

OCR GCE A COMPUTER SCIENCE PROJECT H446-03

Name : Alexander Mills
Candidate Number : 9120
The Bishop Of Winchester Academy : 55151
Title of Project : Colour Between The Lines

H446-03 – PROJECT CONTENTS

Table of Contents

A. Analysis.....	5
Outline.....	5
Stakeholders.....	6
Game Research: Tetris	6
Game research: Hue.....	10
Game Research: World's Hardest Game.....	13
Survey.....	14
Planning	14
Survey Response review	15
Graphics:.....	15
Sound:.....	16
Level Design:.....	17
Other Feedback:	18
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
Asset Selection	36
Theming:	37
Font:	37
Tiles:	38
Character sprites:	39
Systems diagram	40
Overall program layout	41
Module Declarations	43
Module: Config	43
Config – Class: game_config	43
Config – testing	45
Module: Asset_loader	46
Asset_loader – Class: Img_loader	46
Asset_Loader – Class: Snd_Loader	47
Asset_loader – Class: Sprite_Sheet	47
Asset_Loader – Testing and Verification	48
Module: Main	49
Main - Class: Game	49
Main – testing	51
Module: maze_gen	52
Maze_GEN - Class: Maze	52
Maze_GEN - Algorithms	55
Maze_Gen – Testing:	58
Module: Sprites	60
Sprites – Class: Renderable_Sprite	60
sprites – Class: Player	60
Sprites – Class: Enemy	63
Sprites – Class: Wall	63
Sprites – Class: Gateway	64
Sprites – Class: Block	64
Sprites – Class: Checkpoint	65
Sprites – Class: Key	66
Sprites – Class: Exit	66
Sprites – Class: Camera	67
Sprites – Class: Timer	67
Sprites – Testing	67
Module: Menu_Sprites	71
Menu_Sprites – Class: Text	71
Menu_Sprites – Class: Input_box	72
Menu_Sprites – Class: Toggle	72
Menu_Sprites – Class: Slider	73
Menu_Sprites – Class: Spinner	74
Menu_Sprites – Class: Button	74
Menu_Sprites – Testing	75
Module: Menu_System	78

Menu_System – Class: UI_Screen.....	78
Menu_System – Class: Main.....	78
Menu_System – Class: Start	79
Menu_System – Class: End	80
Menu_System – Class: Scoreboard.....	81
Menu_System – Class: Pause.....	82
Menu_System – Class: Options	82
Menu_System – Class: GFX_Options	83
Menu_System – Class: SND_Options.....	84
Menu_System – Class: Level.....	84
Menu_system - Testing	85
C. Developing the coded solution (“The development story”).....	89
Development Plan	89
Stage 1: Config Module.....	90
Stage 1: Development goals	90
Stage 1: Development process	90
Stage 1: Unit Testing	92
Stakeholder Review	95
Stage 2 : Asset_Loaders	96
Stage 2: Development Goals.....	96
Stage 2: Development Process	96
Stage 2: Unit Testing	99
Stakeholder Review	104
Stage 3: Main Module.....	105
Stage 3: Development Goals.....	105
Stage 3: Development Process	105
Stage 3: Unit Testing	107
Stage 3: Stakeholder Review	115
Stage 4: Maze Gen	115
Stage 4: Development Goals:.....	115
Stage 4: Development Process	115
Stage 4: Unit Testing	122
Stage 4: Stakeholder Review	136
Stage 5: Sprites	137
Stage 5: Development Goals:.....	137
Stage 5: Development Process:	137
Stage 5:Unit Testing:.....	155
Stage 5: Stakeholder Review:	185
Stage 6: Menu Sprites.....	185
Stage 6: Development Goals.....	185
Stage 6: Development Process	186
Stage 6: Unit Testing	194
Stage 6: Stakeholder Review	205
Stage 7 : Menu System	205
Stage 7: Development Goals.....	205
Stage 7: Development process	206
Stage 7: Unit testing	226
Stage 7: Stakeholder Review	241
Stage 8: Integration	241
Task1 : collecting The Files:.....	241
Task 2: Merging Sprites Host with Menu System	242

Task 3: Implementing the Win Condition	243
Task 4: integrating sound control	244
Task 5: Polishing.....	245
D. Evaluation.....	245
Post development testing – success criteria	246
Graphics - 1.....	246
Player design - a.....	246
Environment design - b.....	247
ENemy design - c.....	248
Objective design - d	249
User interface - 2	249
Main menu - a.....	249
Pause menu - b	250
Options menus - c.....	251
GUI design - d.....	252
Sound - 3	253
Sprite sounds - a	253
Level sounds - b	253
Background sounds - c	253
Level design - 4	254
Maze layout - a	254
Maze population - b.....	255
Enemies - c.....	256
Checkpoints – d	257
Win criteria - e	257
Game mechanics - 4	258
Player controller - a	258
Enemy controller - b	259
usability features.....	260
Stakeholder review	262
Evaluation.....	265
Successes	265
Limitations	265
Maintainability	266
Project Appendices.....	267
code listing	267
config.py	267
Asset_Loader.py	269
Main.py.....	273
Maze_Gen.py.....	276
Sprites.py	284
Menu_Sprites.py.....	302
Menu_System.py.....	310

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.

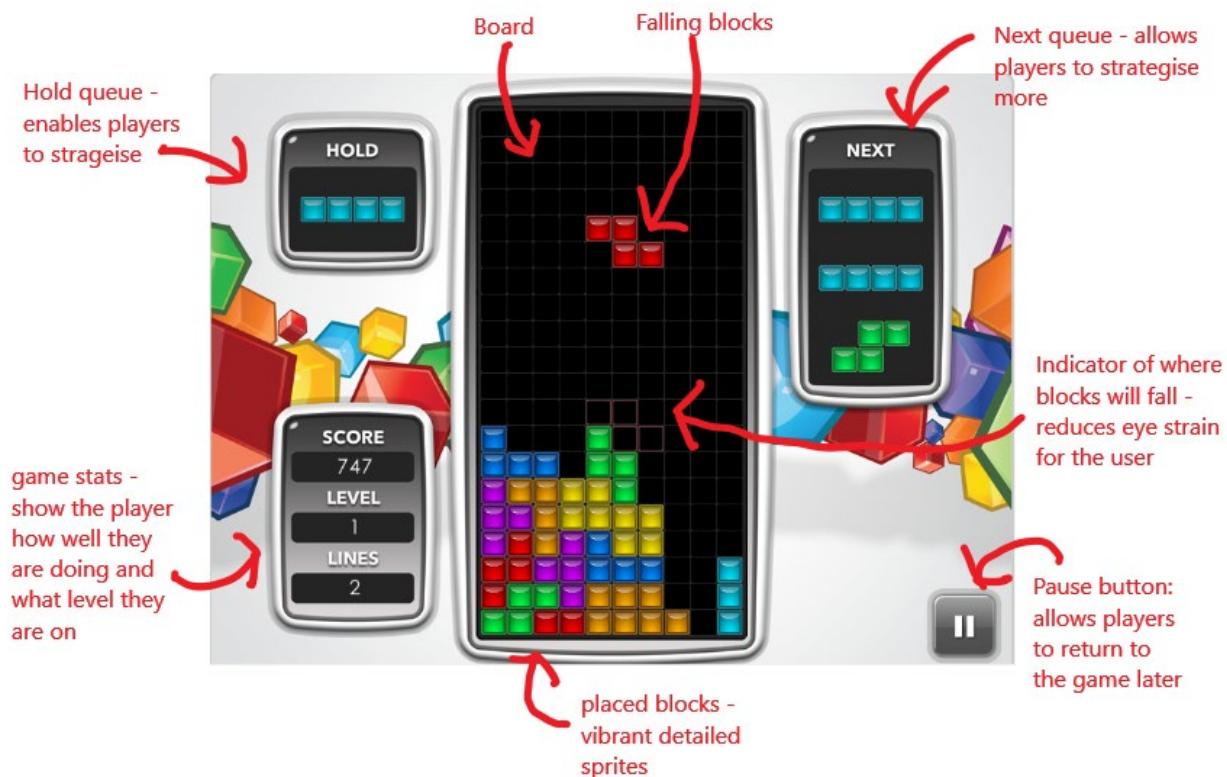
Main menu:



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.

Candidate Name: Alexander Mills Candidate Number: 9120

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.

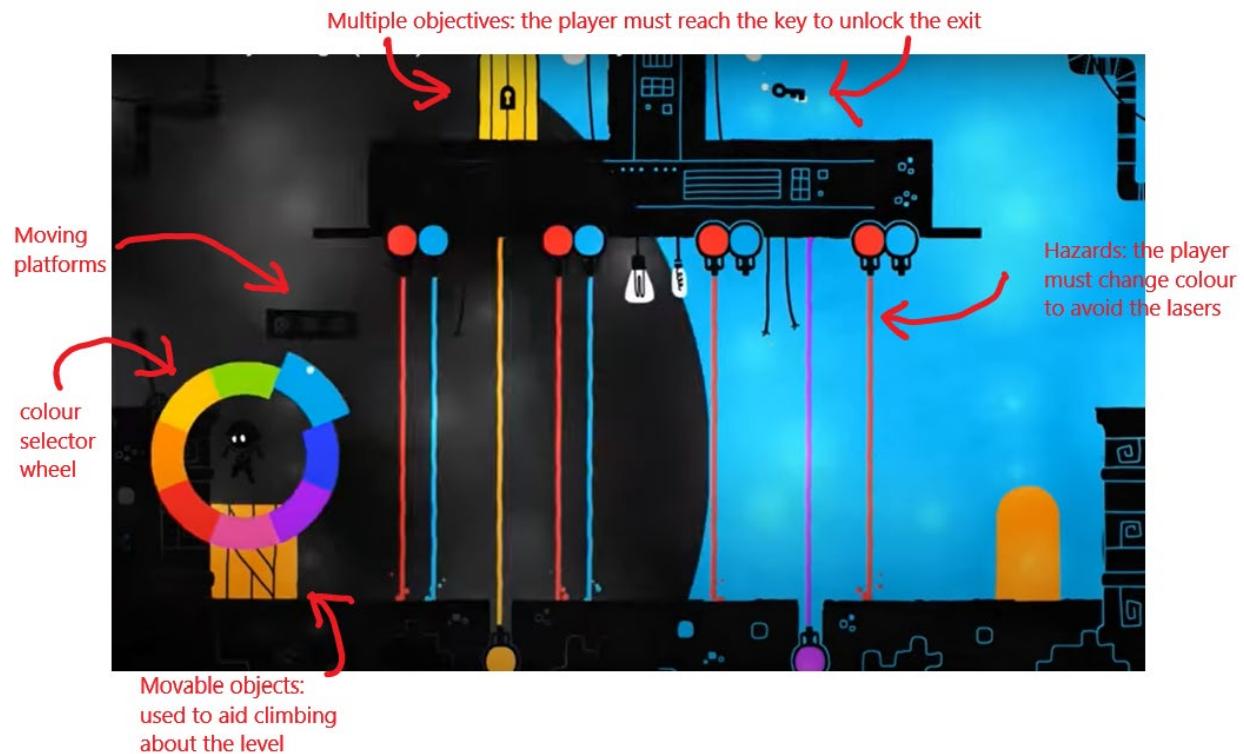
GAME RESEARCH: HUE

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.

