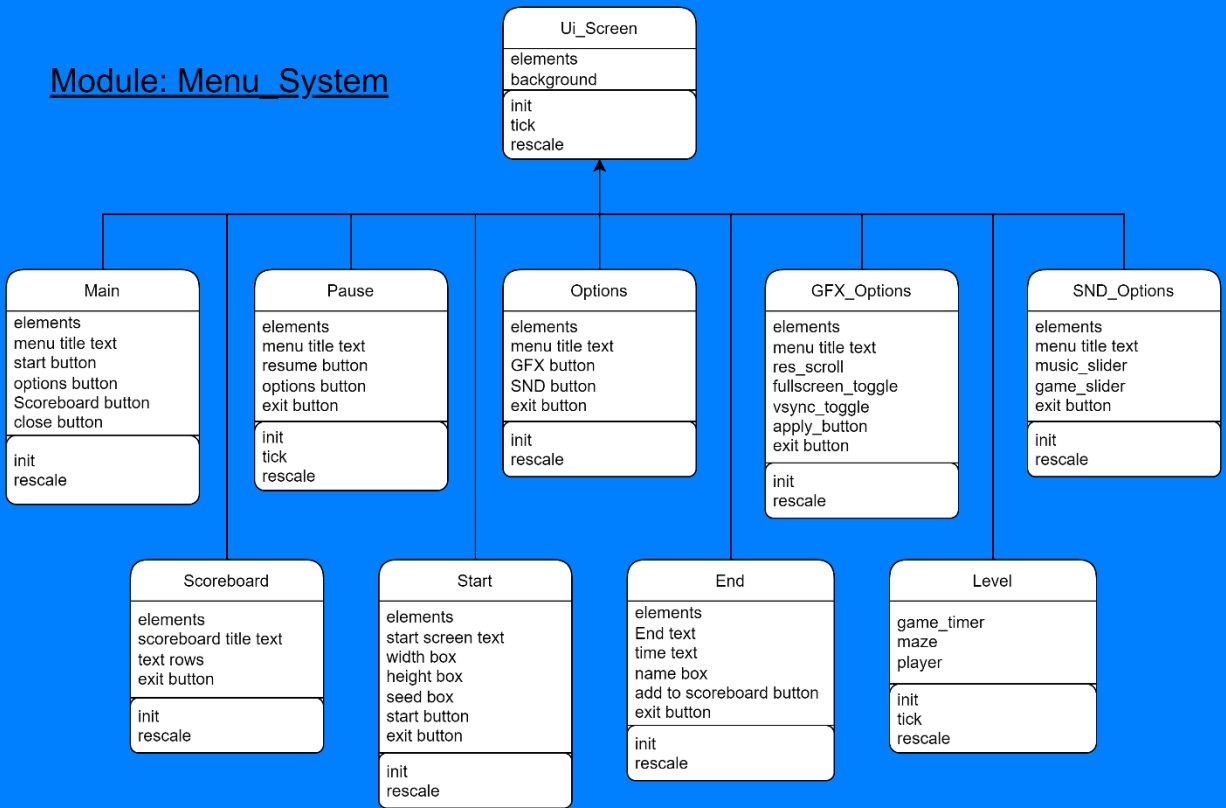
Each module will follow the same development methodology:

1. Declare Classes, identifying inputs and outputs.
2. Describe and explain the algorithms necessary for each method to achieve its function.
3. Define what the test program will do to verify each function, detailing what the inputs will be, and the corresponding outputs that are to be expected.

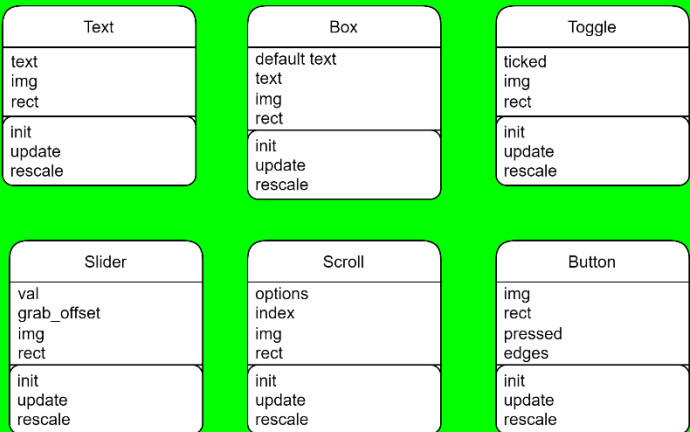
My game will make heavy use of object-oriented programming as this allows code and data to be collected and organized by overall function. As such, my design phase adheres to OOP based paradigms; each object is declared, defined, and assigned tests for each module one by one, as this reflects how it will actually be developed in section C of this document

A large high level game architecture diagram is shown on the next page, which details all the game's classes, their inheritance structure and their methods and attributes, with colour coding for readability.

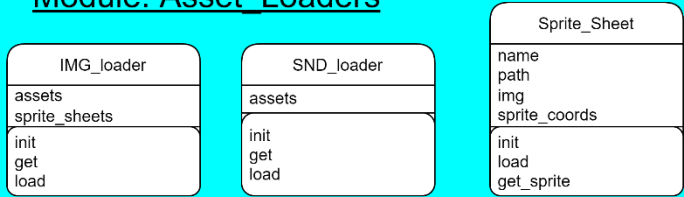
## Module: Menu\_System



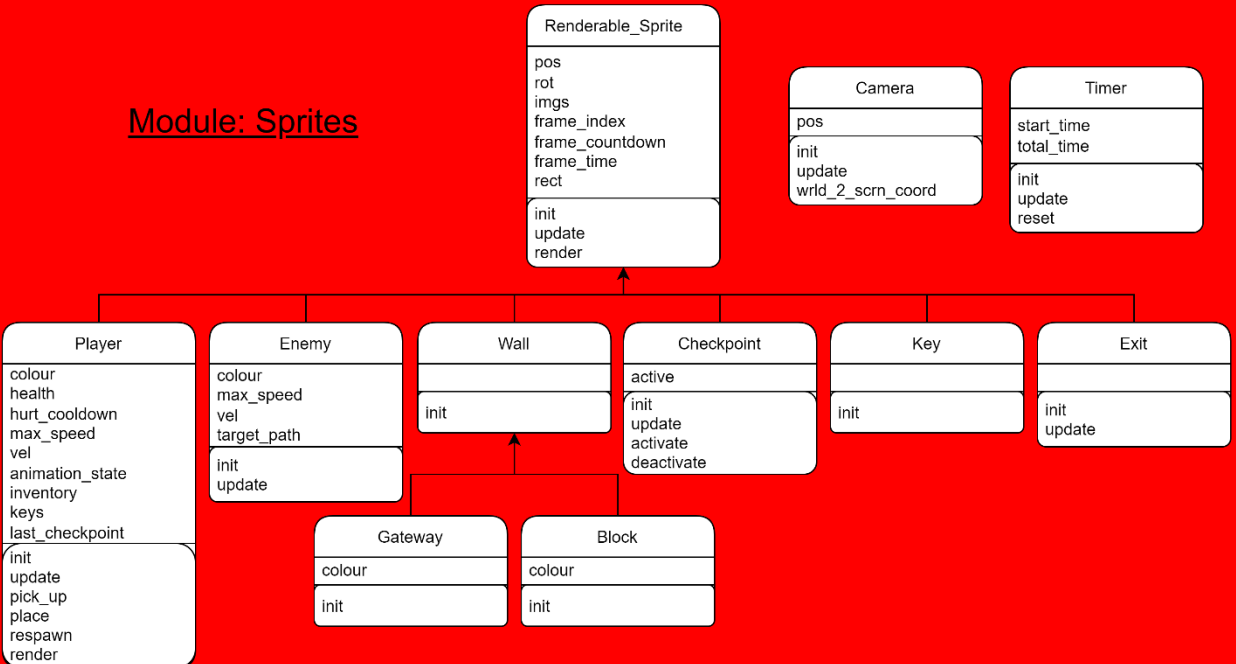
## Module: Menu\_Sprites



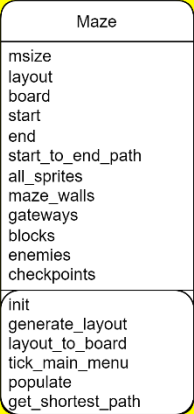
## Module: Asset\_Loaders



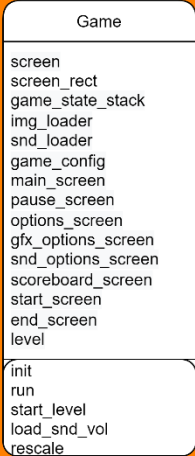
## Module: Sprites



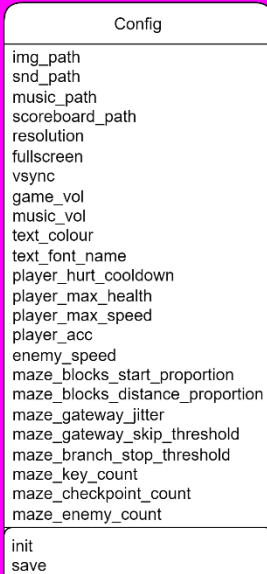
## Module: Maze\_Gen



## Module: Main



## Module: Config



## MODULE DECLARATIONS

### MODULE: CONFIG

- Acts as a single central collection of all game configuration data

### CONFIG – CLASS: GAME\_CONFIG

- Purpose: stores all configuration data for the game, such as user controllable settings and game balancing data. As all gameplay critical values are stored here, it is easy to make large changes to the game's dynamics and behavior in a single place
- Attributes:
  - img\_path – String
    - path of image assets folder
  - snd\_path – String
    - path of sound assets folder
  - music\_path – String
    - path of background music folder
  - scoreboard\_path – String
    - path of scoreboard csv file
  - resolution – array(int width, int height)
    - screen resolution
  - fullscreen – Boolean
  - vsync – Boolean
  - game\_vol – float
    - domain: 0 to 1 continuous
    - sets game volume
  - music\_vol – float
    - domain: 0 to 1 continuous
    - sets music volume
  - text\_colour – Tuple (int r int g in b)
    - each of r g and b are represent colour values
    - domain of r, g and b: 0 to 255
  - text\_font\_name – String
    - name of the font used for ui text
  - player\_hurt\_cooldown – int
    - domain: 0+
    - time in ms until between when the player can be hurt
  - player\_max\_health – int
    - domain: 0+
    - maximum number of heath points the player has



- player\_max\_speed – int
  - domain: 0+
- player\_acc – int
  - domain: 0+
  - how fast the player accelerates
- enemy\_speed – int
  - domain: 0+
  - how fast the enemy moves
- maze\_blocks\_start\_proportion – float
  - domain: 0 to 1 continuous
  - where along the path gateway population starts
- maze\_blocks\_distance\_proportion – float
  - domain: 0 to 1 continuous
  - how far along the path each gateway is from the next
- maze\_gateway\_jitter – int
  - domain: any integer
  - how much gateways randomly move back and forth from their planned position on the path
- maze\_gateway\_skip\_threshold – float
  - domain: 0 to 1
  - how often a gateway isn't placed immediately after it's block, but is saved for further in the maze
- maze\_branch\_stop\_threshold – float
  - domain: 0 to 1
  - probability that a branch keeps branching deeper into the maze
- maze\_key\_count – int
  - domain: 0+
  - number of keys to be generated in the maze
- maze\_checkpoint\_count – int
  - domain: 0+
  - number of checkpoints to be generated in the maze
- maze\_enemy\_count – int
  - domain: 0+
  - number of enemies to be generated in the maze
- Methods
  - VOID \_\_init\_\_ ()
    - initialise
  - VOID save ()
    - Self-modifying code: this method replaces it's class's attribute definitions with ones which have the values of it's attribute's current values; this implements a persistent python file where a class's attributes can be store values across multiple load cycles
    - Reads config.py from secondary storage
    - Replace all variable definitions with new ones in accordance with the current values stored in the attributes