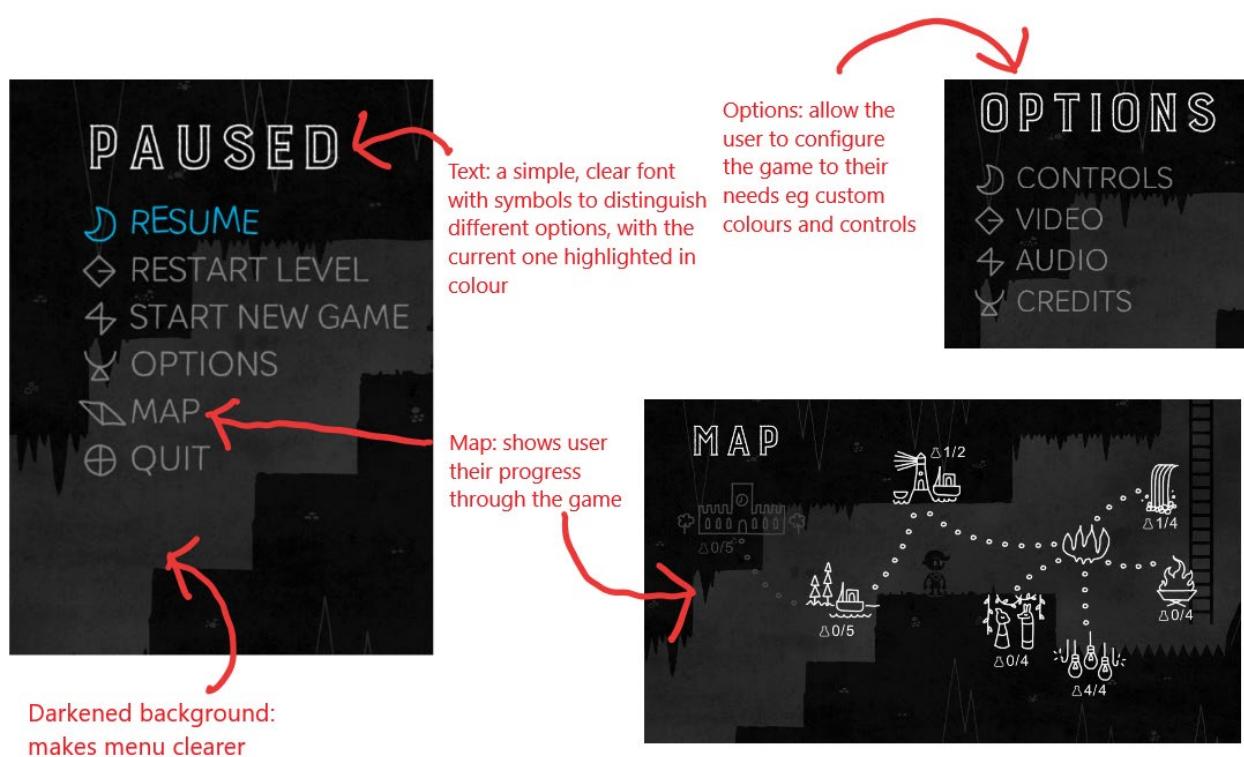
Typical level:



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:



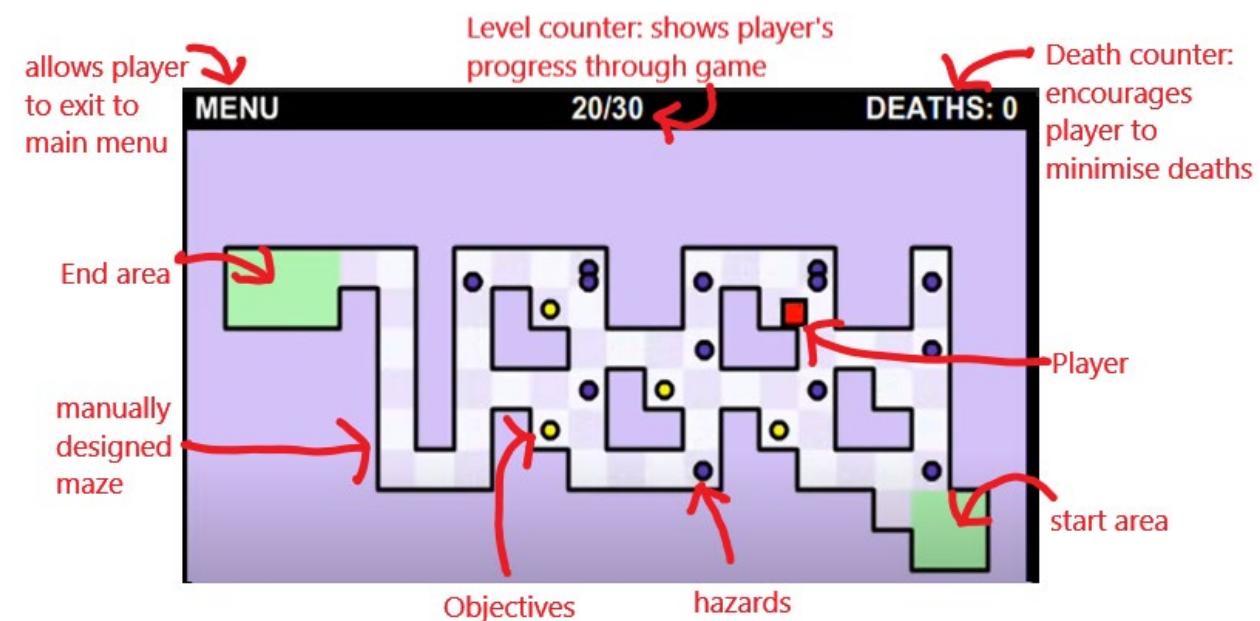
The pause menu allows the user to pause the game, allowing them to return to it later. It also provides some configuration menus for the user to tailor their experience to their needs. This includes a controls menu, where the user can learn the controls or configure them, a video menu where the user 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.

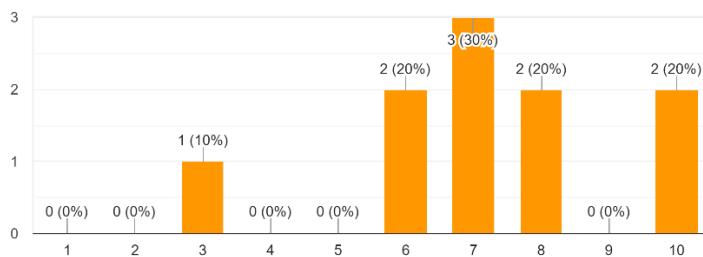
Question	Input type	Function
How important are graphics to make a puzzle game enjoyable?	Slider: 1 to 10 Comments box	Gauges how much work must be put into graphics to meet user needs
How much control over graphics is needed in the settings?	Multi choice: <ul style="list-style-type: none"> • No options • basic options: resolutions, vsync, Fullscreen • advanced: frame rate, rendering settings, toggleable visuals • extensive: full colour scheme configurability, all rendering settings 	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"> • no options • a slider for game volume, and a slider for music volume • all game sounds have individual sliders 	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"> • based on visual theme • based on the style of puzzles • based on a narrative 	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:

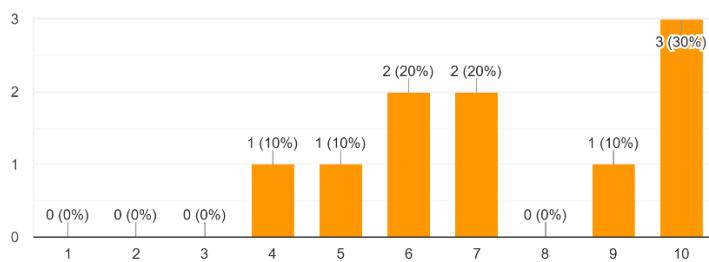
How important are Graphics to make a puzzle game enjoyable?

10 responses



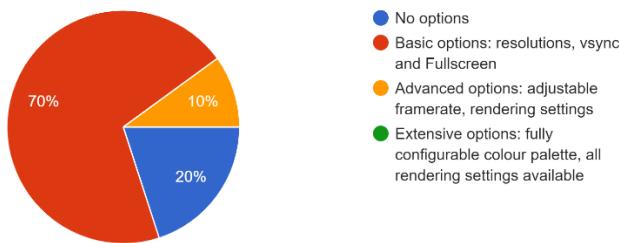
How important are Visual effects to make a puzzle game enjoyable?

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

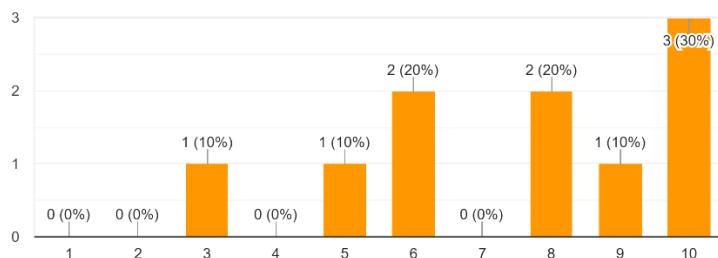
From the graphics part of the survey, it is evident that potential users prefer graphical fidelity over visual effects, though they are both very important. This means that I will have to spend more time on textures and sprites, ensuring they are high resolution with ample colour depth. I won't have 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:

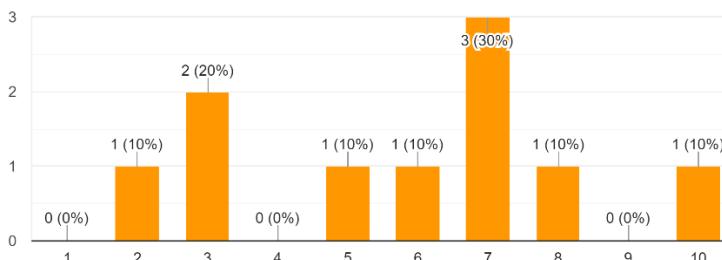
How important are Sound effects to make a puzzle game enjoyable?

10 responses



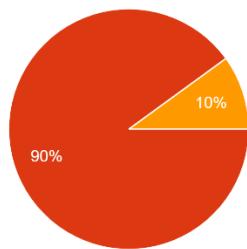
How important is Background Music to make a puzzle game enjoyable?

10 responses



How much control over sound is needed in the settings?

10 responses



- No options
- A slider for game volume and a slider for background music volume
- All game sounds have individual sliders

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.

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.

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.

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

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?

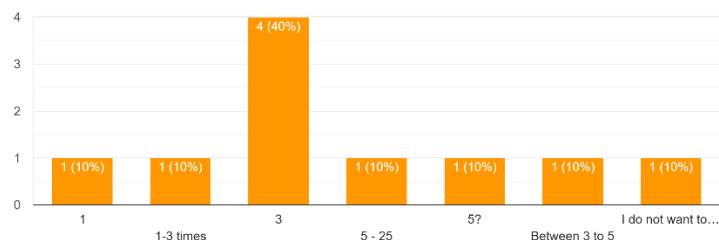
5-10 minutes

Up to 5 minutes (at least that's when it starts getting frustrating and I normally quit)

20 minutes

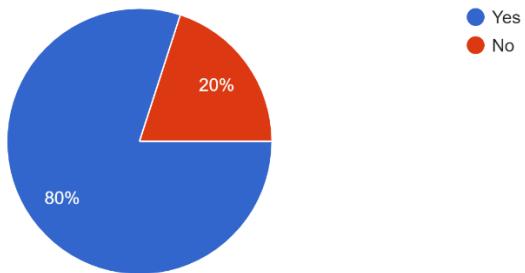
How many times do you want to restart a level before completing it?

10 responses



Should the levels contain checkpoints?

10 responses



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.

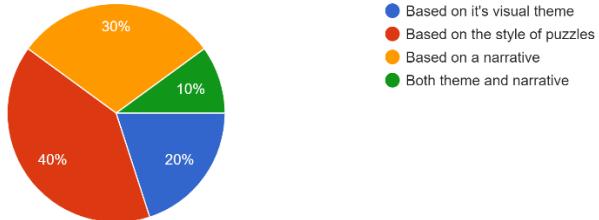
None of the users want to be stuck on a single level for more than about 20 mins on average and 5 minutes looks like a good balance to ensure the levels remain enjoyable for all, and no one gets frustrated, though some are more patient and will happily play a level for up to half an hour. To meet all 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.