- Save config.py to secondary storage

## CONFIG – TESTING

### Test environment:

- File structure:
  - Root
    - Config (file under test)
    - Host
      - Loads and instantiates config
      - Loads values from config
      - Saves values to config
      - Loads them again to check if they were saved correctly

### Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	Test loading string attributes	Host loads img_path and prints it	Valid	Outputs the img path saved there
2	Test loading int attributes	Host loads player_max_health And print it	Valid	Outputs the player_max_health value
3	Test loading bool attributes	Host loads fullscreen and prints it	Valid	Outputs the fullscreen value
4	Test loading float attributes	Host loads game_vol and prints it	Valid	Outputs the game_vol value
5	Test loading array attributes	Host loads resolution and prints it	Valid	Outputs the resolution stored there
6	Test storing string attributes	Host sets snd_path to a new value and calls save()	Valid	Config.py file now contains new value at snd_path definition
7	Test storing int attributes	Host sets player_max_speed to a new value and calls save()	Valid	Config.py file now contains new value at player_max_speed definition
8	Test storing bool attributes	Host sets vsync to a new value and calls save()	Valid	Config.py file now contains new value at vsync definition
9	Test storing float attributes	Host sets music_vol to a new value and calls save()	Valid	Config.py file now contains new value at music_vol definition

10	Test storing array attributes	Host sets text_colour to a new value and calls save()	Valid	Config.py file now contains new value at text_colour definition
11	Test reloading string attributes	Host reloads img_path and prints it	Valid	Outputs the img path saved there
12	Test reloading int attributes	Host reloads player_max_health And print it	Valid	Outputs the player_max_health value
13	Test reloading bool attributes	Host reloads fullscreen and prints it	Valid	Outputs the fullscreen value
14	Test reloading float attributes	Host reloads game_vol and prints it	Valid	Outputs the game_vol value
15	Test reloading array attributes	Host reloads resolution and prints it	Valid	Outputs the resolution stored there

## MODULE: ASSET\_LOADER

- stores classes responsible for loading and storing game assets. This allows asset loading to be handled completely separately from the rest of the game, making it easier and simpler to test and debug.

## ASSET\_LOADER – CLASS: IMG\_LOADER

- Purpose: loads, handles, and caches image assets for sprites to access.
- Attributes:
  - assets – dictionary (string img1\_name : Surface s1,string img2\_name : Surface s2)[img\_name]
    - Stores all images that have already been loaded in RAM so that they don't need to be loaded from secondary storage every time that they are needed.
    - A dictionary is used as it is a form of hash table, meaning all images very quick to access, having equal retrieval times. This will reduce delays in the game as it waits for assets to be accessed.
  - Sprite\_sheets – list (Sprite\_sheet sheet\_1,Sprite\_sheet sheet\_2)
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialises assets to an empty dictionary
    - Initialise all sprite sheets and store them in sprite\_sheets
  - Surface get(img\_name)

- If img\_name is in assets' keys, return assets[img\_name]
- Else if img\_name is in img folder, call load(img\_name) and return the surface it returns
- Else try to find the sprite in sprite sheets and return it if found
- if sprite can't be found, return a Surface of fixed size filled with purple
- Surface load(img\_name)
  - if a file of name img\_name exists in the config.img\_folder, load it from a file to a surface
  - set the colorkey of that surface so that it is transparent
  - return the surface

## ASSET\_LOADER – CLASS: SND\_LOADER

- Purpose: loads, handles, and caches sound assets for sprites to access
- Attributes:
  - assets – dictionary (string snd1\_name : Sound s1, string snd2\_name : Sound s2)[snd\_name]
    - Stores all sounds that have already been loaded in RAM so that they don't need to be loaded from secondary storage every time that they are needed.
    - A dictionary is used as it is a form of hash table, meaning all images very quick to access, having equal retrieval times. This will reduce delays in the game as it waits for assets to be accessed.
- Methods:
  - VOID \_\_init\_\_()
    - Initialises assets to an empty dictionary
  - Sound get(snd\_name)
    - If snd\_name is in asset's keys, return assets[snd\_name]
    - Else, call load(snd\_name) and return the Sound it returns
    - If sound can't be found, print to console
  - Surface load(snd\_name)
    - if a file of name snd\_name exists in the config.snd\_folder, load it from a file to a Sound object
    - return the Sound object

## ASSET\_LOADER – CLASS: SPRITE\_SHEET

- Purpose: stores a spritesheet image and provides functionality for interacting with it, allowing individual sprites to be extracted and retrieved
- Attributes:
  - name – String: contains name of spritesheet for retrieval
  - path – String: the path of the sprite sheet on the disk
  - img – Surface: stores the image of the spritesheet after it is loaded, providing quick access to it as it is stored in memory
  - sprite\_coords – dictionary[sprite\_name]: stores the rect of each sprite within the spritesheet, and as it is a hash table style data structure, all data is equally fast to access.
- Methods:
  - VOID \_\_init\_\_(string sheet\_name, string directory)

- Stores sheet name as attribute
- Define sheet\_path: directory / sheet\_name
- Load image from sheet\_path + .jpg and store to attribute
- Call load\_xml
- VOID load\_xml ():
  - Set sheet\_xml\_path to sheet\_path + .xml
  - Load xml sheet from path
  - Convert from xml to sprite\_coords dictionary
  - Store sprite\_coords dictionary as attribute
- Surface get\_sprite (string sprite\_name)
  - retrieve sprite\_rect from sprite\_coords
  - Store sprite\_rect subsurface of img to sprite\_img Surface
  - Return sprite\_img Surface

## ASSET\_LOADER – TESTING AND VERIFICATION

### Test environment:

- File structure:
  - Root
    - img
      - Img1.png
      - Spritesheet1.png
      - Spritesheet1.xml
    - snd
      - Sound1.wav
    - Asset\_Loader (file under test)
    - Host
      - Initializes pygame display system
      - Initialize a image loader and a sound loader
      - Call functions specified in test table and bit surfaces to screen and play sounds

### Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	basic sprite loading	Host calls get(img1.png) and blits it to screen	Valid	Img1.png appears on screen
2	loading sprites from a sprite sheet	Host calls get(sprite1)	Valid	Sprite 1 from Spritesheet1.png appears on screen
3	sprite caching	Host calls get(img1.png) again and blits it to	Valid	Img1.png appears on the screen again

		a different location on the screen		
4	Recovering from failing to load sprites	Host calls get(img2.png) and blits it to the screen	Invalid	There is a purple square placed on the screen and the game hasn't crashed
5	music loading	Host calls get(sound1.wav) and plays it	Valid	The sound can be heard playing
6	Recovering from failing to load music	Host calls get(sound2.wav) And plays it	invalid	No sound is heard and game doesn't crash

## MODULE: MAIN

- Contains the entry point for the game and main game loop

## MAIN - CLASS: GAME

- Purpose: hosts the main game loop and acts as a single data structure from which all data relating to the game is stored
- Attributes:
  - screen – display surface
  - game\_state\_stack (GSS) – stack (implemented as list)
    - stores which menu / screen the player has navigated to
    - top value is the current screen / menu the player sees
    - enables a single menu function to be used for accessing a menu from multiple places eg the options menu can be opened from the pause screen and the main game screen, and the stack will be used to keep track of which screen to return to
  - img\_loader – loaders.Img\_loader
    - is used to load all visual game assets
    - stores the visual assets so that they can be rendered multiple times while loading them only once
  - snd\_loader – loaders.Snd\_loader
    - is used to load all sound game assets
    - stores the sound assets so that they can be rendered multiple times while loading them only once
  - game\_config – config.Game\_Config
    - stores data about how the game is configured
    - persistently stores settings
    - stores all balancing variables in one place, so balancing the game will be easier
  - main\_screen – Menu\_System.Main

- The screen that the user first sees when they start up the game
  - pause\_screen – Menu\_System.Pause
    - the screen that appears when the user pauses the game
  - options\_screen – Menu\_System.Options
    - the screen used for accessing different options screens
  - gfx\_options\_screen - Menu\_System.GFX\_Options
    - the screen used for configuring graphical options
  - snd\_options\_screen - Menu\_System.SND\_Options
    - the screen used for configuring sound options
  - scoreboard\_screen – Menu\_System.Scoreboard
    - the screen that shows all previous level scores
  - start\_screen – Menu\_System.Start
    - the screen that allows the user to configure the level before starting it
  - end\_screen – Menu\_System.End
    - the screen that shows at the end of a level
  - level – Menu\_System.Level
    - the screen that plays the game
- Methods:
  - VOID \_\_init\_\_ ()
    - starts pygame execution environment
    - initializes video output screen
    - initializes asset loaders
    - initializes all screens except level
    - get music folder path from config.music\_path
    - start music playing
    - Pushes main\_menu to game state stack
  - VOID run ()
    - Runs main game loop.
      - Calculate dt (time since last loop)
      - Collect events from event queue
      - Checks for video rescale events if resolution is set to re-scalable
        - Stores new resolution to config
        - Call rescale ()
      - Calls the correct tick function for the current screen (screen atop game state stack), passing in the event list and dt
      - If game state stack is empty, the main loop exits
      - If config.vsync is true, stall such that the loop takes 16 ms to complete
  - VOID start\_level (array size, int seed)
    - Initialize new level screen with size and seed
    - Push that level to the top of the game state stack
  - VOID load\_snd\_vol ()
    - get volumes for game and music from config
    - call pygame function to set game and music sound volumes
  - VOID rescale ()

- Re initialize screen with config.resolution
- Call rescale methods of all screens

## MAIN – TESTING

### Test environment:

- File structure:
  - Root
    - Main.py (file under test)
    - Config
    - Asset\_loader
    - Menu\_System
      - Provides stripped down equivalents to UI screen objects, which print when any function is called and return generic default data. If their tick function is called, they print such, then wait a time before popping the top of the GSS
      - Main screen object takes command line inputs for which screen to open, then pushes that one onto GSS

### Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	main screen being pushed to GSS on startup	Start main file	normal	Console output saying it is on the main screen and prompting for input
2	start screen execution	Input: start	normal	Console output saying it is on start screen
3	level screen execution	Input: level	normal	Console output saying it is on level screen
4	end screen execution	Input: end	normal	Console output saying it is on end screen
5	scoreboard screen execution	Input: scoreboard	normal	Console output saying it is on scoreboard screen
6	pause screen execution	Input: pause	normal	Console output saying it is on pause screen
7	options screen execution	Input: options	normal	Console output saying it is on options screen
8	graphics options screen execution	Input: gfx	normal	Console output saying it is on graphics options screen
9	sound options screen execution	Input: snd	normal	Console output saying it is on sound options screen
10	Level screen initialisation	Input:level_init 30 50 12314515	normal	Console output saying level has been created with width 30, heigh 50 and seed 12314515 Console output saying it is on level screen



11	Level screen initialisation	Input:level_init 30 -50 12314515	invalid	Console output saying failed to create level: invalid input
12	Music player	Start main file	normal	Music starts playing when file is loaded
13	Music volume control	Music volume in config set to 0 then Input:snd_vol	Normal	Sound volume is set to mute
14	Music volume control	Music volume in config set to 1 then Input:snd_vol	Normal	Sound volume is set to full
15	Music volume control	Music volume in config set to -1 then Input:snd_vol	invalid	Sound volume is not set
16	Rescaling	Input: rescale config.resolution = 640 * 480	Normal	Display surface changes size Console output from all screens saying they rescaled

## MODULE: MAZE\_GEN

Purpose: Separates out code responsible for managing mazes, which are the game's levels