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"> • Single player • Settings • Leaderboard • EXIT 	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 rd dimension would allow the puzzles to be much more intricate, with many more hidden features and more alternate solutions	I'm 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

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

Processor: dual core x86 64bit @ 1GHz or better	The game's code must be executed at a minimum rate to ensure it is fun to play
Memory: 2 GB ddr3	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
Graphics: 256mb video memory, capable of rendering at 640x480	The UI will depend on a minimum resolution to render properly and be readable, and this requires a minimum amount of video memory
Storage: 500MB available space	All the source code and assets use 500MB of free storage on the system
Peripherals: <ul style="list-style-type: none"> • 40% or more qwerty keyboard • 2 button mouse or equivalent 	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.
OS: 64 bit Microsoft Windows 10	Windows is a modern and common operating system providing the required execution environment for the rest of the dependencies
Python 3.10	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
Pygame 2.1.0	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"> • Tetris game research: all blocks are the same texture • Hardware limitations: reduces the number of textures loaded
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

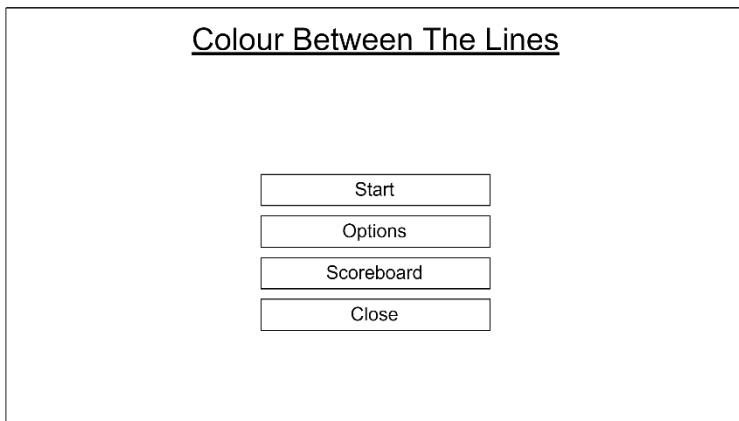
Most modern desktop games use the windows, icons, menus, and pointers (WIMPs) concepts for UI design. They are universal on the target platform; all computers with a mouse and keyboard have a menu system that makes use of these ideas. As well as its popularity and thus familiarity, I have also selected 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

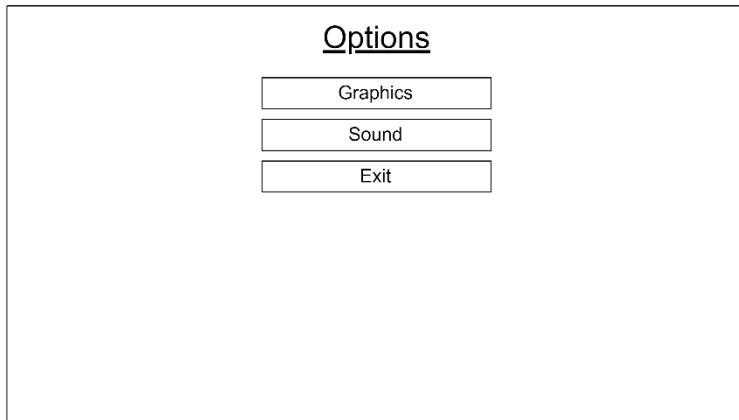
The layout of each screen is also critical, as the user will search for certain buttons in certain locations on the screen before looking around for them, and the faster they can find the button they are looking for, the easier the UI will be for them to use. For example, while looking for the exit button 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:



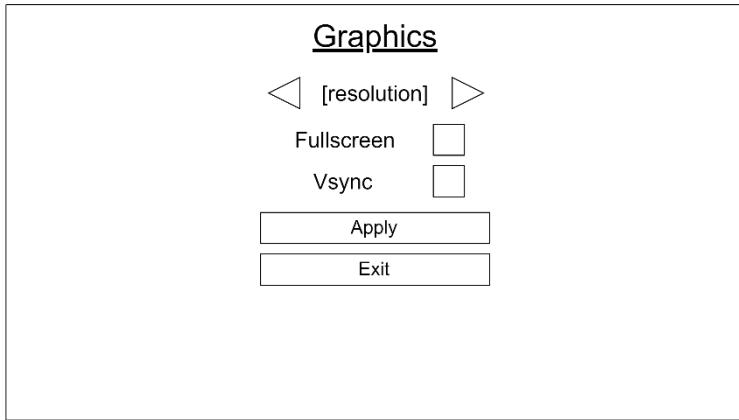
This is the screen that the game will start up to, and thus is the player's first impressions of the game. This screen will allow the user to navigate to all other parts of the game, so each button must be clear in what it will do and how it is a part of the flow of the gameplay. Thus, the buttons are 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:



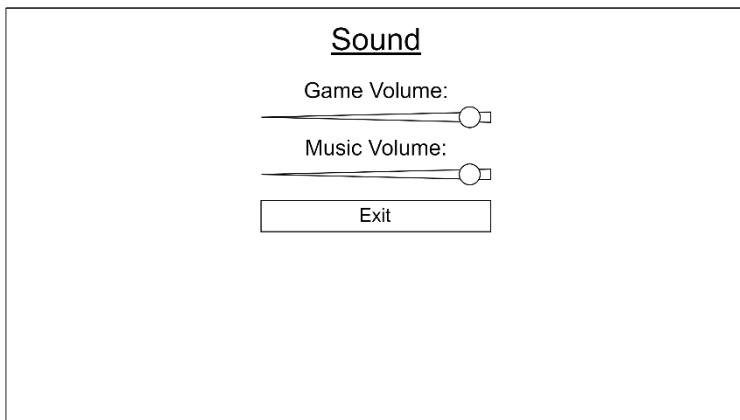
The options screen is what the user will use to navigate to other options screens. This screen will be accessed from multiple locations in the game, but then the exit button must cause the game to return to whichever screen led to the options screen in order to remain intuitive and predictable for the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

Graphics Options Screen:



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:



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.

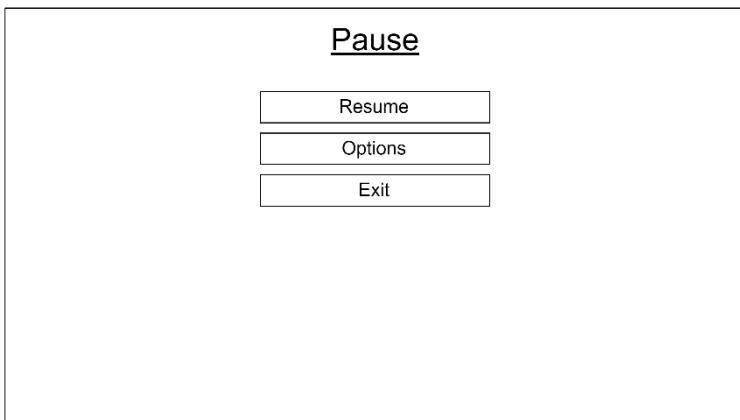
Scoreboard Screen:

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

At the bottom of the Scoreboard screen is a single rectangular button labeled 'Exit'.

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:



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.

Start Screen

Start Level

Maze Size:

Width: [width] X Height: [height]

Seed: [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!

[width] X [height] Put Your Name on the Scoreboard:

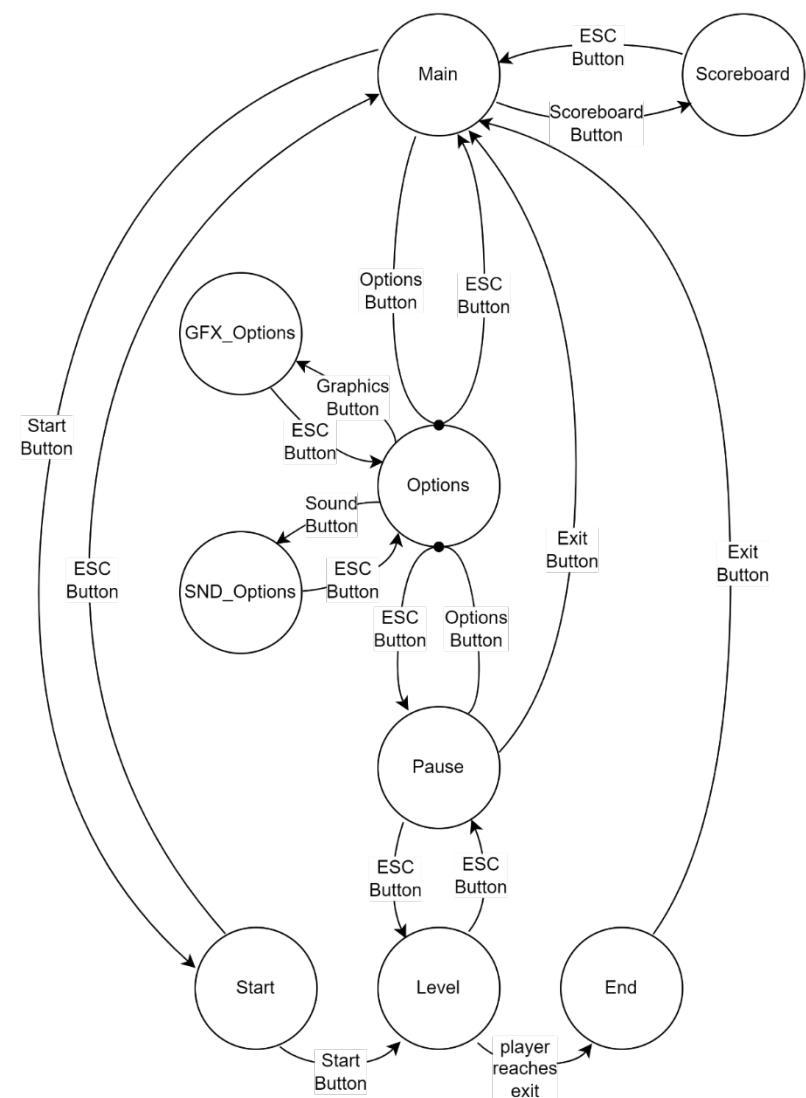
Seed: [Seed] [Player Name]

Time: [Time] Add Name to 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.



ASSET SELECTION

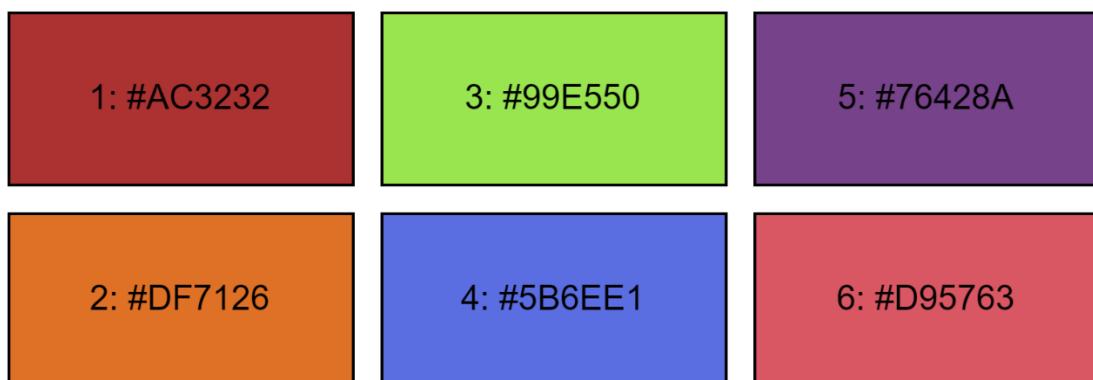
In the user base survey, the stakeholders made it clear that the game's visuals contribute significantly to the overall user experience and are thus a high priority. This means the assets must be carefully selected such that they have a cohesive visual style that aligns with the overall theme of the game. As I don't have time to develop assets myself, I must source them from the internet. Subsequently I will have to obey and licenses that are attributed to any assets I find, making sure to attribute the creators as they require. To ensure all assets I use have an agreeable license, they will all be sourced from opengameart.org as this website allow them to be filtered by license.

THEMING:

Due to the time I have for development, my game is quite limited in terms of complexity, meaning gameplay is about as complex as an advanced 8 bit game. 8 bit graphics therefore naturally fit with the rest of my game and give it a cohesive theme. This will mean that the individual pixels will be very apparent on screen, even more so when the sprites are scaled up such that only a small proportion of the maze fits on screen, so I must ensure that the images are upscaled such that the sharp edges of the pixels are maintained.

As colours are critical to the gameplay, the game will make use of a more diverse colour palette than 8 bit era games could. My game will make use of the standard 24bit rgb colour space as that is what is natively supported by pygame, but I will keep primary, game critical colours to a consistent colour palette so that the player can easily make connections between linked sprites, such as blocks and their corresponding gateways.

The colour palette I've selected is from the assets I have chosen, and so it will work naturally with them. This colour palette has a nice selection of different colours that are all distinct as to make it easy to discern between them, so the player can easily see which sprites in the game they can interact with and which ones they will pass straight through.