### MAZE\_GEN - CLASS: MAZE

- Purpose: manages all maze related data in the game, generating and storing the layout and wall sprites, and then populating the maze
- Attributes:
  - msize – tuple (int width, int height) [axis\_index]
    - stores how big the maze grid is
    - tuple allows it to be passed around efficiently
  - layout – array [y\_index][x\_index][side\_index]

(bool wall_below, bool wall_right)	(bool wall_below, bool wall_right)	(bool wall_below, bool wall_right)	...	(bool wall_below, bool wall_right)
(bool wall_below, bool wall_right)	(bool wall_below, bool wall_right)	(bool wall_below, bool wall_right)	...	(bool wall_below, bool wall_right)
:	:	:	...	:
(bool wall_below, bool wall_right)	(bool wall_below, bool wall_right)	(bool wall_below, bool wall_right)	...	(bool wall_below, bool wall_right)

- 3d Array structure provides random access, so all nodes are equally quick to work with
  - 3d Array allows multiple attributes to be stored for each node: different wall adjacencies
- board – array[y\_index][x\_index]

wall(corner)	wall(edge)	wall(corner)	...	wall(corner)
wall(edge)	checkpoint		...	wall(edge)
wall(corner)	gateway	wall(corner)	...	wall(corner)
:	:	:	...	:
wall(corner)	wall(edge)	wall(corner)	...	wall(corner)

- 2d array structure to store a grid representation of the maze, where each cell is a tile in the maze
  - 2d Array structure provides random access, so all nodes are equally quick to work with
- start – array (int x, int y)[axis\_index]
  - Start coordinate for the maze
- end – array (int x, int y)[axis\_index]
  - End coordinate for the maze
- start\_to\_end\_path – array ((node\_y, node\_x), (node\_y, node\_x), ... (node\_y, node\_x)) [dist\_from\_start]
  - Provides a linear abstraction of the maze
  - Used to place blocks and gateways in the maze
  - Allows the maze to remain solveable
- all\_sprites – SpriteGroup
  - store all sprites rendered during playing the level
  - The pygame.SpriteGroup data structure provides functionality for storing, updating and rendering sprites
- maze\_walls – SpriteGroup
  - Stores all sprites that will be used to create the walls
  - The pygame.SpriteGroup data structure provides functionality for storing sprites
- gateways – SpriteGroup
  - Stores all gateway sprites
  - The pygame.SpriteGroup data structure provides functionality for storing sprites
- blocks – SpriteGroup
  - Stores all block sprites
  - The pygame.SpriteGroup data structure provides functionality for storing sprites
- enemies – SpriteGroup
  - stores enemies in the maze
  - The pygame.SpriteGroup data structure provides functionality for storing sprites
- checkpoints – SpriteGroup
  - stores checkpoints in the maze
  - The pygame.SpriteGroup data structure provides functionality for storing sprites
- Keys – SpriteGroup
  - Stores keys in the maze
- Exit – sprites.Exit
  - Is the final objective for the level
- Methods:
  - VOID \_\_init\_\_ (array maze\_size, int seed)
    - Stores maze size and maze seed to attributes
    - Initializes generation RNG with seed

- Calls for maze layout to be generated
- Calls to convert layout to wall sprites
- Sets Start\_to\_end\_path using get\_shortest\_path
- Calls populate method to place blocks, gateways, and enemies in the maze
- VOID generate\_layout ()
  - uses kruskal's algorithm to generate the maze layout
  - stores viable maze layout into layout attribute
- VOID layout\_to\_board (funct wall\_generator)
  - Generates a 2d array of size msize\*2 + 1 by msize\*2 + 1
  - Initialise walls in set locations where both y index and x index are even to create corners
  - Initialise walls in set locations dictated by layout to create edges
  - Uses wall\_generator to generate walls in the correct position as dictated by the layout
  - Stores each wall in as it is generated attribute maze\_walls
- VOID populate()
  - Uses the population algorithms to populate the maze
- array get\_shortest\_path (tuple start\_pos, tuple end\_pos)
  - uses dijkstra's algorithm to find the shortest path from one point in the maze to another. Uses Dijkstra as it is a simpler algorithm to implement than A\* and is easier to balance so that it works for all possible mazes. As maze sizes won't be too big, the extra performance from A\* isn't needed

## MAZE\_GEN - ALGORITHMS

### Kruskal's Algorithm:

The purpose of Kruskal's algorithm is to generate a minimum spanning tree for any weighted graph. This can be thought about with a specific situation: Imagine a graph of all towns, where each town has an edge with all other town, who's weight is the distance between them. Kruskal's algorithm finds the road structure that should be built such that it connects all the towns into one network with the minimum length of road.

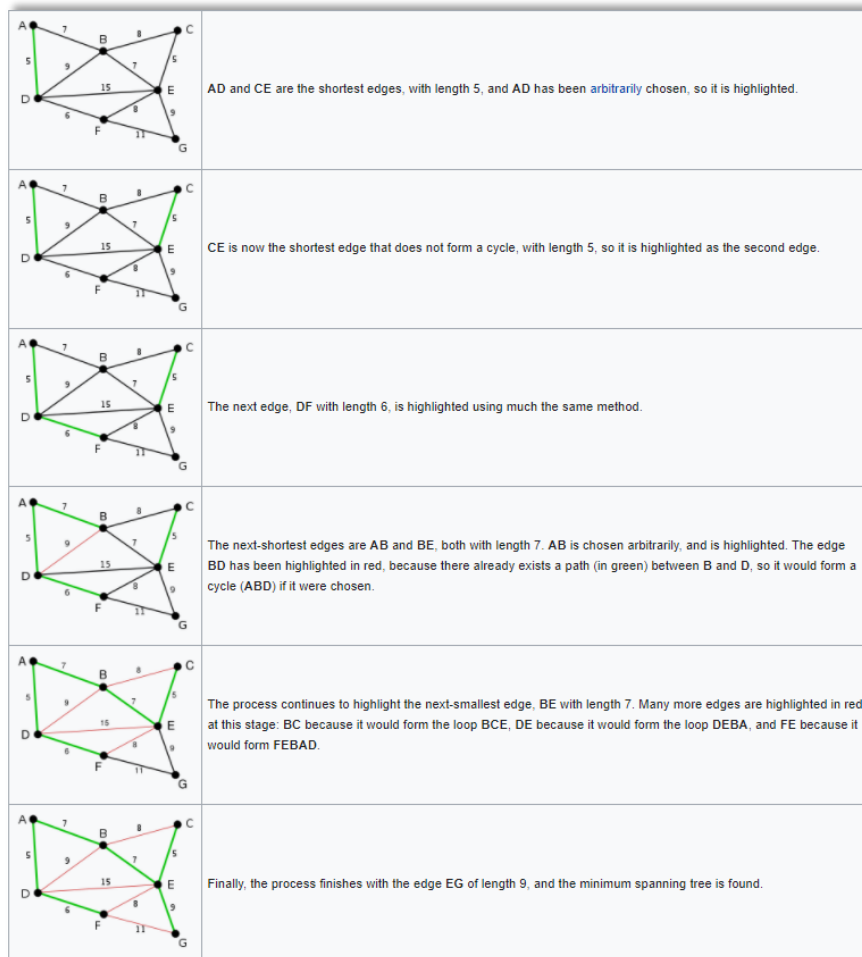


Figure 2: [https://en.wikipedia.org/wiki/Kruskal%27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)

then there must be a path between the start and end. The number of edges must be minimized as this makes the maze as difficult as possible, ensuring that there are no large open areas spread around the maze.

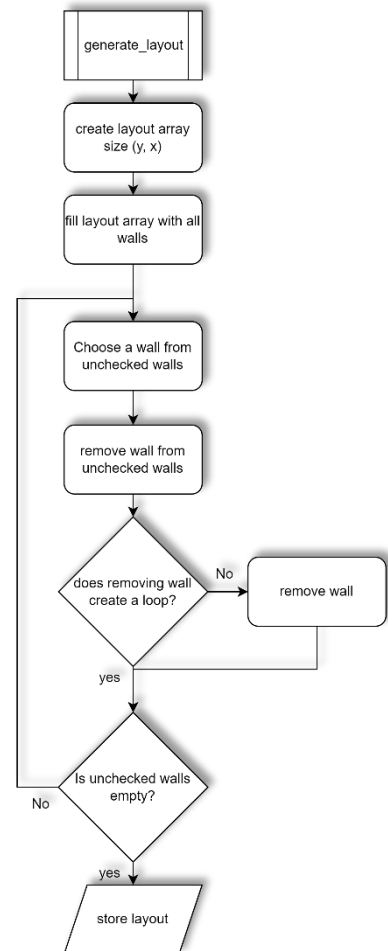


Figure 1: flow chart of Kruskal's algorithm

This can be used to generate a maze because it has 2 properties: it connects all nodes and minimizes the number of edges needed. All nodes must be interconnected as each node represents a room in the maze, and if all nodes are in some way connected to all other nodes,

## Dijkstra's Algorithm:

Dijkstra's algorithm is a classic algorithm for finding the shortest path between 2 nodes in a weighted graph. It maintains a list of nodes to search, and goes through them in order of closeness to the start, adding new nodes to the nodes to search as it discovers them. This has the effect of radially searching out from the start, until it finds then end, at which point it stops and retraces its steps to generate a list of nodes to path through to navigate from start to end. This means it can solve mazes, converting a complex interlinked graph structure into a simple sequence of nodes.

This will be utilised to position gateways and blocks in the correct order so that the levels are always solveable by iterating forwards through the path, adding the blocks first and then their corresponding gateways afterwards, and this will prevent a block from being placed behind it's own active block, which would make the maze impossible.

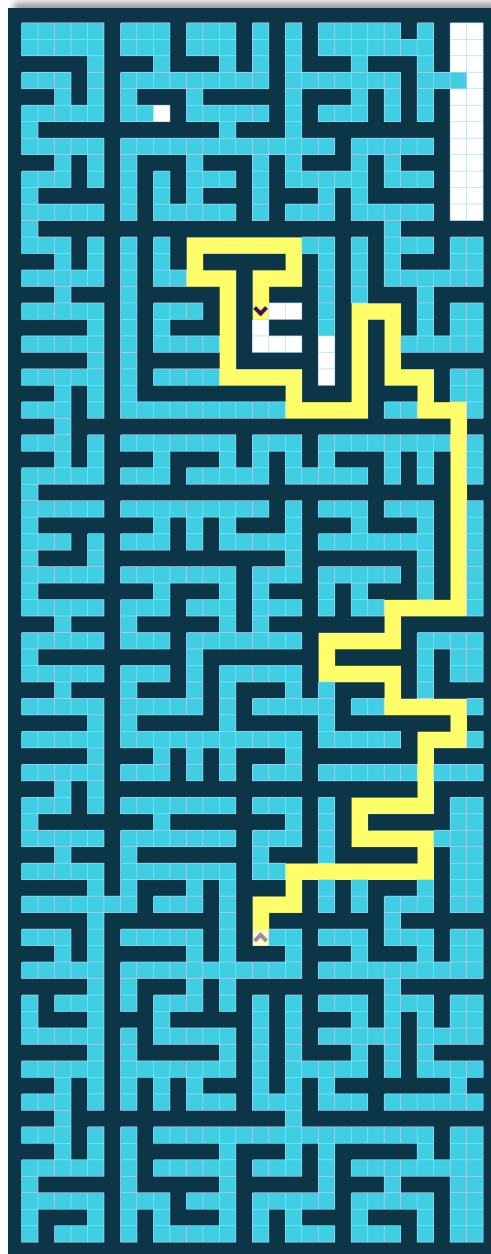


Figure 3: Visual demonstration of Dijkstra's path finding algorithm

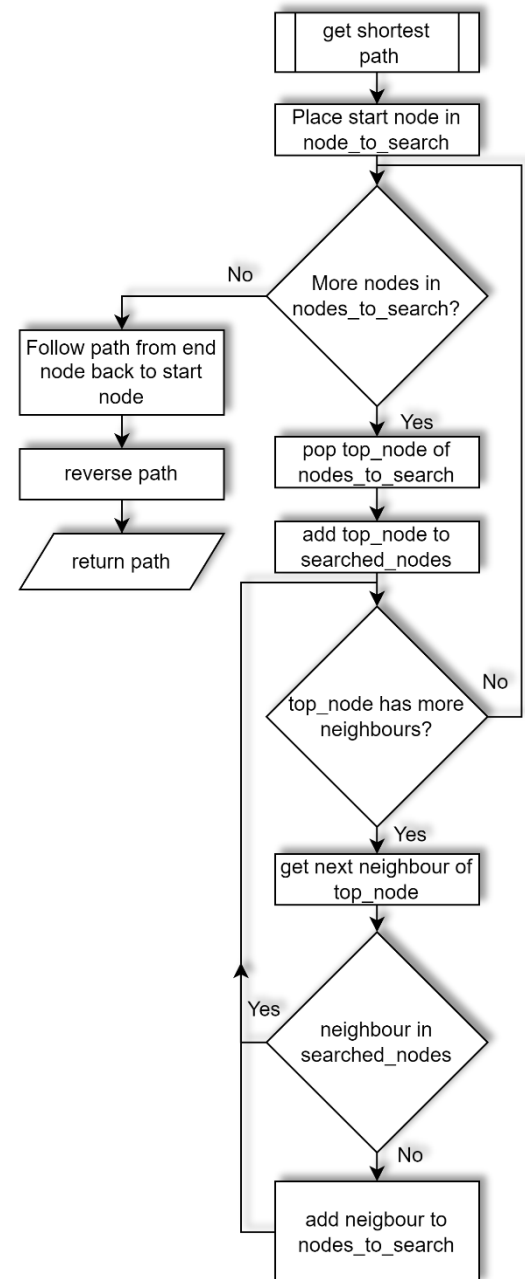


Figure 4: Flowchart of Dijkstra's path finding algorithm

## Maze Population algorithms:

These algorithms are responsible for populating the maze with the sprites the player will interact with during gameplay. To meet their success criteria, they must ensure that the maze remains solvable while making it harder to solve so that the game is more challenging and fun to play.

The first part of maze population is the placement of blocks and corresponding gateways. This algorithm comes first as it has the hardest success criteria to meet. This is because it must introduce features (such as gateways) which could easily make the maze unsolvable if not placed in the correct order. This algorithm has a very specific sequence to ensure that a block is

accessible before the gateway that it will unlock, otherwise it would be impossible to navigate the level. To make the game more interesting, it doesn't put the gateways in the same order that the blocks appear, meaning that they player would have to carry more blocks in their inventory than they are able to; this causes them to have to backtrack and remember their way around the maze.

The second algorithm, the Branch algorithm is not critical to level playability but makes the game much more enjoyable.

It decides where to put the blocks in relation to the path to the exit, moving them away and of into the maze, so the player has to go out into the maze and explore for them. It does this by repeatedly choosing a random neighbour and advancing to that neighbour until it has gone sufficiently far into the maze. To ensure it doesn't backtrack, it isn't allowed to advance to the node before the current one, and to ensure solvability, it isn't allowed to choose nodes that

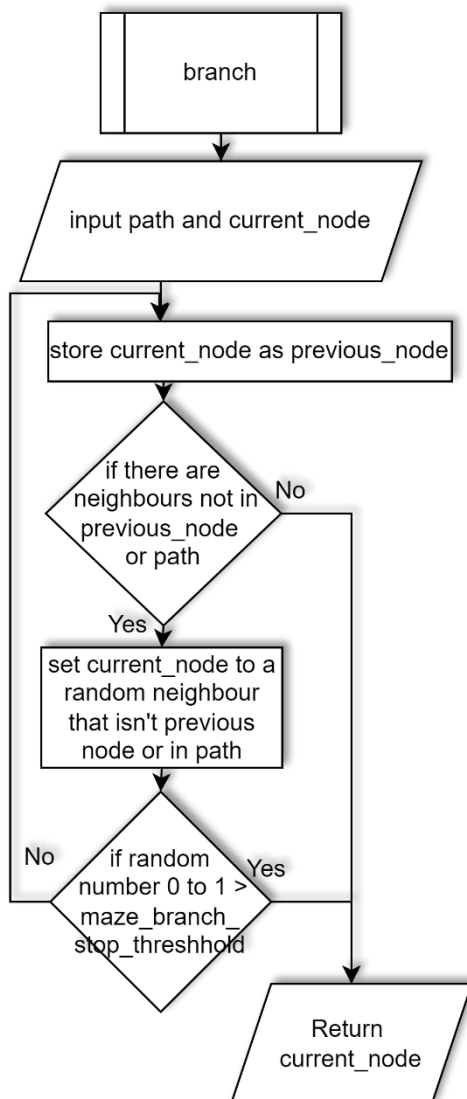


Figure 6: Flowchart for the Branch algorithm

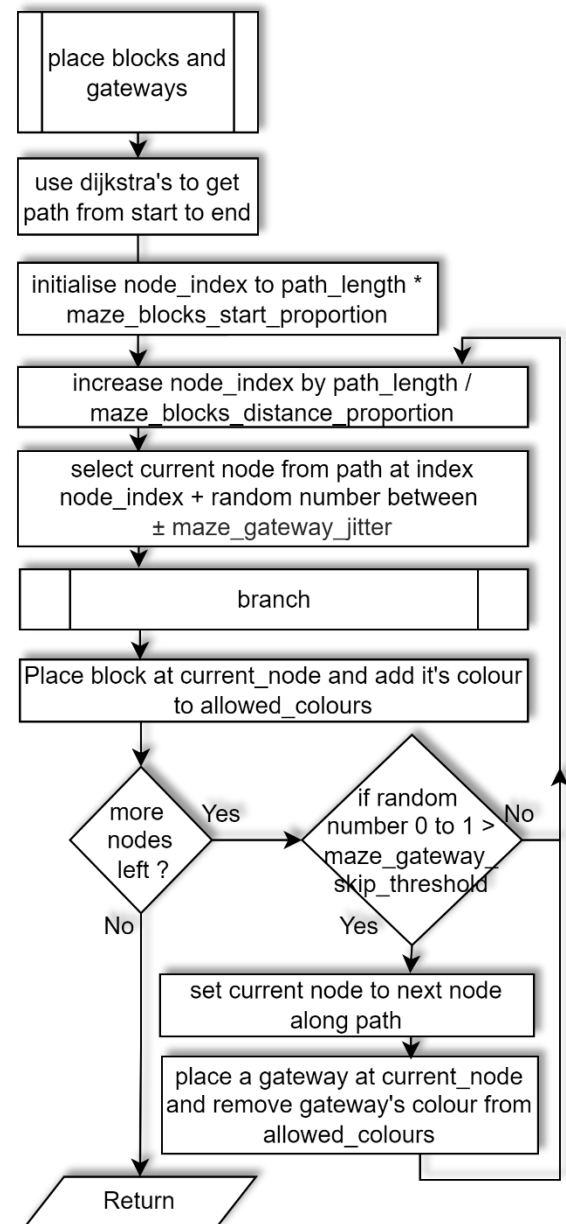
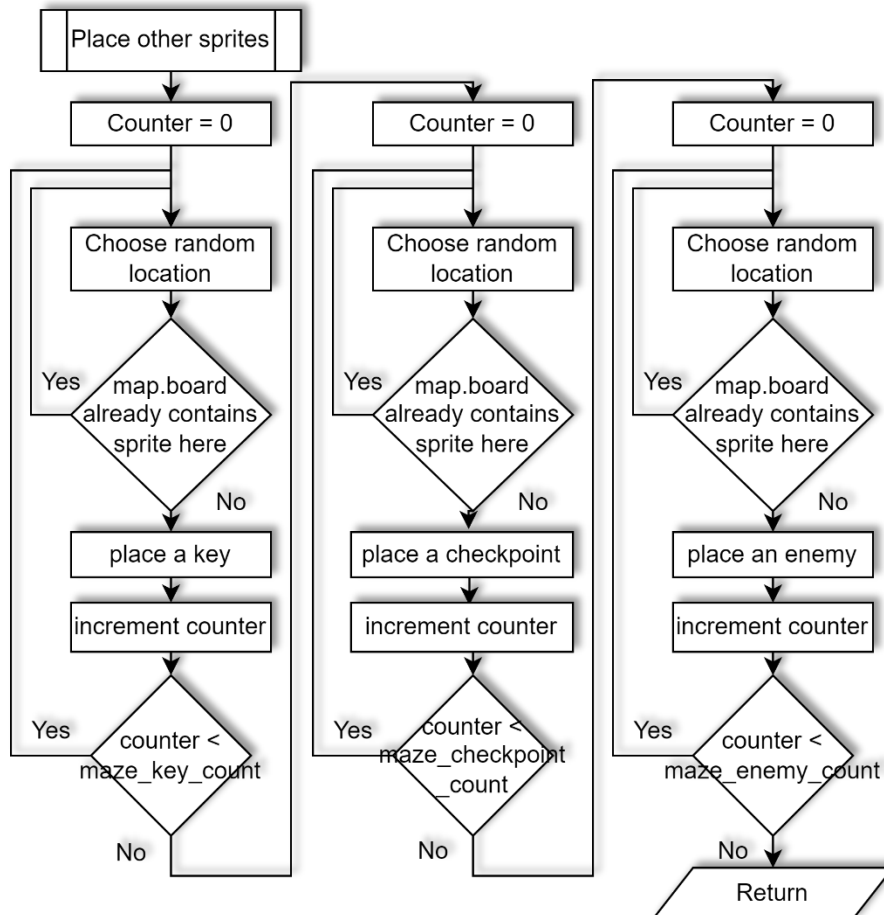


Figure 5: Flowchart for the block and gateway placement algorithm

are further up the path than the current one; this stops blocks from being placed behind their gateways. It is named so because it branches out from the start node until it selects an end node.