FONT:

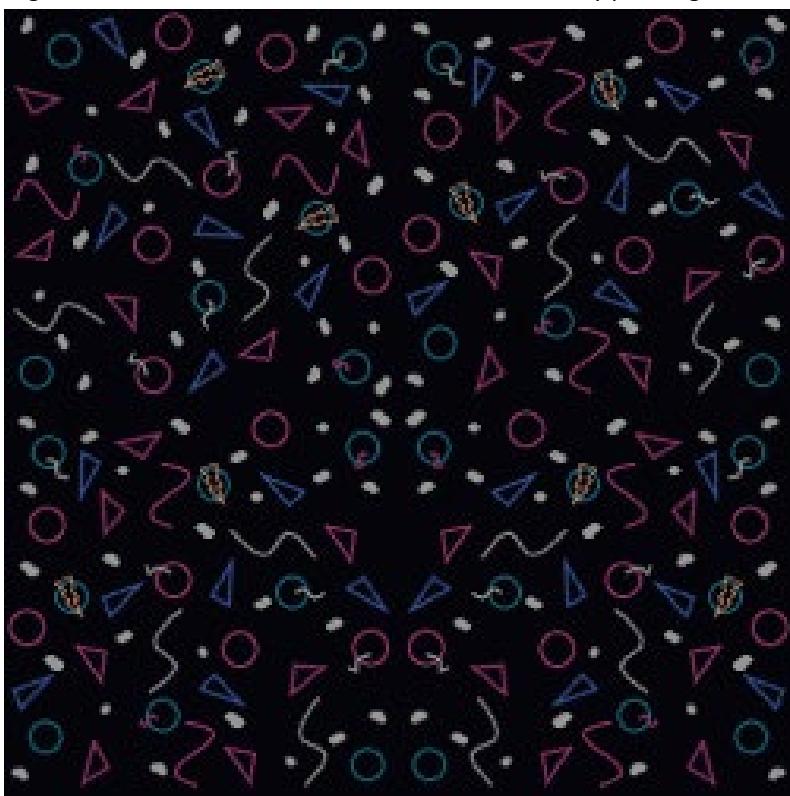
The font selected for the game is critical to how the Text elements of the UI integrate into the rest of the aesthetic. It must have an 8-bit style while still being very readable and simple – it must be sans-serif for simplicity. I have selected Pixeloid by GGBotNet as it meets these criteria and has a free license so I can use it in the game.



TILES:

The maze will be constructed out of many similar tile sprites, which tile together to create one large maze structure, so these tiles must have a very regular, square shape so that they can be automatically stitched together by the procedural maze generation without creating any seams which would break the immersion of the game. The majority of tiles must also maintain a dark colour palette as to meet the success criteria. There must be some tiles that are vibrant, primary colours to signal the critical sprites for the game mechanics.

In light of these requirements, I have selected Buch's [DawnBlockerOrtho](#) sprite sheet. It meets all the requirements, with many colour options for each tile, as well as having an orthographic pseudo 3d look that will make the game less flat and therefore much more aesthetically pleasing.



The floor of the maze is going to take up a large proportion of the screen at all points during gameplay, so is significant to the user's experience. To meet success criteria, it must be dark and not stand out too much. At the same time, a large slab of block colour will be unappealing, so it must have some visual interest while not sealing the user's focus. [Arcade Carpet Textures](#) from Luckius meets these requirements perfectly and aligns with the 8-bit aesthetic.

CHARACTER SPRITES:

The Player and enemy sprites are lot more dynamic than other sprites in the game, so they will be a lot more animated and thus need many more animation frames, so they have smooth movement. To indicate what is the current colour, the player needs to be wearing bright, primary colours in line with the colour palette. For the enemies, it needs to be clear that the player must avoid them; the success criteria therefore mandates that they are angular in appearance.



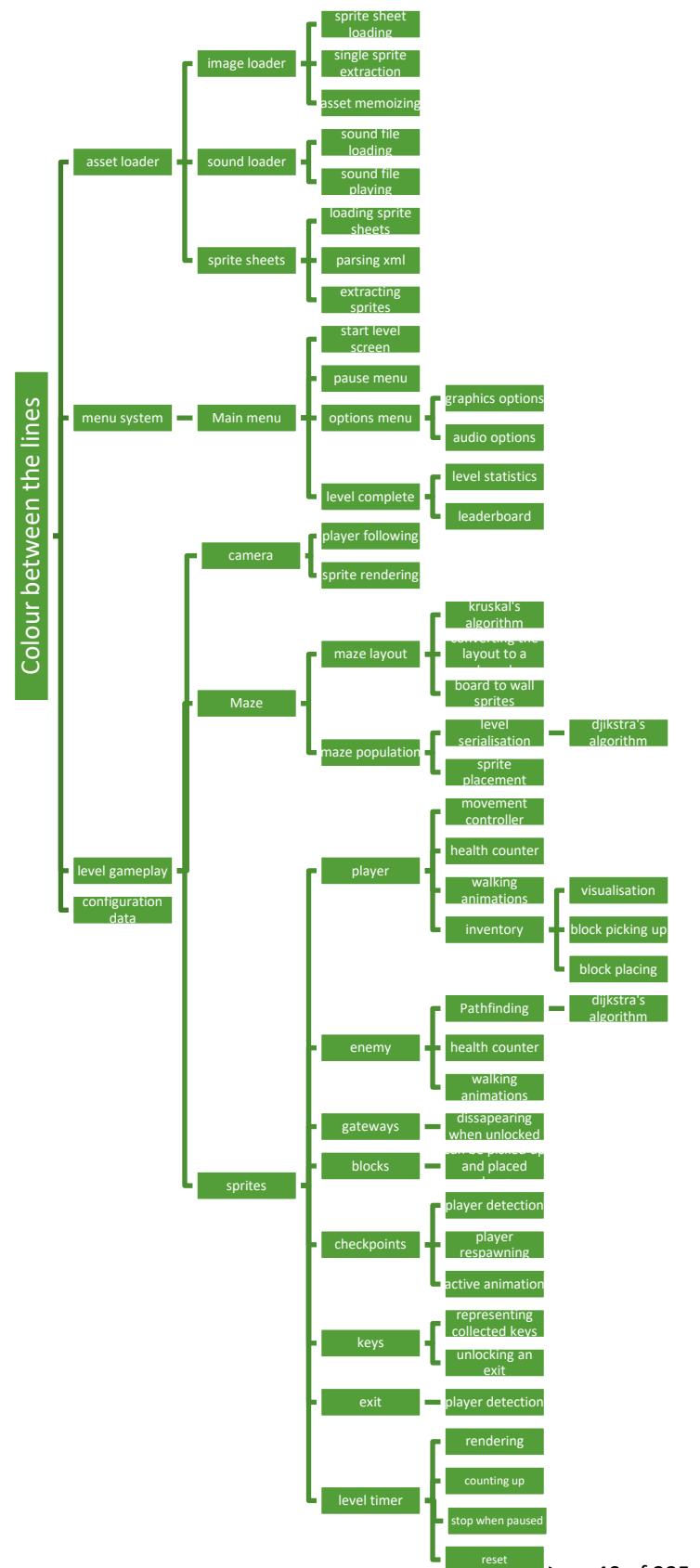
These requirements are met by GrafXKid's [Classic Hero and Baddies Pack](#) and devurandom's [RPG Character Sprites](#). They have more than ample different animation frames to make the characters' movement smooth.

While there are multiple player sprites, I will only select the first one and create multiple variants for all the colours in the colour palette. The reason why 2 different packs are necessary is because the first one's player, while it has plenty of frames, doesn't have forwards or backwards frames. This will interfere with rendering the blocks in the player's inventory on their backpack, so I have selected a different player sprite that won't cause this conflict.

There aren't satisfactory assets for all sprites in the game; the pre existing options don't align with the game's theme and don't visually instruct the player enough. Subsequently, I have adapted assets from other packs I have already selected, using photoshop to add the needed details which show how that sprite's game mechanics work. For example, with the exit, there are good options that clearly show it is the main objective and indicate that the player must collect the 6 keys needed to finish the level. I have created an asset based on one of the tiles from the tiles I will use to construct the maze; this means that it has the same dimensions and will thus tile correctly. I have coloured it gold to show it is a goal and put 6 key slots that contain the currently acquired keys; this shows the player how many keys they still need to collect before completing the maze.

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.

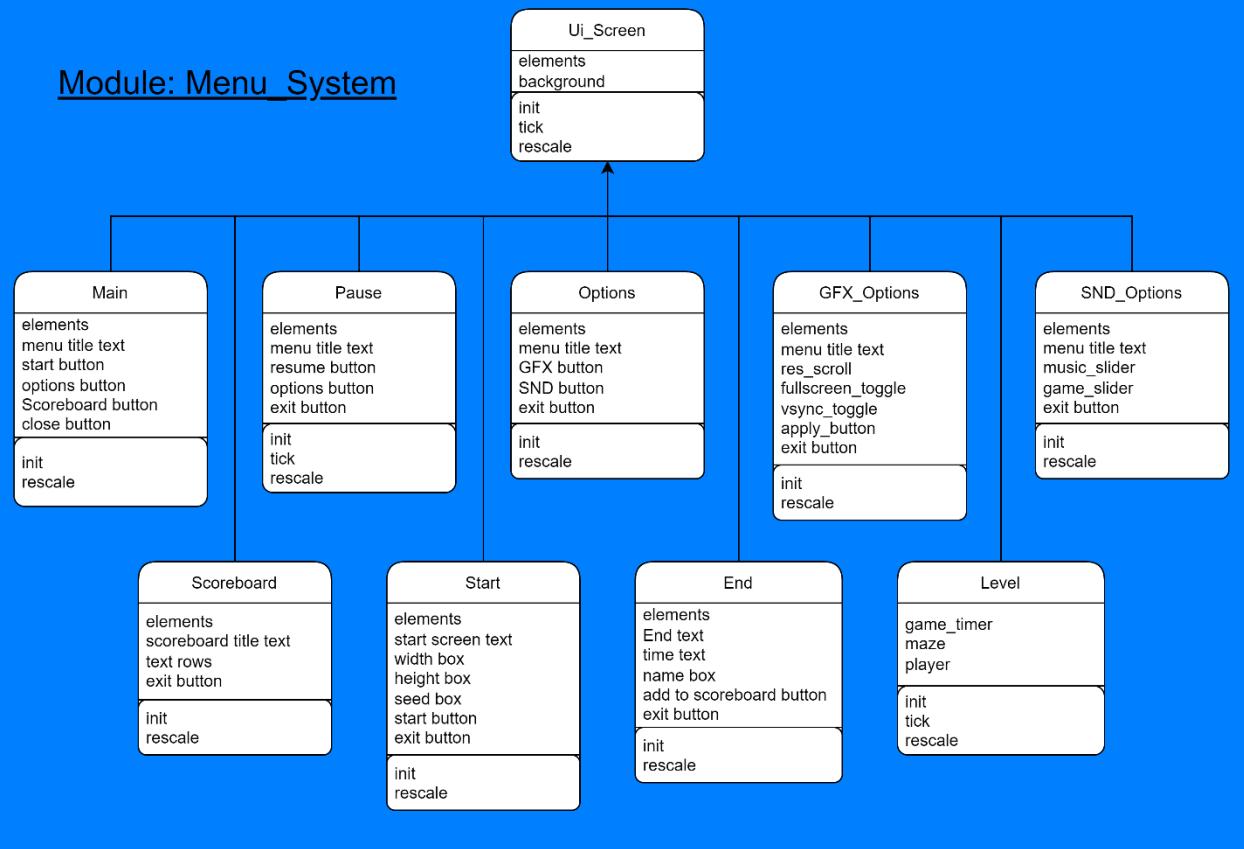
Each module will follow the same development methodology:

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.