The third algorithm is responsible for placement of other sprites; this is a much less critical task than the other two as these sprites can be navigated around, and therefore can't block the maze and make it unsolvable. Subsequently, this algorithm is last to execute as the maze is populated. It does a simple task; it uses a count controlled loop to spawn a specified count of sprites at random coordinates, checking to ensure this random coordinate is within the maze and not inside a wall or other sprite. It does this process twice, once for spawning key and again for spawning

Figure 7: Flowchart for other maze population tasks

the enemies

## MAZE\_GEN – TESTING:

### Test environment:

- File structure:
  - Root
    - Config
    - Asset\_loader
    - Maze\_Gen (module under test)
    - Sprites

- Provides stripped down equivalents for sprites instantiated during maze generation
- Host:
  - Instantiates maze with width 20, height 10 and seed 12345
  - Prints maze layout to screen
  - Prints board to screen
  - Renders the dummy sprites on screen

**Test table:**

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	Maze initialization	New maze object instantiated	Valid	New maze object is created with the specified parameters without causing any errors
2	layout generation	New maze object instantiated and layout is printed to console	Valid	Maze layout is printed to screen, with no clearly visible inaccessible locations
3	board generation	New maze object instantiated and board is printed to console	Valid	Maze board is printed to screen, with no inaccessible locations and a solvable path from start to end
4	Seed functionality generating the same random maze multiple times	New maze object instantiated and board is printed to console	Valid	Maze board is Identical to previous maze board
5	wall sprite generation	New maze object instantiated	Valid	Maze's maze_walls group is filled with sprites
6	wall sprite positioning	New maze object instantiated and board is printed	Valid	When rendered on screen, maze wall sprites appear in the same pattern as the board
7	Gateway and block generation	New maze object instantiated	Valid	Number of sprites in "blocks" is greater than or equal to number of sprites in gateways
8	Gateway and block positioning	New maze object instantiated and board is printed	Valid	Blocks and gateways appear in same pattern on screen as they do on the board
9	Other sprite population	New sprite object is instantiated	Valid	Number of enemies, keys and checkpoints is equal to the numbers in config and one exit is present
10	Other sprite positioning	New maze object instantiated and board is printed	Valid	Pattern of enemies, checkpoints, keys, and exit is the same as on the board



## MODULE: SPRITES

- Classes for all visual sprites used in game

### SPRITES – CLASS: RENDERABLE\_SPRITE

- Purpose: provides supporting framework for sprites to render correctly from the maze coordinates onto the screen's pixel coordinates, making rendering on screen sprites much simpler by re-using the rotating and translating code
- Inherits from pygame.Sprite
- Attributes:
  - pos – array (int x, int y) [axis index]
    - stores where the sprite is located
    - array allows for fast and mutable access to the position
  - rot – int
    - stores rotation of this sprite
  - imgs – array (Surface frame\_0, Surface frame\_1, ... , Surface frame\_n)[frame\_index]
    - stores the animation frames for this sprite
  - frame\_index – int
    - stores which animation frame the sprite is on
  - frame\_countdown – int
    - stores how long until moving onto next animation frame
  - frame\_time – int
    - what frame\_countdown is initialised to; how long each frame lasts
  - rect – Rect
    - used to position where the sprite renders on screen
  -
- Methods:
  - VOID \_\_init\_\_ (tuple start\_pos, int start\_rot)
    - Initialise pos to start\_pos, or default to (0,0)
    - Initialise rot to start\_rot or default to 0
  - VOID update (dt)
    - Empty template method to ensure this sprite can be updated without the game crashing because it can't find an update function for a sprite
  - VOID render (dt)
    - Decrease frame countdown by dt
    - If dt < 0, advance frame\_index to index of next frame
    - Retrieve correct image from imgs
    - Rotates image according to rot
    - Set rect to position to render on the screen using camera position

### SPRITES – CLASS: PLAYER

- Purpose: the sprite the user controls to move around the maze
- Inherits from `sprites.renderable_sprite`
- Attributes:
  - `pos` – inherited from `renderable_sprite`
  - `rot` – inherited from `renderable_sprite`
  - `imgs` – inherited from `renderable_sprite`
  - `frame_index` – inherited from `renderable_sprite`
  - `frame_countdown` – inherited from `renderable_sprite`
  - `frame_time` – inherited from `renderable_sprite`
  - `colour` – tuple (int R, int G, int B)
    - stores the current colour of the player
  - `health` – int
    - stores how much health the player has
  - `hurt_cooldown` – int
    - ms until the player can be attacked again. Acts as a timer for the player to flash red when hurt
  - `vel` – array(int vx, int vy)
    - velocity of the player's movement
  - `max_speed` – int
    - the maximum speed the player can walk at
  - `acc` - int
    - how much the player's speed increases each frame (acceleration)
  - `animation_state` – string
    - indicates what animation the player is enacting
  - `inventory` – array (Block block\_Q, Block block\_E) [slot\_index]
    - stores which blocks are currently in the player's inventory
    - limited to a size of 2 to make the game a challenge
    - limited size will make it easier to render visually on screen to ensure minimal ui during gameplay
  - `keys` – int
    - how many keys the player has collected
    - doesn't need to actually store the keys as once they have been collected they are abstracted, as only the number of them collected is needed
  - `last_checkpoint` – `sprites.checkpoint`
    - stores the checkpoint the player will respawn at so that the code doesn't have to iterate over all checkpoints to find one that is active
- Methods:
  - `VOID __init__` (array start\_pos)
    - Call parent constructor
    - Set health to `config.player_max_health`
    - Set `hurt_cooldown` to `config.player_hurt_cooldown`
    - set `max_speed` to `config.player_max_speed`
    - set `acc` to `config.player_acc`
    - Load all animation frames into `imgs` using `img_loader`

- VOID update (dt)
  - if arrows or wasd is pressed, set animation\_state to “walking” and increase player vel by player\_acc up to max\_speed
  - if arrows or wasd are not pressed, set animation state to “standing” and decrease player vel by player\_acc down to 0
  - if q(slot\_index=0) or e (slot\_index=1) is pressed, call pickup or place functions depending on if the player already has a block in that slot
  - if colliding with a key, delete that sprite and add one to the number of keys
  - updates player rot based on current moving direction
  - call collide ()
  - If health = 0, call respawn
- VOID pick\_up (int slot\_index)
  - Determine distance to each block
  - if the board coordinate in front of the player contains a block and inventory[slot\_index] isn't full:
    - store this block to inventory[slot\_index]
    - Play block collection sound
    - Remove block from maze.blocks and game.all\_sprites
    - Store block in inventory[slot\_index]
- VOID place (int slot\_index)
  - Find board coordinate of the tile in front of player
  - if the board coordinate is empty in maze board store the wall in inventory[slot\_index] there,
    - Remove wall from inventory[slot\_index]
    - Add wall back to all\_sprites and maze.blocks
    - Update block's pos to new position
  - Else If the board coordinate where it was going to place in maze board is an gateway, delete both the block and the gateway.
- VOID respawn ()
  - Play respawn sound
  - Set health back to config.max\_player\_health
  - If last\_checkpoint isn't Null Set pos to last\_checkpoint's position
  - Else set pos to maze.start
- VOID collide ()
  - checks for collisions with walls of all colours other than the current one and prevents player walking through them by setting velocity in that axis to zero and moving the player until their outside of the wall they are colliding
- VOID render (dt)
  - Decrease frame countdown by dt
  - If frame\_countdown < 0, advance frame\_index to index of next frame
  - Retrieve correct image from imgs
  - Blit blocks in inventory to image – allows UI-less inventory display
  - Rotates image according to rot
  - Set rect to position to render on the screen using camera position

### Sprites – Class: Enemy

- Purpose: makes the game more challenging by randomly patrolling the maze, hurting the player if they get in the way
- Inherits from `sprites.renderable_sprite`
- Attributes:
  - `pos` – inherited from `renderable_sprite`
  - `rot` – inherited from `renderable_sprite`
  - `imgs` – inherited from `renderable_sprite`
  - `frame_index` – inherited from `renderable_sprite`
  - `frame_countdown` – inherited from `renderable_sprite`
  - `frame_time` – inherited from `renderable_sprite`
  - `colour` – tuple (int R, int G, int B)
    - stores the colour of the enemy
  - `max_speed` – int
    - how fast this enemy walks
  - `Vel` – array (int vx, int vy)
    - Enemy velocity
  - `target_path` – queue ((int x\_1, int y\_1), (int x\_2, int y\_2), ... , (int x\_n, int y\_n) ) [target\_index]
    - stores targets for the enemy to navigate to, where it navigates to the first one
- Methods:
  - `VOID __init__ (array start_pos)`
    - `pos` – inherited from `renderable_sprite`
    - `rot` – inherited from `renderable_sprite`
    - `imgs` – inherited from `renderable_sprite`
    - Load all animation frames into `imgs` using `img_loader`
  - `VOID update (dt)`
    - Move towards current target at end of `target_path_queue`
    - If within a certain radius of current target, remove current target from `target_path_queue`
    - If target path is empty, generates a random location to target then `maze.get_shortest_path` to refill `target_path`
    - updates `rot` based on current moving direction
    - check if colliding with the player; if so, decrease the player's health value and play injury sound
  - `VOID render (dt)` – inherited from `renderable_sprite`

### SPRITES – CLASS: WALL

- Purpose: acts as the walls that make up the maze, allowing the player to collide with the maze
- Inherits from `sprites.renderable_sprite`

- Attributes:
  - pos – inherited from renderable\_sprite
  - rot – inherited from renderable\_sprite but always set to 0 as walls don't rotate
  - imgs – inherited from renderable\_sprite
  - frame\_index – inherited from renderable\_sprite
  - frame\_countdown – inherited from renderable\_sprite
  - frame\_time – inherited from renderable\_sprite
- Methods:
  - VOID \_\_init\_\_ (array start\_pos)
    - Calls parent constructor with position and rotation 0
    - Load all animation frames into imgs using img\_loader
  - VOID update (dt) – inherited from renderable\_sprite
  - VOID render (dt) – inherited from renderable\_sprite

## SPRITES – CLASS: GATEWAY

- Purpose: blocks the path through the maze, and can only be removed by placing the correct block in it, which makes the game more challenging as it introduces multiple objectives per level
- Inherits from sprites.Wall
- Attributes:
  - pos – inherited from sprites.Wall
  - rot – inherited from sprites.Wall
  - imgs – inherited from sprites.Wall
  - frame\_index – inherited from sprites.Wall
  - frame\_countdown – inherited from sprites.Wall
  - frame\_time – inherited from sprites.Wall
  - colour – tuple (int R, int B, int G)
    - stores the colour of this gateway
- Methods:
  - VOID \_\_init\_\_ (array start\_pos, tuple colour)
    - Calls parent constructor and passes in start\_pos
    - Load imgs from img\_loader
    - Initialise colour
  - VOID update (dt) – inherited from Wall
  - VOID render (dt) – inherited from Wall

## SPRITES – CLASS: BLOCK

- Purpose: acts as an item that can be picked up, and then allows the player to release gateways to get through
- Inherits from sprites.Wall
- Attributes:

- pos – inherited from renderable\_sprite
- rot – inherited from renderable\_sprite but always set to 0 as blocks don't rotate
- imgs – inherited from renderable\_sprite
- frame\_index – inherited from sprites.Wall
- frame\_countdown – inherited from sprites.Wall
- frame\_time – inherited from sprites.Wall
- colour – tuple (int R, int B, int G)
  - stores the colour of this block
- Methods:
  - VOID \_\_init\_\_ (array start\_pos, tuple colour):
    - Calls parent constructor and passes in start\_pos
    - Load all animation frames into imgs using img\_loader
    - Initialise colour
  - VOID update (dt) – inherited from Wall
  - VOID render (dt) – inherited from Wall

## SPRITES – CLASS: CHECKPOINT

- Purpose: allows the player to restart part way through the level if they die
- Inherits from sprites.renderable\_sprite
- Attributes:
  - pos – inherited from renderable\_sprite
  - rot – inherited from renderable\_sprite but always set to 0 as checkpoints don't rotate
  - imgs – inherited from renderable\_sprite
  - frame\_index – inherited from renderable\_sprite
  - frame\_countdown – inherited from renderable\_sprite
  - frame\_time – inherited from renderable\_sprite
  - active – bool
    - if this is the most recently reached checkpoint, it is active, and is thus the player will respawn at it
- Methods:
  - VOID \_\_init\_\_ (array start\_pos)
    - Calls parent constructor with position and rotation 0
    - Load all animation frames into imgs using img\_loader
    - Initialize active to false
  - VOID update (dt)
    - If distance to player is less than one tile, call activate
  - VOID activate ()
    - set active to true
    - call Player's last\_checkpoint's deactivate
    - Store this checkpoint to Player's last\_checkpoint
    - Retrieve active imgs from img\_loader and store them to imgs
  - VOID deactivate ()
    - set active to false

- Retrieve deactivated imgs from img\_loader and store them to imgs
- VOID render (dt) – inherited from renderable\_sprite

## SPRITES – CLASS: KEY

- Purpose: used to unlock the exit. Acts as a secondary objective which must be met first before the player can complete the level.
- Inherits from sprites.renderable\_sprite
- Attributes
  - pos – inherited from renderable\_sprite
  - rot – inherited from renderable\_sprite but always set to 0 as exits don't rotate
  - imgs – inherited from renderable\_sprite
  - frame\_index – inherited from renderable\_sprite
  - frame\_countdown – inherited from renderable\_sprite
  - frame\_time – inherited from renderable\_sprite
- Methods:
  - VOID \_\_init\_\_(array start\_pos)
    - Call parent constructor with position and rotation 0
    - Load img from asset loader
    - Generate imgs by generating a list of blank Surfaces. Blit img onto each surface at different heights to represent the item bobbing up and down
  - VOID update(dt) – inherited from Sprites.Renderable\_Sprite
  - VOID render(dt) – inherited from Sprites.Renderable\_Sprite

## SPRITES – CLASS: EXIT

- Purpose: exists at the exit to the maze. This is the main objective for the level, and once the player has reached it they have beaten the level.
- Inherits from sprites.renderable\_sprite
- Attributes
  - pos – inherited from renderable\_sprite
  - rot – inherited from renderable\_sprite but always set to 0 as exits don't rotate
  - imgs – inherited from renderable\_sprite
  - frame\_index – inherited from renderable\_sprite
  - frame\_countdown – inherited from renderable\_sprite
  - frame\_time – inherited from renderable\_sprite
- Methods:
  - VOID \_\_init\_\_ (array start\_pos)
    - Calls parent constructor with position and rotation 0
    - Load all animation frames into imgs using img\_loader
  - VOID update (dt)
    - if the player is within one tile of the exit and has all the keys, push end screen to game state stack and play level complete sound
  - VOID render (dt) – inherited from renderable\_sprite

## SPRITES – CLASS: CAMERA

- Purpose: stores data about how to render other sprites onto the screen
- Attributes:
  - pos – array (int x, int y) [axis index]
    - stores where the camera is centered on screen
    - array allows for fast and mutable access to the position
- Methods:
  - VOID \_\_init\_\_ ()
    - Initializes pos to (0,0)
  - VOID update (dt)
    - Calculate pixel position on screen of player
    - Ensure this position won't allow the user to see outside of the maze.
    - If they can see out of the maze, calculate the pixel position of the edge of the screen and set the camera position so that it meets the edges of the maze to the edge of the screen
  - array wrld\_2\_scrn\_coord(array wrld\_coord)
    - transform the world coordinate to a screen coordinate using the camera position and screen size
    - return screen coordinate

## SPRITES – CLASS: TIMER

- Purpose: used to track how long the level took to complete
- Attributes:
  - start\_time – float
    - stores the system time when the level was started
  - total\_time – float
    - stores the total time this counter has been counting
- Methods:
  - VOID \_\_init\_\_ ()
    - Call reset ()
  - VOID update (dt)
    - Increase total\_time by dt
  - VOID reset ()
    - Set start time to system time
    - Set total\_time to 0

## SPRITES – TESTING



**Test environment:**

- File structure:
  - Root
    - Config
    - Asset\_loader
    - Sprites (module under test)

**Test table:**

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	Player – arrow inputs	Arrow keys are pressed	Valid	Player moves in the corresponding direction
2	Player – wasd inputs	Wasd keys are pressed	Valid	Player moves in the corresponding direction
3	Player – smooth movement	Wasd keys are pressed	Valid	Player accelerates up to speed when moving in a direction
4	Player – standing animation	No keys are pressed	Valid	Player displays the standing animation
5	Player – walking animation	Wasd or arrow keys are pressed	Valid	Player displays walking animation
6	Player – orientation	Wasd or arrow keys are pressed	Valid	Player turns around to face in whichever direction they are moving
7	Player – 1 sided collision	Player is moved such that it hits a wall on a single side	Valid	Player stops up against the wall and doesn't move through it
8	Player – 2 sided collisions	Player is moved so that it hits a wall on 2 sides	Valid	Player stops up against the walls and doesn't move through either of them
9	Player – colour changing	Number keys 1 to 6	Valid	Player sprite changes to reflect the current colour
10	Player – no collisions with blocks of the same colour	Player is moved towards a block who's colour equals that of the player	Valid	Player passes through block
11	Player – collisions with blocks of different colours	Player is move towards a block who's colour equals that of the player	Valid	Player stops at the edge of the block and doesn't move through it
12	Enemies – navigation	Enemy spawned in the maze	Valid	Enemy moves towards a open space in the maze
13	Enemies – smooth movement	Enemy spawned in the maze	Valid	As enemy goes round turns, it follows a smooth path, not jittering around

14	Enemies – navigation around maze	Enemy spawned in the maze	Valid	Enemy doesn't intersect with the maze as it is moving
15	Enemies – reallocating objective	Enemy has moved to target position	Valid	Enemy selects a new target and moves towards it
16	Enemies – attacking player	Player moves to same location as an enemy	Valid	Player health is decremented
17	Enemies – attack cooldown	Player moves to same location as an enemy	Valid	Player health is decremented at regular intervals, not every frame
18	Enemies – walking animation	Enemy is spawned in the maze	Valid	Enemy animates as it moves along
19	Enemies - orientation	Enemy is spawned in the maze	Valid	Enemy turns such that it points in the direction it moves
20	Walls - animation	Walls are spawned in the maze	Valid	Wall sprite displays its animation on screen
21	Walls – collisions with player	Player is moved to collide with the wall	Valid	Player can collide with multiple walls and walls are unaffected
22	Blocks – colours	Block is spawned in the maze	Valid	Blocks have colours randomly selected from all colours
23	Blocks – being picked up to left slot	player is near block and q key is pressed	Valid	Block is removed from the maze and stored into player's left inventory slot
24	Blocks – being picked up to right slot	player is near block and e key is pressed	Valid	Block is removed from the maze and stored into player's right inventory slot
25	Blocks – rendering in player inventory	Blocks stored in both slots of player's inventory	Valid	Blocks are rendered on player's backpack, with colours clearly visible
26	Blocks – placing from left slot	Q key is pressed with block in left inventory slot	Valid	Block from left slot is placed into the maze in front of the player, maintaining the same colour as before
27	Blocks – placing from right slot	E key is pressed with block in right inventory slot	Valid	Block from right slot is placed into the maze in front of the player, maintaining the same colour as before
28	Block – trying to place where there is already a wall or block	Q key is pressed with block in left slot and wall in front of the player	Valid	Block isn't placed, stays in player's backpack and block not placed sound plays

29	Gateway – collisions when closed	player moves up to edge of a gateway	Valid	Player can't move through the gateway
30	Gateway – collisions when open	Player moves towards edge of a gateway	Valid	Player can move through gateway
31	Gateway – opening with correct block	Player places block of correct colour in a gateway	Valid	Gateway changes from closed state to open state
32	Gateway – not opening with incorrect block	Player places block of incorrect colour in a gateway	Valid	Gateway stays closed and the block isn't taken from the player's inventory
33	Gateway – rendering	Gateway is opened	Valid	The visual state of the gateway changes from closed to open to show the player they can now traverse it
34	Checkpoint – player detection	Player walks past checkpoint	Valid	The checkpoint activates
35	Checkpoint – visual indication of activation	Player walks past checkpoint	Valid	The checkpoint sprite changes to visually show that it has been activated
36	Checkpoint – respawning	Player dies	Valid	Player respawns at the checkpoint, not at start
37	Second checkpoint – activation	One checkpoint is activated, then another, and then the player dies	Valid	Player respawns at most recently activated checkpoint
38	Key - rendering	Key is spawned	Valid	Key animates, moving up and down on the spot
39	Key - collection	Player is close to key	Valid	Key disappears and the player's key count increases
40	Exit – player wins	Player is close to the exit with all keys	Valid	Level closes and end screen shows
41	Exit – player doesn't have all the keys	Player is close to exit without all keys	Valid	Level keeps playing
42	Camera – player tracking	Player moves around	Valid	Camera moves around, following player
43	Camera – positioning on screen	Player moves around	Valid	Camera follows player such that they are located centrally on screen
44	Camera – positioning on screen at edges of maze	Player moves near edges of maze	Valid	Camera follows player but ensures that sprites beyond the edge of the maze aren't loaded
45	Timer – initialization	timer object is created	Valid	timer object created, with start time set to system time, without crashing
46	Timer – reset	Timer is reset	Valid	Timer sets start time to current system time and total time is 0

47	Timer - timing	Timer is ticked for duration of gameplay	Valid	Total_time holds the amount of time the timer has been counting for ; timer is accurate
----	----------------	--	-------	---

## MODULE: MENU\_SPRITES

Purpose: general purpose GUI elements to be instantiated across the multiple menus.

### MENU\_SPRITES – CLASS: TEXT

- Purpose: render text on the screen
- Attributes:
  - text - string
    - Stores the text that this text element displays
    - Allows for easy formatting as variables and be inserted and formatted using f strings
  - font – Font
    - stores the loaded font of this text
  - image - Surface
    - Image surface that is rendered to the screen each frame
  - rect - Rect
    - Stores where the text is located on screen
- Methods:
  - VOID \_\_init\_\_(string text)
    - Store text to attributes
    - Retrieve font name from config
    - Load font
    - Call rescale method
  - VOID update (dt)
    - Empty function to ensure all menu sprites can be updated
  - VOID rescale ()
    - Renders text to image using font
    - Scales image to the width and height of rect

### MENU\_SPRITES – CLASS: INPUT\_BOX

- Purpose: allows user to send text input into the game
- Attributes:
  - default\_text – string:
    - stores the text represented on this box if text is empty
    - shows the user what it intended to be typed into this input box

- text - string
  - Stores the text currently in this box
- font – Font
  - stores the loaded font of this input box
- selected – Boolean
  - stores whether this input box has been selected by the user or not
  - ensure that only the correct input box receives input from the keyboard
- keys\_to\_chars – Dictionary
  - converts keyboard event IDs into characters to append to text
  - dictionaries are a form of hash table and therefore provide high speed random access to the data they store, which will ensure keyboard events are registered quickly, making typing responsive
- image - Surface
  - Image surface that is rendered to the screen each frame
- rect - Rect
  - Stores where the input box is located on screen
- Methods:
  - VOID \_\_init\_\_(string default\_text, dict keys\_to\_chars)
    - Store default\_text and keys\_to\_chars to attributes
    - Retrieve font name from config
    - Load font
    - Call rescale method
  - VOID update (dt)
    - If cursor is within rect and left mouse button is clicked, set selected to true, otherwise set it to false
    - If selected, For each event in keyboard events:
      - If event ID is backspace, remove last char from text
      - If event ID is in keys\_to\_chars, append char to text
  - VOID rescale ()
    - If text is empty, render default\_text to image using font with colour grey
    - If selected, render text + “\_” to image using font
    - Else render text to image using font
    - Scales image to the width and height of rect

## MENU\_SPRITES – CLASS: TOGGLE

- Purpose: allows the user to input Boolean values into the game
- Attributes:
  - ticked – Boolean
    - Stores whether this toggle is true or false
  - Image – Surface
    - Stores the image to be rendered to the screen
  - rect – Rect
    - stores where on this toggle is located on screen

- Methods:
  - VOID \_\_init\_\_()
    - Initialize ticked to false
    - Initialize rect
    - Call rescale to render image
  - VOID update (dt)
    - If cursor is within rect and left mouse is clicked, negate ticked
  - VOID rescale ()
    - If ticked, set image to toggle\_ticked from img\_loader
    - If not ticked, set image to toggle\_unticked from img\_loader
    - Scale image to width and height of rect