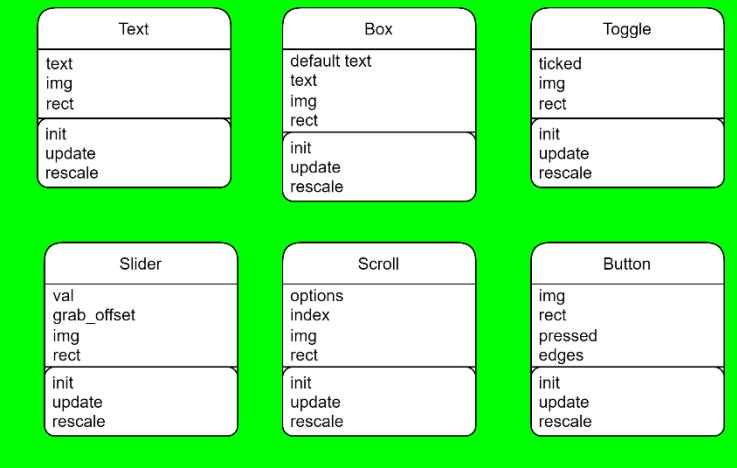
My game will make heavy use of object-oriented programming as this allows code and data to be collected and organized by overall function. As such, my design phase adheres to OOP based paradigms; each object is declared, defined, and assigned tests for each module one by one, as this reflects how it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

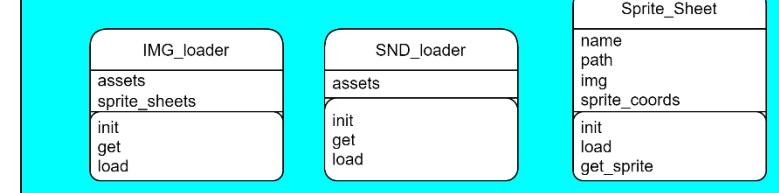
Module: Menu_System



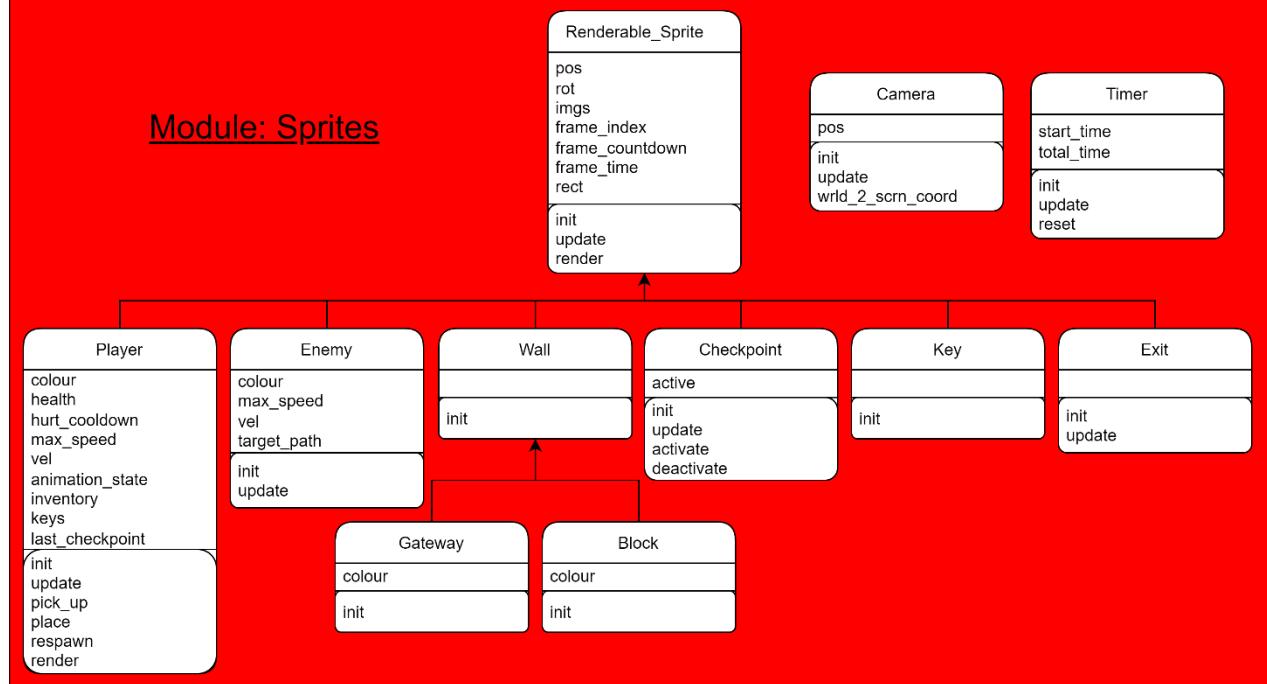
Module: Menu_Sprites



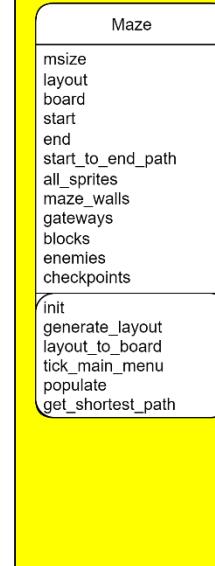
Module: Asset_Loaders



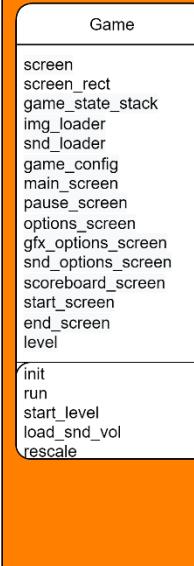
Module: Sprites



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 health 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 its 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 its class's attribute definitions with ones which have the values of its attribute's current values; this implements a persistent python file where a class's attributes can be stored 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
 - If image couldn't be found, return False

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 return a "no_sound.wav"
 - 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_path)
 - Stores sheet name as attribute
 - Load image from sheet_path + .jpg and store to attribute
 - Call load_xml
- VOID load_xml (xml_path):
 - Load xml sheet from _xml_path
 - Convert from xml to sprite_coords dictionary
 - Store sprite_coords dictionary as attribute
- Surface get(string sprite_name)
 - retrieve sprite_rect from sprite_coords
 - Store sprite_rect subsurface of img to sprite_img Surface
 - Return sprite_img Surface
 - If sprite wasn't found, return False

ASSET_LOADER – TESTING AND VERIFICATION

Test environment:

- File structure:
 - Root
 - img
 - Img1.png
 - Spritesheet1.png
 - Spritesheet1.xml
 - snd
 - Sound1.wav
 - no_sound.wav
 - config
 - Asset_Loader (file under test)
 - Host
 - Initializes pygame display system
 - Initialize config
 - 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)	Valid	Img1.png appears on the screen again

		again and blits it to 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, height 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.

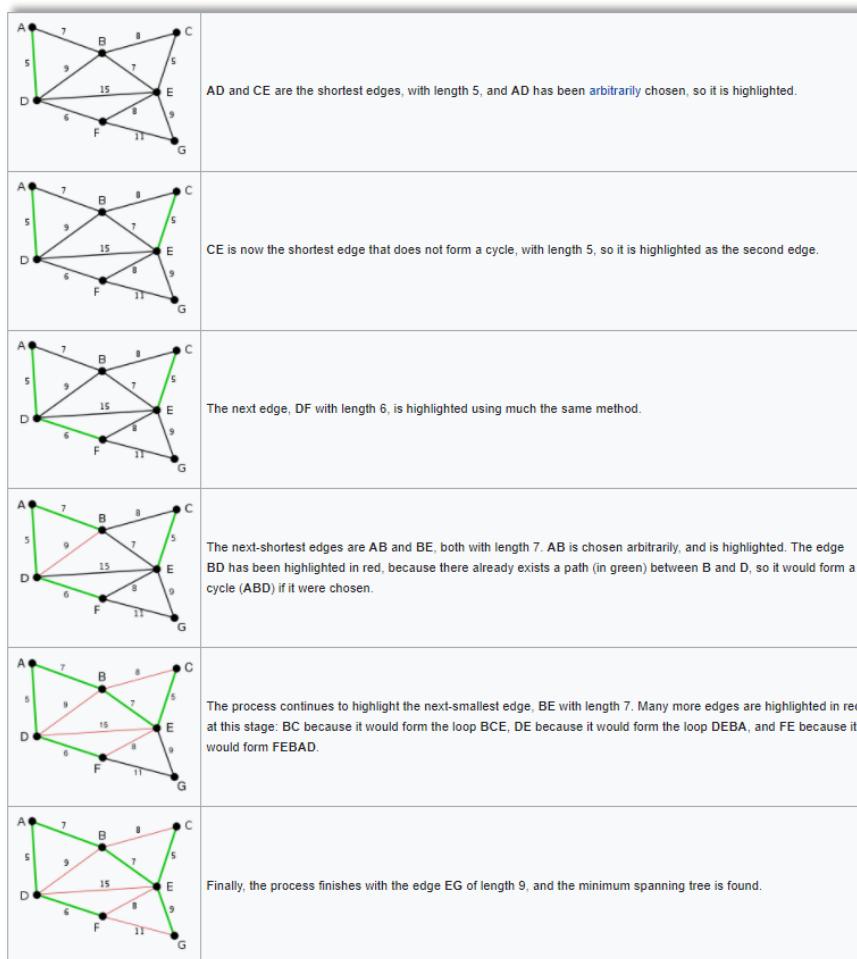


Figure 2: https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

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.

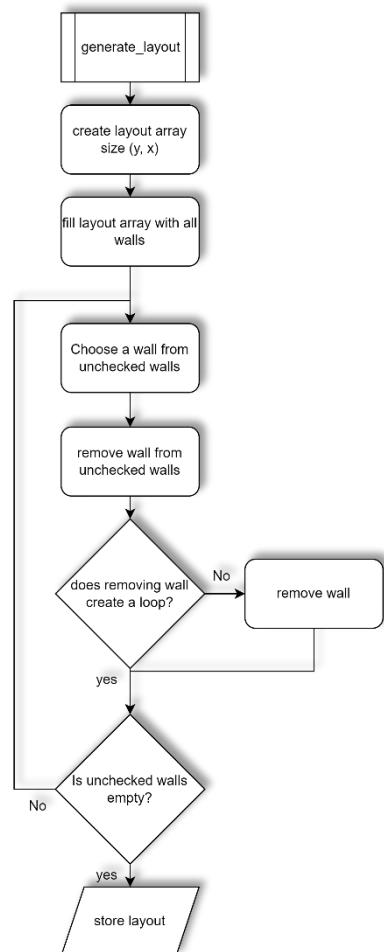


Figure 1: flow chart of Kruskal's algorithm

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 the 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 solvable 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.