## MENU\_SPRITES – CLASS: SLIDER

- Purpose: allows the user to slide a bar to any position to give a linear input
- Attributes:
  - val – float
    - Value from 0 to 1 that indicates how far along this slider is
  - grabbed - Boolean
    - true if the slider has been grabbed and is being moved around
  - image – Surface
    - stores the image to be rendered to the screen
  - rect – Rect
    - stores the position of the slider on the screen
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialize val to 0
    - Initialize rect
  - VOID update (dt)
    - If grabbed, set slider value depending on how far along the slider's length cursor's position is and call rescale
    - If cursor is above this slider's thumb and left mouse button is held, set grabbed to true
  - VOID rescale ()
    - Set image to slider\_track from img\_loader
    - blit slider\_thumb from img\_loader onto image at correct position dependent on val
    - rescale image to width and height of rect

## MENU\_SPRITES – CLASS: SPINNER

- Purpose: allows the user to select from a list of ordered predefined options
- Attributes:
  - options – array (string option1, string, option2 ...) [option\_index]
    - Stores the possible options for this spinner
  - index – int

- stores the index of which option is currently selected
- font – Font
  - stores the loaded font used in this spinner
- image – Surface
  - stores the image to be rendered to the screen
- rect – Rect
  - stores the spinner's location on screen
- Methods:
  - VOID \_\_init\_\_ (list options)
    - Store options to attribute
    - Initialize index to 0
    - Get font name from config
    - Initialize font
  - VOID update (dt)
    - If cursor is over the portion of the rect that represents the left button and left mouse button is pressed, decrement index and call rescale. If index is at zero play spinner\_end sound from snd\_loader
    - If cursor is over the portion of the rect that represents the right button and left mouse button is pressed, increment index and call rescale. If index is at length of options - 1, play spinner\_end sound from snd\_loader
  - VOID rescale ()
    - Set image to spinner\_buttons from img\_loader
    - Use font to render options[index]
    - Blit into center of image
    - Rescale image to width and height of rect

## MENU\_SPRITES – CLASS: BUTTON

- Purpose: the user can press buttons to interact with the ui
- Attributes:
  - text – string
    - stores the text that the button says on it
  - font – Font
    - stores the loaded font used in this button
  - pressed – list (Bool left\_button, Bool middle\_button, Bool right\_button)[button\_index]
    - stores which buttons are pressed on this button on this frame
  - rising\_edges – list (Bool left\_button, Bool middle\_button, Bool right\_button)[button\_index]
    - stores which buttons have had a rising edge this frame
  - falling\_edges – list (Bool left\_button, Bool middle\_button, Bool right\_button)[button\_index]
    - stores which buttons have had a falling edge this frame
  - image – Surface
    - stores the image to be rendered to the screen
  - rect – Rect
    - stores the position of this button on screen

- Methods:
  - VOID \_\_init\_\_ (string text)
    - Store text as attribute
    - Get font name from config
    - Initialize font
  - VOID update (dt)
    - Set rising\_edges and falling\_edges to all falses
    - Get pressed mouse buttons and store to new\_pressed
    - For button\_index from 0 to 2
      - if new\_pressed[button\_index] is true and pressed[button\_index] is false, set rising\_edges[button\_index] to true
      - if new\_pressed[button\_index] is false and pressed[button\_index] is true, set falling\_edges[button\_index] to true
    - set pressed to new\_pressed
  - VOID rescale ()
    - Set image to button\_img from img\_loader
    - Use font to render text and blit this to image
    - Rescale image to width and height of rect

## MENU\_SPRITES – TESTING

### Test environment:

- File structure:
  - Root
    - Host
      - Runs a simple main game loop
      - Has a single UI screen, which has one of every element for testing
      - Prints when any element changes at all
    - Asset\_loader
      - Provides stripped down equivalents to loader objects, which print when any function is called and return generic default data
    - Menu\_Sprites (file under test)
    - Config
      - Prints when data is accessed and provides default values

### Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	Text – initialization	A text element is instantiated with “text1” and rendered	Valid	Text 1 appears on screen, using the font from font .config



2	Text – rescaling	Screen is rescaled	Valid	Text 1 still appears on the screen in the same position, now scaled down as to maintain the proportions on screen
3	Input_box - initialization	Input box is instantiated with no default text	Valid	Input box appears on screen
4	Input_box – default text	Input box is instantiated with default text “input box 1”	Valid	Default text renders within input box
5	Input_box - selection	cursor is hovered over input_box and left clicked	Valid	Input_box is highlighted, indicating it will accept text input
6	Input_box – registering allowed keystrokes	Input_box is selected and allowed keys are pressed	Valid	The characters corresponding to the keystrokes appear in the input box
7	Input_box – not registering disallowed keystrokes	input box is selected and disallowed keystrokes are pressed	Invalid	The characters of the corresponding keystrokes aren’t added to the input box
8	Input_box – deselection	cursor is moved away from input box and left button is pressed	Valid	Input box is deselected: highlight goes away and keystrokes are no longer registered
9	Input_box – rescaling	Screen is rescaled	Valid	Input box rescales as to maintain it’s proportions on screen
10	Toggle – initialization	New toggle box is initialised	Valid	New toggle object is created and rendered to screen with no crashing
11	Toggle – rendering	New toggle box is initialised	Valid	Toggle box appears on screen unticked
12	Toggle – toggling	cursor is moved over toggle and left button is pressed	Valid	Toggle changes state and renders that it is now in the other state
13	Toggle – rescaling	Screen is rescaled	Valid	Toggle rescales such that it maintains the same proportions on screen
14	Slider – initialization	New slider is initialized	Valid	New slider object appears on screen and causes no crashing
15	Slider - grabbing	cursor hovered over slider and left button is pressed	Valid	Slider thumb moves around so that it follows the mouse cursor
16	Slider – thumb stays on slider	Slider grabbed and cursor is moved around	Valid	If the cursor goes of the end of the slider, the thumb stops at the corresponding end until cursor returns
17	Slider – releasing	Left mouse button is released	Valid	Thumb stays in the same place it was in before left button was released
18	Slider – rescaling	Screen is rescaled	Valid	Slider rescales such that it’s proportions on screen are maintained

19	Spinner - initialization	A new spinner object is initialized with options: o1, o2, o3	Valid	New spinner object is created and appears on screen
20	Spinner – right button	Cursor hovers over right button and right click is pressed	Valid	Spinner moves to next option
21	Spinner – left button	Cursor hovers over left button and left click is pressed	Valid	Spinner moves to previous option
22	Spinner - ends	Left and right buttons are pressed until spinner gets to either end	Valid	Spinner moves to the final option and then doesn't go any further and doesn't wrap around
23	Spinner – rescaling	Screen is rescaled	Valid	Spinner rescaled as to maintain the same proportions on screen
24	Button – initialization	A new button object is initialized with text b1	Valid	Button object appears on screen with correct text and font
25	Button – press registration	Cursor hovers over button and mouse buttons are pressed	Valid	Console output saying which buttons are pressed equating to the mouse buttons that are currently pressed
26	Button – rising edge registration	Cursor hovers over button and mouse buttons are pressed	Valid	Console output saying when there is a rising edge of each button as they are depressed
27	Button – falling edge detection	Cursor hovers over button and mouse buttons are pressed	Valid	Console output saying when there is a falling edge of each button as they are released
28	Button – clicking elsewhere on the screen	Cursor is moved elsewhere on screen and mouse buttons are pressed	Valid	
29	Button - rendering	Cursor hovers over button and mouse buttons are pressed	Valid	Button visually indicates that has been pressed
30	Button – rescaling	Screen is rescaled	Valid	Button rescales to maintain same proportions on screen

**MODULE: MENU\_SYSTEM**

Purpose: handles all of the user interfaces that are presented during different states of the game

**MENU\_SYSTEM – CLASS: UI\_SCREEN**

- Purpose: this is a template class that provides basic functionality of a menu screen for other screens to inherit from
- Attributes:
  - Elements – SpriteGroup
    - contains all sprites for this screen and provides functionality for updating and rendering them
  - background – Surface
    - an image to be rendered before the rest of the elements
    - defaults to False, so that there isn't necessarily a background
- Methods:
  - VOID \_\_init\_\_ ()
    - Empty template method to ensure this screen can be updated without the game crashing because it can't find a constructor for a screen
  - VOID tick (array event\_list, float dt)
    - Updates all sprites in elements
    - Render background if present
    - Render all sprites in elements
  - VOID Rescale ()
    - Call rescale method of all elements

## MENU\_SYSTEM – CLASS: MAIN

- Purpose: updates and renders the main menu screen
- Inherits from UI\_Screen
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - main\_title\_text – Menu\_Sprites.Text
    - the text at the top of the title screen
  - Start\_Button – Menu\_Sprite.Button
    - Allows the user to open the start menu
  - options\_button – Menu\_Sprite.Button
    - Allows the user to open the options menu
  - scoreboard\_button – Menu\_Sprite.Button
    - Allows the user to open the Scoreboard
  - close\_button – Menu\_Sprite.Button
    - Removes all items from the game\_state\_stack to close the game
- Methods:
  - VOID \_\_init\_\_ ()
    - Initializes maint\_title\_text, start\_button, options\_button, scoreboard\_button and close\_button
    - Loads background image from image loader and stores into attribute
    - Call rescale to render image
  - VOID tick (array event\_list, float dt)

- Call parent tick method
- If a button has been pressed, push the corresponding screen onto the game state stack
- If the close button has been pressed, clear the game state stack
- VOID rescale ()
  - Set rect of all elements such that they are stacked vertically in the center of the screen
  - Call parent rescale method

## MENU\_SYSTEM – CLASS: START

- Purpose: updates and renders the level start screen
- Inherits from UI\_Screen
- Attributes:
  - elements – inherited from UI\_Screen
  - background - inherited from UI\_Screen
  - start\_screen\_text
    - shows the user which screen is currently open
  - width\_input\_box
    - allows the user to enter in a width for the level
  - height\_input\_box
    - allows the user to enter in a height for the level
  - seed\_input\_box
    - allows the user to enter in the random number seed for the level. This will allow them to play the same level multiple times or play the same level as seen in the scoreboard
  - start\_button
    - calls the code that generates the level with the requisite settings
  - exit button
    - allows the user to return to the main menu screen
- Methods:
  - VOID \_\_init\_\_ ()
    - Initializes start\_screen\_text, width\_input\_box, height\_input\_box, seed\_input\_box, start\_button and exit\_button
    - Loads background image from image loader and stores to attribute
    - Call rescale to render image
  - VOID tick (array event\_list, float dt)
    - Call parent tick method
    - When load button is pressed, check if input\_boxes contain only numbers and if so, call game.start\_level with the width, height and seed values of the boxes
    - If exit button has been pushed, clear game state stack and push main's tick method to game state stack.
  - VOID rescale ()
    - Set rect of all elements such that they form a grid
    - Call parent rescale method

## MENU\_SYSTEM – CLASS: END

- Purpose: updates and renders all functionality for the end screen
- Inherits from UI\_Screen
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - Score\_added – Boolean
    - Stores if the user has already added their score to the scoreboard; prevents the player submitting the same run to the scoreboard multiple time
  - end\_text
    - shows user what screen they are currently on
  - time\_text
    - shows user how long they took to complete the level
  - name\_input\_box
    - allows user to input their name it be added to the scoreboard
  - add\_to\_scoreboard\_button
    - adds user's score to scoreboard
  - exit\_button
    - allows user to return to the main screen
- Methods:
  - VOID \_\_init\_\_ ()
    - Initializes end\_text, time\_text, name\_input\_box, add\_to\_scoreboard\_button and exit\_button
    - Loads background image from image loader and stores to attribute
    - Call rescale to render image
  - VOID tick (array event\_list, float dt)
    - Call parent tick method
    - If add\_to\_scoreboard\_button has been pushed and score\_added isn't true, call scoreboard's add\_score method, set score\_added to true and set add\_to\_scoreboard\_button's colour to grey to indicate it doesn't work anymore
    - If exit button has been pushed, clear game state stack and push main\_menu screen to game state stack
  - VOID rescale()
    - Set element's rects so that they are stacked on top of the other
    - Call parent rescale method

## MENU\_SYSTEM – CLASS: SCOREBOARD

- Purpose: update and render the scoreboard screen so the player can see how well they have done
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen

- title\_text – menu\_sprites.Text
  - Title to indicate to the user which menu they are on
- scoreboard\_data – array [entry\_index][field\_index]
 

Name1	Time1	Width1	Heigh1	Seed1
Name2	Time2	Width2	Heigh2	Seed2
⋮	⋮	⋮	⋮	⋮
NameN	TimeN	WidthN	HeighN	seedN

  - Stores loaded values for all scoreboard values
  - Stops the scoreboard from having to be constantly loaded
  - 2d array allows for all items to be accessed equally quickly
- text\_rows – array (menu\_sprites.Text row1, menu\_sprites.Text row2 ...)[row\_index]
  - stores the text sprites that are used to render the rows
- exit button – menu\_sprites.Button
  - allows the user to exit the scoreboard screen
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialize title\_text and exit\_button
    - Load background image from image loader and store to attribute
    - Call load
    - Call rescale to render image
  - VOID tick (array event\_list, float dt)
    - Call parent tick method
    - if exit\_button or esc key is pressed, pop top off game state stack
  - VOID rescale ()
    - Call parent rescale method
    - Set rects of title and exit button in top centre
    - Set rects of text\_rows such that the top 10 are vertically stacked on screen
  - VOID load()
    - Load scoreboard file from config.scoreboard\_path
    - Split by newlines to get rows
    - Split rows by commas and store to scoreboard\_data
    - Close scoreboard file
  - VOID save()
    - Load scoreboard file from config.scoreboard\_path
    - Concatenate rows of scoreboard\_data to get lines
    - Write lines to file
    - Close scoreboard file
  - VOID add\_score (name, time, width, height, seed)
    - Append (name, time, width, height, seed) to scoreboard\_data
    - Call save () to save data
    - Call load () to reload the scoreboard

- Purpose: updates and renders the pause menu, allowing the user to take a break from the game
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - pause\_menu\_text
  - resume\_button
  - options\_button
  - exit\_button
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialize resume\_button, options\_button, exit\_button
    - Load paused background from image loader and store to asset
  - VOID tick (array event\_list, float dt)
    - Call parent tick method
    - If resume\_button or esc key pressed, pop top value of game state stack
    - If options button pressed, push options screen onto game state stack
    - If exit button pressed, clear game state stack and push main's tick method
  - VOID rescale ()
    - Set rects of elements such that they are vertically stacked in the middle of the screen
    - Call parent rescale method

## MENU\_SYSTEM – CLASS: OPTIONS

- Purpose: renders the general options screen
- Inherits from UI\_Screen
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - options\_title\_text – menu\_sprites.Text
    - Title to indicate to the user which menu they are on
  - gfx\_button – menu\_sprites.Button
    - opens graphics options menu
  - snd\_button – menu\_sprite.Button
    - opens sound options menu
  - exit\_button
    - allows the user to return to the previous screen
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialize options\_title\_text, gfx\_button, snd\_button and exit\_button
    - Load background image from image loader and store to attribute
  - VOID tick (array event\_list, float dt)
    - Call parent tick method
    - If gfx\_button is pressed, push gfx\_Options screen to game state stack

- If snd\_button is pressed, push snd\_Options screen to game state stack
- if exit\_button or esc key is pressed, pop top off game state stack
- VOID rescale ()
  - Set rects of elements so that they are vertically centered on the screen and vertically stacked
  - Call parent rescale method

## MENU\_SYSTEM – CLASS: GFX\_OPTIONS

- Purpose: updates and renders the graphics options screen
- Inherits from UI\_Screen
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - gfx\_title\_text – menu\_sprites.Text
    - Indicates which menu the user is looking at
  - res\_spinner – menu\_sprites.Spinner
    - allows the user to scroll through available resolutions to select one to use
  - fullscreen\_toggle – menu\_sprites.Toggle
    - allows the user to toggle if the game is in fullscreen mode or not.
  - vsync\_toggle – menu\_sprites.Toggle
    - Allows the user to choose if the game's framerate is limited to 60 fps or not
  - apply\_button – menu\_sprites.Button
    - the user presses this button for the graphics changes to take affect
  - exit\_button – menu\_sprites.Button
    - allows the user to return to the previous screen
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialize gfx\_title\_text, res\_spinner, fullscreen\_toggle, vsync\_toggle, apply\_button and exit\_button
    - set res\_spinner to config.resolution
    - set fullscreen\_toggle to config.fullscreen
    - set vsync\_toggle to config.vsync
    - Load background image from image loader and store to attribute
  - VOID tick (array event\_list, float dt)
    - Call parent tick method
    - If apply button is pressed, store graphics settings to config and call game's rescale method
    - If exit button or esc key is pressed, pop top of the game state stack
  - VOID rescale ()
    - Set rects of elements such that they are vertically stacked
    - Call parent rescale method



## MENU\_SYSTEM – CLASS: SND\_OPTIONS

- Purpose: updates and renders the sound options screen
- Inherits from UI\_Screen
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - snd\_options\_text – menu\_sprites.Text
  - music\_slider – menu\_sprites.Slider
  - game\_slider – menu\_sprites.Slider
  - exit\_button – menu\_sprites.Button
- Methods:
  - VOID \_\_init\_\_ ()
    - Initialize snd\_options\_text, game\_slider, music\_slider and exit\_button
    - Set game slider to config.game\_vol
    - Set music slider to config.music\_vol
    - Load background image from image loader and store to attribute
  - VOID tick (array event\_list, float dt)
    - if any sliders have changed, store the values to config
    - call game.load\_snd\_vol
  - VOID rescale ()
    - Set rects of elements such that they are vertically stacked
    - Call parent rescale method

## MENU\_SYSTEM – CLASS: LEVEL

- Purpose: updates and renders the level; this code runs every tick when the user is playing a level.
- Inherits from UI\_Screen
- Attributes:
  - Elements – inherited from UI\_Screen
  - Background - inherited from UI\_Screen
  - game\_timer – Sprites.timer
    - times how long the level has been played for
  - maze – Maze\_Gen.Maze
    - stores all data for the maze
  - player – Sprites.Player
    - the player character that the user controls
- Methods:
  - VOID \_\_init\_\_ (tuple size, int seed)
    - Initialize new maze with parameters size and seed

- Initialize player with position maze.start
- Initialize new game\_timer
- VOID tick (array event\_list, float dt)
  - Runs playing state of the game
  - If esc key is pressed push tick\_pause\_screen to game state stack
  - Parses event\_list and acts on each event
  - Updates maze.all\_sprites
  - Renders background
  - Renders all sprites
- 
- VOID rescale()
  - Call parent rescale method

## MENU\_SYSTEM - TESTING

### Test environment:

- File structure:
  - Root
    - Main
    - Asset\_loader
      - Provides stripped down equivalents to loader objects, which print when any function is called and return generic default data
    - Menu\_System (file under test)
    - Menu\_Sprites (already developed)
    - Config
      - Prints when data is accessed and provides default values

### Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior
1	Main menu - start button	Start button pressed	Normal	Start screen appears
2	Main menu – options button	Options button pressed	Normal	Options screen appears
3	Main menu – scoreboard button	Scoreboard button pressed	Normal	Scoreboard screen appears
4	Main menu – close button	Close button pressed	Normal	Game exits
5	Start screen – width box	Input number into width box: 123	Normal	Number can be entered
6	Start screen – width box	Input characters into width box: abc	invalid	Characters can't be entered
7	Start screen – height box	Input number into height box: 123	Normal	Number can be entered

8	Start screen – height box	Input characters into height box: abc	invalid	Characters can't be entered
9	Start screen – seed box	Input string into width box: string123	Normal	Any string can be entered
10	Start screen – exit button	Exit button pressed	Normal	Main menu screen appears
11	Start screen – start button	Start button pressed	Normal	Dummy level screen appears
12	End screen – name box	Name string can be entered: Adam	Normal	Name string can be entered
13	End screen – name box	Strings with invalid chars cant be entered: ';/#\	Invalid	Non name string can't be entered
14	End screen – add to scoreboard button	Add to scoreboard button pressed	Normal	Scoreboard stores that row to scoreboard file
15	End screen – exit button	Exit button pressed	Normal	Main menu screen appears
16	Scoreboard – add score	End screen's add score button is pressed with user name Adam,	Normal	Score is added to scores .csv file
17	Scoreboard – rendering	Scoreboard button on main menu is pressed	Normal	Scoreboard appears on screen, listing scores of all previous players
18	Scoreboard - rescaling	Screen is rescaled	Normal	Scoreboard is re rendered such that it is still centered on screen
19	Scoreboard - saving	Game is closed	Normal	New entry is present in scoreboard.csv
20	Scoreboard - loading	Game is reopened and scoreboard button is pressed	Normal	All previous entries are present in the scoreboard
21	Scoreboard – exit button	Exit button is pressed	normal	Returns to main menu
22	Pause – resume button	Resume button is pressed	Normal	Returns to gameplay
23	Pause – options button	Options button is pressed	Normal	Options screen opens
24	Pause – exit button	Exit button is pressed	Normal	Main menu screen appears
25	Pause screen - rescaling	Screen is rescaled	Normal	Elements are still arranged centrally on screen
26	Options - rendering	Options screen is opened	Normal	Options text and buttons are on screen
27	Options – opening graphics menu	Graphics button is pressed	Normal	Graphics options screen opens
28	Options – opening sound menu	Sound button is pressed	Normal	Sound options screen opens
29	Options – exit button	Exit button is pressed	Normal	Returns to pause screen

30	Options – rescaling	Screen is rescaled	Normal	Options text and buttons are placed on screen such that they are still central
31	GFX options – rendering	Graphics screen is opened	Normal	Graphics text, spinners, toggles and buttons appear centrally arranged on screen
32	GFX options – resolution selection works	Spinner buttons are pressed	Normal	Resolution spinner can cycle between all available resolution options
33	GFX options – fullscreen toggle works	Fullscreen toggle is pressed	Normal	Fullscreen toggle toggles between being ticked and unticked
34	GFX options – vsync toggle works	vsync toggle is pressed	Normal	vsync toggle toggles between being ticked and unticked
35	GFX options – apply buttons	Apply button is pressed	Normal	Resolution, fullscreen and vsync settings are applied, without the game crashing
36	GFX options – exit button	Exit button is pressed	Normal	Returns to options screen
37	GFX options - rescaling	Screen is rescaled	Normal	Graphics text, spinner, toggle, and buttons still appear in center of the screen.
38	SND options - rendering	Sound options are opened	Normal	Sound options text, sliders and buttons render
39	SND options – music volume	Music volume slider is changed	Normal	Music volume changes, with feedback sound so user knows how loud the volume is
40	SND options – game volume	Game volume slider is changed	Normal	Game volume changes, with feedback sound so user knows how loud the volume is
41	SND options – exit button	Exit button is pressed	Normal	Options screen renders
42	SND options – rescaling	Windows is rescaled	Normal	Sound options text, sliders and buttons render
	Level – will be tested during integration testing as it is dependent on many other modules			

### C. DEVELOPING THE CODED SOLUTION (“THE DEVELOPMENT STORY”)

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

### D. EVALUATION

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

## PROJECT APPENDIXES

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

- Complete Code Listing (ESSENTIAL)
- Interview Transcripts
- Meeting notes
- Observation notes or questionnaires