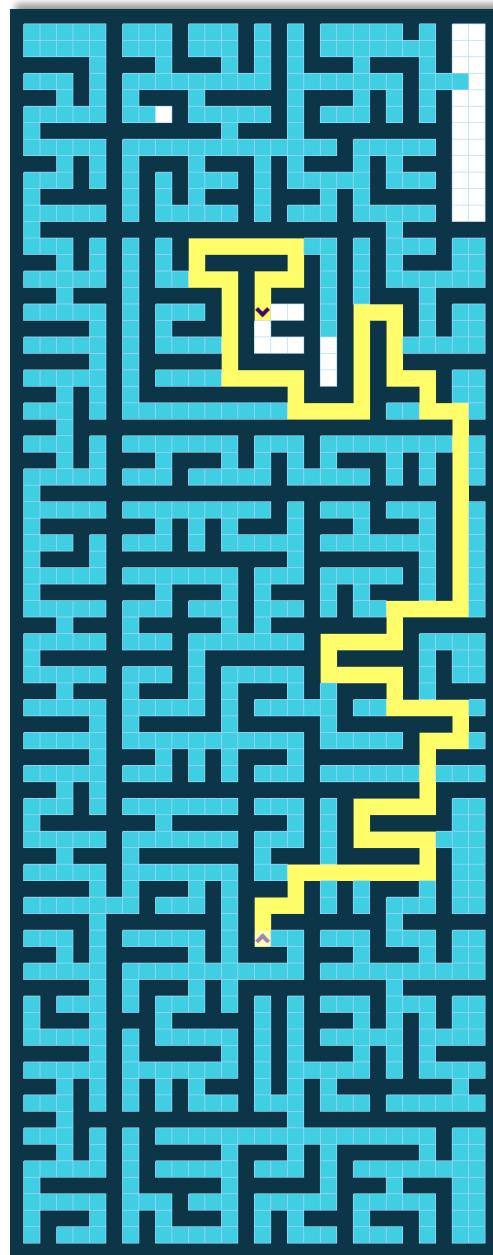


Figure 3: Visual demonstration of Dijkstra's path finding algorithm

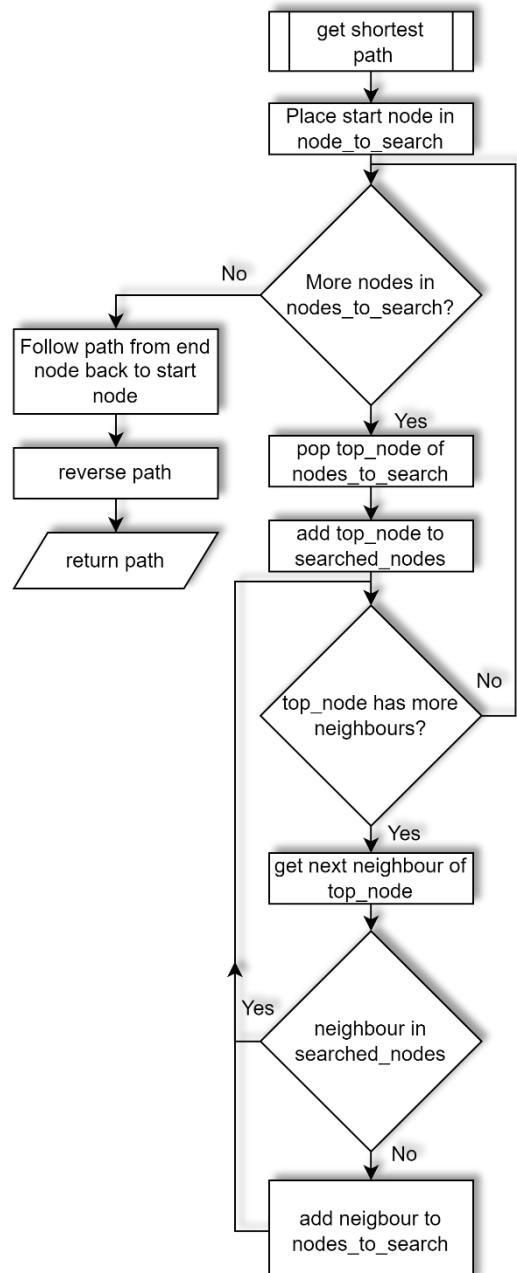
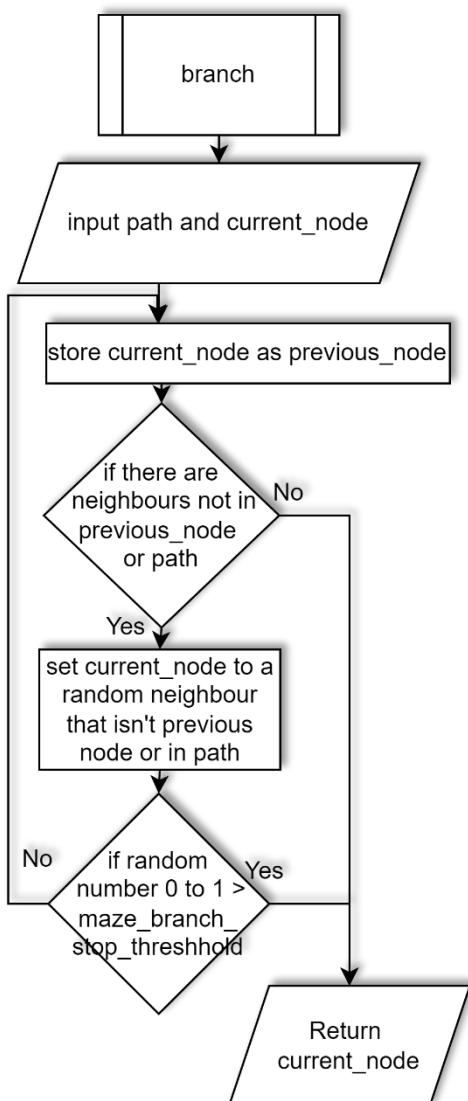


Figure 4: Flowchart of Dijkstra's path finding algorithm

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 the 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 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

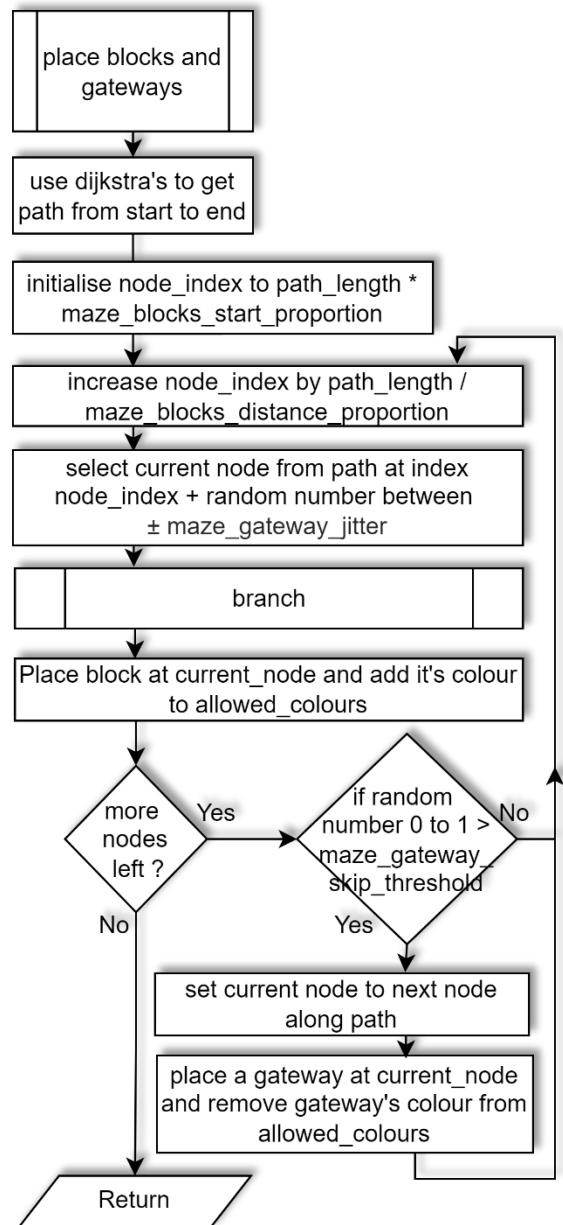
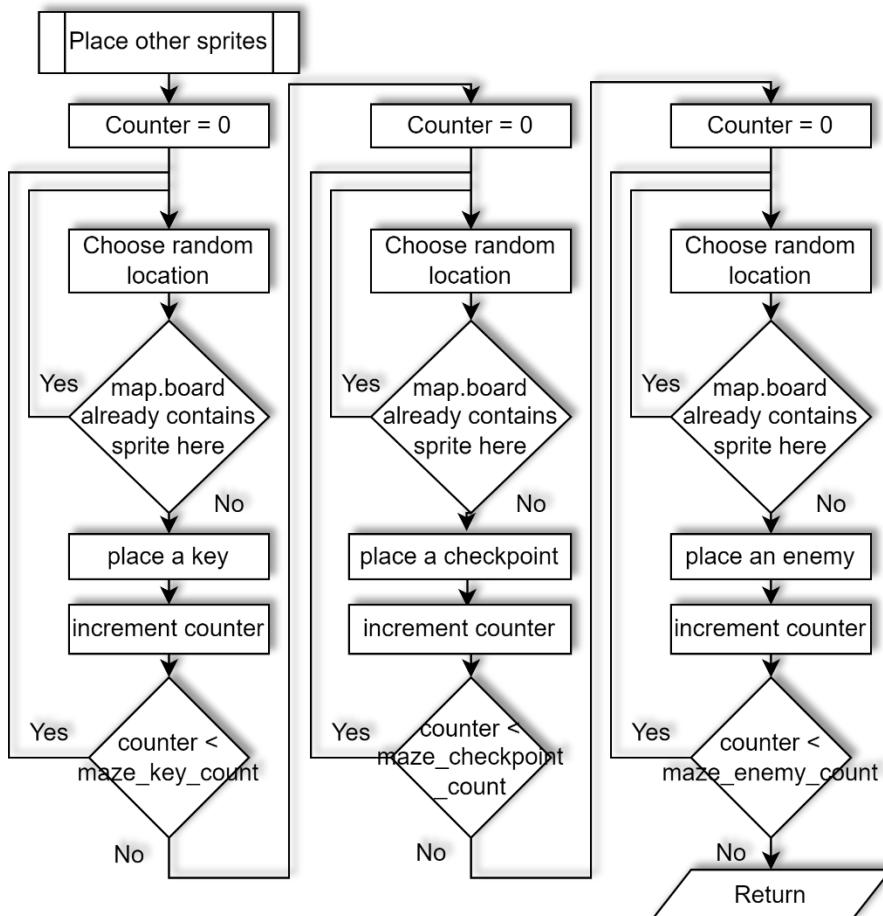


Figure 5: Flowchart for the block and gateway placement algorithm

Figure 6: Flowchart for the Branch 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
 - Maze_Gen (module under test)
 - Sprites_dummy

- 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 frame_countdown < 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:
 - 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)
 - Sprites Host
 - Initializes pygame
 - Initializes config
 - Initializes asset loaders
 - Generate a maze
 - Update all sprites
 - Render all sprites
 - Draw sprites
 - Maze Gen

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	Valid	Player passes through block

		equals that of the player		
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 it's 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 playining
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 its 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 off 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 its 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

MENU_SYSTEM – CLASS: PAUSE

- 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			

Candidate Name: Alexander Mills Candidate Number: 9120

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

DEVELOPMENT PLAN

Due to the game architecture’s modular nature, it will be developed in 8 separates stages:

1. **Config** module – the module responsible for game configuration data
2. **Asset_loader** module – the module responsible for loading all assets used by the game
3. **Main** module – the main entry point for the game that coordinates the rest of the code
4. **Maze_Gen** module – the module responsible for generating and storing mazes
5. **Sprites** module – the module responsible for all sprites in the level
6. **Menu_Sprites** module – the module responsible for all sprites that construct the menu screen
7. **Menu_System** module – the module responsible for managing the game’s menu system
8. **Integration** – wrapping everything together

The first 7 stages correspond to modules set out in the design stage. This allows each stage to be self contained and thus developed independent of the others, allowing me to focus on solving one problem at a time, rather than having constantly think about how each one will interact with the others while developing it. This approach accelerates not only the coding process, but also the testing, debugging and review. As each module is much smaller and simpler than the whole game, it has many less failure modes, each of which can be tested fully to ensure that it will meet its success criteria to create a final program in line with the design. Should a module fail a test, it is immediately apparent that the failure has been introduced during the development of that module only, drastically reducing the amount of code that has to be traversed, considered and reworked when fixing issues to achieve the desired behaviour. If the module serves to interact with the user in any way, I can then present a demo program centred around the module (the Testing Environment) to my stakeholders to collect their thoughts on it, and then I can make suitable adjustments based on their feedback, ensuring my game still meets the needs that are core to its development in the first place.

The 8th stage is Integration; this is where all 7 modules previously developed are integrated together in to the final game, which then be integration tested and presented to the stakeholders as a final product. Integration will involve collecting all the separate modules and linking them together, which will involve importing all the separate modules (without their Testing Environments) and then changing function calls in the Main module to reference each module’s functionality, tying it into the game, such that it works with all the other modules correctly. Once this is complete, the final game can be integration tested; this is where the final code is tested to ensure it meets all requirements of the success criteria fully and any incompatibilities between modules are identified and resolved before it is presented to the stakeholders for final review.

Each stage will follow the following steps:

- Declare development goals – what must be completed in this stage
- Development process– what code has been developed, how it was developed, what issues were encountered during development and how were they resolved
- Unit Testing – the module is put through the tests set out in the design section, identifying anything it failed on, and what changes to the design were needed to rectify the issues

- Review with stakeholders – if the module will interact with the user in any way, the stakeholders are shown the functionality developed in the module, and their feedback about it is collected. The module is then adjusted based on any constructive feedback given by the stakeholders, ensuring the functionality it implements aligns well with the users' needs.

STAGE 1: CONFIG MODULE

This module is for storing all game config data, such as asset directories, game balancing variables and other configuration data. It provides internal functionality, and thus will never be interacted with by the end user. The stakeholders will never interact with this code directly, so their input on it isn't needed; this module will be deemed fully functional and meeting success criteria if it fully passes all tests with no errors.

STAGE 1: DEVELOPMENT GOALS

- A single python file containing a single class whose objects are capable of:
 - Returning attributes
 - Storing attributes
 - Maintaining any changed attributes' states throughout the duration of the game
 - Maintaining any changed attributes' states while the game is closed
 - Maintaining any changed attributes' states while the computer is powered off
 - Having new attributes added to support future maintenance and development of the game

STAGE 1: DEVELOPMENT PROCESS

The first writeup:

```
import re

class Config():
    def __init__(self):
        # file paths: they are in reference to the game root folder
        self.img_path = img
        self.snd_path = other_snd_path
        self.music_path = snd/music
        self.scoreboard_path = scoreboard.csv

        # graphics config
        self.resolution = (640, 480)
        self.fullscreen = False
        self.vsync = True

        # volumes
```

```
self.game_vol = 1.0
self.music_vol = 0.3

# fonts
self.text_colour = (123, 45, 67)
self.text_font_name = Pixeloid

# walking sprites
self.player_hurt_cooldown = 500
self.player_max_health = 100
self.player_max_speed = 30
self.player_acc = 2
self.enemy_speed = 10

# maze generation
self.maze_blocks_start_proportion = 0.0833333333333333
self.maze_blocks_distance_proportion = 0.1666666666666666
self.maze_gateway_jitter = 6
self.maze_gateway_skip_threshold = 0.3
self.maze_branch_stop_threshold = 0.1
self.maze_key_count = 6
self.maze_checkpoint_count = 10
self.maze_enemy_count = 4

# self modifying code: save the attributes by rewriting this file
def save(self):
    self_file = open(__file__, "r")
    self_file_str = self_file.read()
    self_file.close()

    for identifier, val in self.__dict__.items():
        self_file_str = re.sub(f" self.{identifier} = .+\n",
                              f" self.{identifier} = {val}\n",
                              self_file_str)

    self_file = open(__file__, "w")
    self_file.write(self_file_str)
    self_file.close()
```

This code is mostly just defining the variables declared in the config module's design phase, so it is just a list of attribute assignments. For readability, they are separated by category, with each category clearly marked, making this code easier to navigate and maintain

The most complicated part of this code to figure out was the save method; it must do 3 separate things: read and write from files, access all of the object's attributes and replace certain parts of the file string. Reading and writing is handled by opening the file: “`__file__`” ; this is a pre-defined global variable for the path of this python file. Each object in python has a “`__dict__`” attribute, which is a dictionary of all attribute identifiers and values. I can iterate through this dictionary and use regular expressions to search for and replace (“`re.sub`”) the line of text which defines this attribute, thus replacing it with the current value. The file string can then be written back to the file.

STAGE 1: UNIT TESTING

Test host program:

- Loads and instantiates config
- Loads values from config
- Saves values to config
- Loads them again to check if they were saved correctly

```
# load config
import config


# prints all attributes of a config object
def print_config_vals(cfg):
    for id, val in game_config.__dict__.items():
        print(f"id:{str(id):<40}, " +
              f"val:{str(val):<30}, " +
              f"type:{str(type(val)):<15}")


# init config
game_config = config.Config()

# load and print all values from config
print_config_vals(game_config)

# make changes to values
game_config.snd_path = "other_snd_path"
game_config.player_max_speed = 30
game_config.vsync = True
game_config.music_vol = 0.3
game_config.text_colour = (123, 45, 67)

# save changes
game_config.save()

# reload config
del config
import config

# reinit config
game_config = config.Config()
```

```
# print all values from config
print("\n\n\nconfig reloaded: ")
print_config_vals(game_config)
```

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior	Resultant behaviour	Pass / Fail + resolution
1	Test loading string attributes	Host loads img_path and prints it	Valid	Outputs the img path saved there	<code>id:img_path , val:img , type:<class 'str'></code>	Pass
2	Test loading int attributes	Host loads player_max_health And print it	Valid	Outputs the player_max_health value	<code>id:player_max_health , val:100 , type:<class 'int'></code>	Pass
3	Test loading bool attributes	Host loads fullscreen and prints it	Valid	Outputs the fullscreen value	<code>id:fullscreen , val:False , type:<class 'bool'></code>	Pass
4	Test loading float attributes	Host loads game_vol and prints it	Valid	Outputs the game_vol value	<code>id:game_vol , val:1.0 , type:<class 'float'></code>	Pass
5	Test loading array attributes	Host loads resolution and prints it	Valid	Outputs the resolution stored there	<code>id:resolution , val:(640, 480) , type:<class 'tuple'></code>	Pass
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	<code>self.snd_path = other_snd_path</code>	<p>Fail: Strings are not being formatted correctly when rewriting to the config file:</p> <pre>for identifier, val in self.__dict__.items(): self_file_str = re.sub(f" self.{identifier} = .+\n", f" self.{identifier} = {val}\n", self_file_str)</pre> <p>solution: adjust the formatting code such that it will include parentheses correctly:</p> <pre>for identifier, val in self.__dict__.items(): self_file_str = re.sub(f" self.{identifier} = .+\n", f" self.{identifier} = {repr(val)}\n", self_file_str)</pre>
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	<code>self.player_max_speed = 30</code>	Pass
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	<code>self.vsync = True</code>	Pass
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	<code>self.music_vol = 0.3</code>	Pass
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	<code>self.text_colour = (123, 45, 67)</code>	Pass

11	Test reloading string attributes	Host reloads snd_path and prints it	Valid	Outputs the snd_path saved there	<code>id: snd_path , val: other_snd_path , type:<class 'str'></code>	Pass
12	Test reloading int attributes	Host reloads player_max_speed And print it	Valid	Outputs the player_max_speed value	<code>id: player_max_speed , val: 30 , type:<class 'int'></code>	Pass
13	Test reloading bool attributes	Host reloads vsync and prints it	Valid	Outputs the vsync value	<code>id: vsync , val: True , type:<class 'bool'></code>	Pass
14	Test reloading float attributes	Host reloads music_vol and prints it	Valid	Outputs the music_vol value	<code>id: music_vol , val: 0.3 , type:<class 'float'></code>	Pass
15	Test reloading array attributes	Host reloads text_colour and prints it	Valid	Outputs the text_colour stored there	<code>id: text_colour , val: (123, 45, 67) , type:<class 'tuple'></code>	Pass

STAKEHOLDER REVIEW

This is an internal module, so the stakeholders will never see it therefore their input isn't needed to determine if this module meets its success criteria.

As this module has now passed all tests set out for it in the design section, it now meets its success criteria, and is thus considered complete and can be set aside until integration.

STAGE 2 : ASSET_LOADER

This module is responsible for loading and caching all assets that are used within the game. It provides a class with functionality for loading and storing the sprites in the game, which will be stored on disk as both in discrete images and in sprite sheets. It must locate the requested resource and load it into a pygame surface and return that. This module also provides a class capable of loading sound files and returning them as pygame sound objects.

STAGE 2: DEVELOPMENT GOALS

Image loader:

- Can load single sprite images from the image directory folder
- Can load a sprite sheet from the image directory folder, along with its xml representation
- Given the name of any asset, it can locate whether it is stored on a sprite sheet or a single file, and can then return a surface containing only the requested image
- Given the name of an asset that doesn't exist
- When an asset is requested for a second time, no files are accessed, and instead the cached images are returned

Sound loader:

- Loads single sound files from the sound directory folder
- Given the name of any sound asset, it can locate the asset and then load and return it as a pygame sound object
- Given the name of a sound asset that doesn't exist, return a default sound

STAGE 2: DEVELOPMENT PROCESS

First draft:

```
import pygame as pg
from pathlib import Path
import xml.etree.ElementTree as ET

class Img_Loader():
    def __init__(self, game):
        self.game = game
        self.assets = {}
        self.sprite_sheets = []

    # load sprite sheets
```

Path is used for file path handling and manipulation

ElementTree is used for parsing the xml files used to store sprite coords on sprite sheets

```
img_path = Path(self.game.config.img_path)
for file_path in img_path.glob("*"):
    file_name = file_path.as_posix()
    if file_name.endswith(".xml"):
        # create a sprite sheet object for each xml file in img_path
        self.sprite_sheets.append(Sprite_Sheet(file_name[:-4]))
```

```
def get(self, img_name):
    image = False
    # check already loaded assets for the image
    if img_name in self.assets.keys():
        return self.assets[img_name]
```

```
# check for the image in the image folder
elif (loaded_img := self.load(img_name)):
    image = loaded_img
```

```
# try to find sprite in spritesheets
else:
    for sheet in self.sprite_sheets:
        if loaded_image := sheet.get(img_name):
            # when we find an image, stop looking
            image = loaded_image
            break
```

```
# sprite cant be found
if not(image):
    image = pg.surface.Surface((100, 100)).convert_alpha()
    image.fill((255, 0, 255))
```

```
# cache and return image
self.assets[img_name] = image
return image
```

```
def load(self, img_name):
    # search img_path for images
    img_path = Path(self.game.config.img_path)
    # iterate through files in img_path
    for file_path in img_path.glob("*"):
        if file_path.name.startswith(img_name):
            # if the names match, load image and return it
            image = pg.image.load(file_path.as_posix()).convert_alpha()
            return image
```

Checks if the image is cached and returns that first; doesn't need to proceed to the end of the function as it doesn't need to be cached again

2 cases for trying to load image from file:

- Image found; and returned; this is a Boolean true, so is assigned to image
- Image not found; False returned, nothing assigned to image

Check all spritesheets for the image in a similar way to checking for images from file

Glob is an iterator for all files in the img path directory

Convert_alpha allows the image to have transparency render correctly

```
# no image was found
return False

class Snd_Loader():
    def __init__(self, game):
        self.game = game
        self.assets = {}

    def get(self, snd_name):
        # check in assets
        if snd_name in self.assets.keys():
            return self.assets[snd_name]

        # try to load sound
        if loaded_sound := self.load(snd_name):
            self.assets[snd_name] = loaded_sound
            return loaded_sound

        # failed to load sound, return generic sound
        else:
            no_sound_path = Path(__file__).parent() / "no_sound.wav"
            return pg.mixer.Sound(no_sound_path.as_posix())

    def load(self, snd_name):
        snd_path = Path(self.game.config.snd_path)
        # iterate through all files in snd_path
        for file_path in snd_path.glob("*"):
            if file_path.name.startswith(snd_name):
                # return the sound if it has the correct name
                return pg.mixer.Sound(file_path.as_posix())

class Sprite_Sheet():
    def __init__(self, game, sheet_path):
        self.game = game
        self.sheet_path = sheet_path
        self.sprite_coords = {}

        # ensure img and xml are loadable
        img_path = Path(sheet_path + ".png")
        xml_path = Path(sheet_path + ".xml")
        if img_path.is_file() and xml_path.is_file():
            # load image
```

Similar loading process to that of images, but there are no sprite sheets to check

Before trying to load the spritesheet, the requisite files are checked for. If they aren't found, this sprite sheet ends up empty, and thus will just return False to all get requests

```
        self.img = pg.image.load(img_path.as_posix()).convert_alpha()

        # load xml
        self.load_xml(xml_path)

def load_xml(self, xml_path):
    # load xml tree
    xml_tree_root = ET.parse(xml_path.as_posix()).getroot()
    # iterate through entries in the xml file
    for entry in xml_tree_root:
        # extract name and rect
        attributes = entry.attrib
        name = attributes["name"]
        rect = [int(attributes[i]) for i in ["x", "y", "width", "height"]]
        # store to sprite_coords
        self.sprite_coords[name] = rect

def get(self, sprite_name):
    # check the this sheet has this sprite
    if sprite_name in self.sprite_coords.keys():
        # find the rect of the requested sprite
        rect = self.sprite_coords[sprite_name]
        # gain the sprite surface
        image = self.img.subsurface(rect)
        image_copy = pg.surface.Surface(image.get_size())
        image_copy.blit(image, (0,0))
        return image_copy

    # no sprite found
    return False
```

The subsurface command takes a slice of the surface but doesn't copy it. As surfaces are mutable, this will cause problems, so it is copied into a new surface and that is returned.

STAGE 2: UNIT TESTING

The test Host program:

- Initialises config
- Initialises loaders, passing them the game with the config
- Load image assets and blit them to the screen
- Attempt to load image assets that don't exist
- Load sound assets and start playing them
- Attempt to load sound assets that don't exist

Test Host program code:

```
import pygame as pg
import config
import Asset_Loader as AL

class Game():
    def __init__(self):
        # initialise pygame
        pg.init()
        self.screen = pg.display.set_mode((1000, 800))

        # init config
        self.config = config.Config()

        # init loaders
        self.img_loader = AL.Img_Loader(self)
        self.snd_loader = AL.Snd_Loader(self)

        # fill screen with dark green
        self.screen.fill((0, 128, 0))

        # load and render img1.png
        img1 = self.img_loader.get("img1.png")
        new_size = [i*8 for i in img1.get_size()]
        self.screen.blit(pg.transform.scale(img1, new_size), (10,10))

        # load "block light blue"
        sprite1 = self.img_loader.get("block light blue")
        new_size = [i*8 for i in sprite1.get_size()]
        self.screen.blit(pg.transform.scale(sprite1, new_size), (150,10))

        # re load img1.png
        img1 = self.img_loader.get("img1.png")
        new_size = [i*8 for i in img1.get_size()]
        self.screen.blit(pg.transform.scale(img1, new_size), (300,10))

        # load img2.png
        # re load img1.png
        img1 = self.img_loader.get("img2.png")
        new_size = [128,128]
        self.screen.blit(pg.transform.scale(img1, new_size), (450,10))

        # flip so that sprites are rendered to screen
        pg.display.flip()
```

```
# load sound1.wav
snd1 = self.snd_loader.get("sound1.wav")
snd1.play(10)

# delay between sounds
pg.time.delay(round(snd1.get_length()*1000 *10))

# load sound2.wav
snd1 = self.snd_loader.get("sound2.wav")
snd1.play(10)

# wait for user input to close
input()

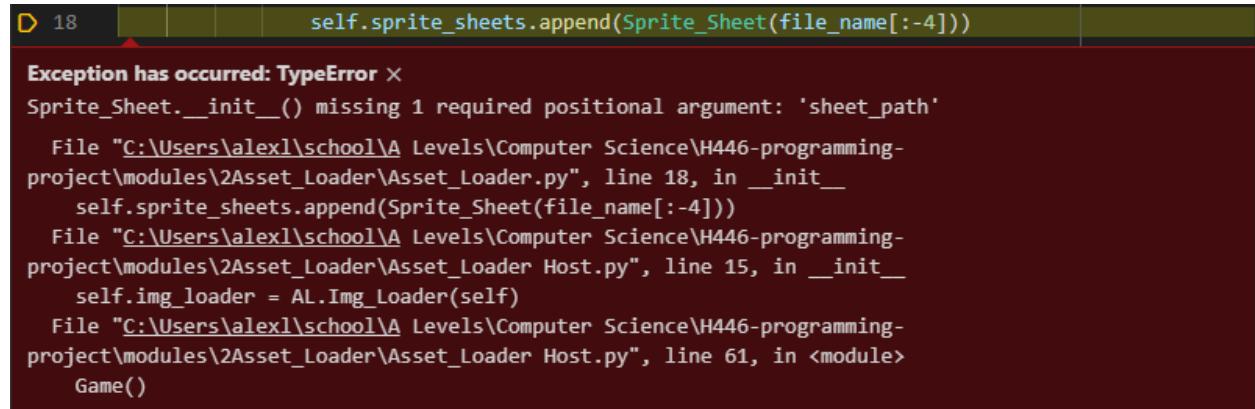
pg.quit()
```

Game()

Resolving Errors that prevent the code from being tested:

Before I can start testing the Asset loader module, I need to get the module to a state where it actually runs; this means resolving syntax errors and other simple programming errors

Error 1:



```
D 18     self.sprite_sheets.append(Sprite_Sheet(file_name[:-4]))  
  
Exception has occurred: TypeError ×  
Sprite_Sheet.__init__() missing 1 required positional argument: 'sheet_path'  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\2Asset_Loader\Asset_Loader.py", line 18, in __init__  
    self.sprite_sheets.append(Sprite_Sheet(file_name[:-4]))  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\2Asset_Loader\Asset_Loader Host.py", line 15, in __init__  
    self.img_loader = AL.Img_Loader(self)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\2Asset_Loader\Asset_Loader Host.py", line 61, in <module>  
    Game()
```

Solution:

I wasn't passing in the game argument that the sprite sheet objects need so they can access the game's config, so I've added it so that is passed in now:

```
# create a sprite sheet object for each xml file in img_path
self.sprite_sheets.append(Sprite_Sheet(game, file_name[:-4]))
```

Error 2:

```
106     def load_xml(self, xml_path):  
107         # load xml tree  
108         xml_tree_root = ET.parse(xml_path.as_posix()).getroot()  
  
Exception has occurred: ParseError x  
no element found: line 226, column 14  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
project\modules\2Asset_Loader\Asset_Loader.py", line 108, in load_xml  
      xml_tree_root = ET.parse(xml_path.as_posix()).getroot()  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
project\modules\2Asset_Loader\Asset_Loader.py", line 104, in __init__  
      self.load_xml(xml_path)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
project\modules\2Asset_Loader\Asset_Loader.py", line 18, in __init__  
      self.sprite_sheets.append(Sprite_Sheet(game, file_name[:-4]))  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
project\modules\2Asset_Loader\Asset_Loader Host.py", line 15, in __init__  
      self.img_loader = AL.Img_Loader(self)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
project\modules\2Asset_Loader\Asset_Loader Host.py", line 61, in <module>  
    Game()
```

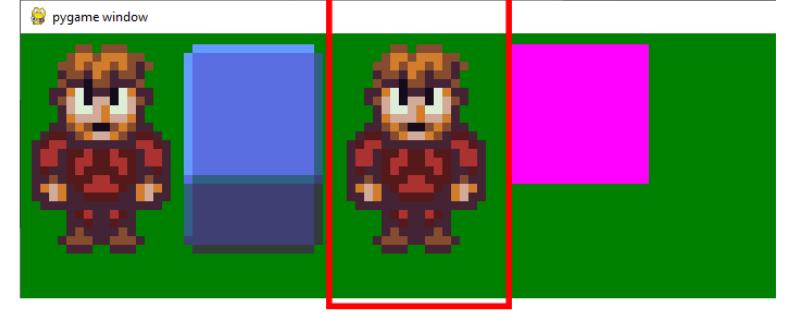
Problem: The sprite sheet's xml file wasn't formatted correctly; a closing tag had mistakenly written as an opening tag:

```
<Subtexture name="stairB dark grey" x="187" y="435" width="16" height="24" />  
<Subtexture name="stairU dark grey" x="204" y="435" width="16" height="24" />  
<Subtexture name="rampB dark grey" x="221" y="435" width="16" height="24" />  
<Subtexture name="rampU dark grey" x="238" y="435" width="16" height="24" />  
<TextureAtlas>
```

Solution: fix the closing tag

```
<Subtexture name="stairU dark grey" x="204" y="435" width="16" height="24" />  
<Subtexture name="rampB dark grey" x="221" y="435" width="16" height="24" />  
<Subtexture name="rampU dark grey" x="238" y="435" width="16" height="24" />  
</TextureAtlas>
```

Now that the code can start and runs without instantly crashing, it can be tested on the tests set out be the design stage:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior	Observed behavior	Pass / Fail + Solution
1	basic sprite loading	Host calls get(img1.png) and blits it to screen	Valid	Img1.png appears on screen	 img1.png has successfully loaded onto screen	Pass
2	loading sprites from a sprite sheet	Host calls get(sprite1)	Valid	Sprite 1 from Spritesheet1.png appears on screen	 Sprite from the sprite sheet appears on screen	Pass
3	sprite caching	Host calls get(img1.png) again and blits it to a different location on the screen	Valid	Img1.png appears on the screen again	 img1.png has been successfully loaded a second time	Pass
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	 Purple square (or arguably pink) appears on screen, and game doesn't crash	Pass
5	music loading	Host calls get(sound1.wav) and plays it	Valid	The sound can be heard playing	 sound1.wav can be heard playing	Pass

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	 	game has crashed and no sound can be heard playing	Fail: Problem1: can't locate no_sound.wav as it isn't checking in the snd folder for it Problem2: calling the parent path, but it isn't callable <pre># failed to load sound, return generic sound else: no_sound_path = Path(__file__).parent() / "no_sound.wav" return pg.mixer.Sound(no_sound_path.as_posix())</pre> Solution1: load no_sound.wav from the snd folder Solution2: don't call the parent path <pre># failed to load sound, return generic sound else: no_sound_path = Path(__file__).parent / "snd" / "no_sound.wav" return pg.mixer.Sound(no_sound_path.as_posix())</pre>
---	---------------------------------------	---	---------	---	---	--	--

STAKEHOLDER REVIEW

This is an internal module, so the stakeholders will never see it therefore their input isn't needed to determine if this module meets its success criteria.

As this module has now passed all tests set out for it in the design section, it now meets its success criteria, and is thus considered complete and can be set aside until integration.

STAGE 3: MAIN MODULE

This module hosts all gameplay and is the main entry point for the game. This module contains the main game object and is thus responsible for initialising the game, it's loaders and config and then running the main game loop. The main game loop must select the correct tick function to update, which will update and render the screen the game should currently be displaying.

STAGE 3: DEVELOPMENT GOALS

- Start pygame environment
- Initialise config and loader modules
- Implement game state stack
- The correct tick function is called for the current item in the game state stack
- When the game window is rescaled, everything else is also rescaled

STAGE 3: DEVELOPMENT PROCESS

First draft:

```
import pygame as pg
import config as cfg
import Asset_Loader as AL
import Menu_System_dummy as MS

class Game():
    def __init__(self):
        self.game_state_stack = []

        # init pygame env
        pg.init()
        pg.mixer.init()

        # init asset loaders and config
        self.config = cfg.Config()
        self.img_loader = AL.Img_Loader(self)
        self.snd_loader = AL.Snd_Loader(self)

        # init video
        self.rescale()

        # init screens other than level
        self.main_screen = MS.Main(self)
        self.pause_screen = MS.Pause(self)
```

```
self.options_screen = MS.Options(self)
self.gfx_options_screen = MS.GFX_Options(self)
self.snd_options_screen = MS.SND_Options(self)
self.scoreboard_screen = MS.Scoreboard(self)
self.start_screen = MS.Start(self)
self.end_screen = MS.End(self)
self.level = False

# load and play background music
self.music = self.snd_loader.get("music.wav")
self.load_snd_vol()
self.music.play(loops = -1)

# push main menu onto game state stack
self.game_state_stack.append(self.main_screen.tick)

# call run
self.run()

def run(self):
    # main loop
    clock = pg.time.Clock()
    while len(self.game_state_stack) > 0:
        # delay to achieve correct frame rate
        if self.config.vsync:
            # calculate dt
            dt = clock.tick(60)
        else:
            dt = clock.tick()

        # event collect events
        event_list = list(pg.event.get())

        # check events
        for event in event_list:
            # close event: close the game
            if event.type == pg.QUIT:
                return

            # rescale events: change size of the screen
            elif event.type == pg.VIDEORESIZE:
                if self.config.rescaleable:
                    self.config.resolution = event.size
                    self.rescale()

    # update game state
    self.game_state_stack[-1].update(dt)
```

Level screen is set to false until it is loaded; this makes it easy to check if it is loaded before trying to use it

Music loops is set to -1 to enable continuous looping

Clock.tick does 2 functions: return time since the last loop and delay the code such that the requested fps is achieved

Event list is explicitly converted into a list to avoid generator exhaustion

If the close button is pressed, the main loop halts and the game closes

```
# call correct tick function
self.game_state_stack[:-1](event_list, dt)

def start_level(self, size, seed):
    # initialise new level
    self.level = MS.Level(self, size, seed)

    # push tick function to game state stack
    self.game_state_stack.append(self.level.tick)

def load_snd_vol(self):
    # change music volume
    self.music.set_volume(self.config.music_vol)

def rescale(self):
    # rescale screen
    if self.config.rescaleable:
        self.screen = pg.display.set_mode(self.config.resolution,
                                         pg.RESIZABLE)
    else:
        self.screen = pg.display.set_mode(self.config.resolution)

    # call rescale method of all screens
    self.main_screen.rescale()
    self.pause_screen.rescale()
    self.options_screen.rescale()
    self.gfx_options_screen.rescale()
    self.snd_options_screen.rescale()
    self.scoreboard_screen.rescale()
    self.start_screen.rescale()
    self.end_screen.rescale()
    if self.level:
        self.level.rescale()

Game()
```

List Index -1 means the last element
in the list (top of the stack)

The level screen is only rescaled if it
already exists

STAGE 3: UNIT TESTING

Test Host Environment:

The file under test is the main entry point for the game, and as such can't have a test host that calls its methods, as it is the module that initiates the majority of method calls. Instead, this module has a host environment comprised of stripped down equivalents to the modules containing the methods that will be called when the game is running. These simple equivalents have almost no functionality, and simply print that they have been called. They also implement the needed functionality to enable testing: it is the job of the screens to request main to navigate around them, so a simple way of doing this has been implemented; all screens delay a little, then return back to the main screen, which can be commanded from the command line to navigate to any screen.

Test Host Environment Code: Menu_System_dummy.py

```
import pygame as pg

class dummy_parent_screen():
    def __init__(self, game):
        self.game = game

    def tick(self, event_list, dt):
        print(f"{self.__class__.__name__} screen")
        pg.time.delay(1000)
        self.game.game_state_stack.pop(-1)

    def rescale(self):
        print(f"{self.__class__.__name__} rescaled")

class Main(dummy_parent_screen):
    def __init__(self, game):
        self.game = game

    def tick(self, event_list, dt):
        match input("next screen: ").split(" "):
            case ["start"]:
                s = self.game.start_screen.tick
                self.game.game_state_stack.append(s)
            case ["level"]:
                s = self.game.level.tick
                self.game.game_state_stack.append(s)
            case ["end"]:
                s = self.game.end_screen.tick
                self.game.game_state_stack.append(s)
            case ["scoreboard"]:
                s = self.game.scoreboard_screen.tick
                self.game.game_state_stack.append(s)
            case ["pause"]:
                s = self.game.pause_screen.tick
```

```
        self.game.game_state_stack.append(s)
    case ["options"]:
        s = self.game.options_screen.tick
        self.game.game_state_stack.append(s)
    case ["gfx"]:
        s = self.game.gfx_options_screen.tick
        self.game.game_state_stack.append(s)
    case ["snd"]:
        s = self.game.snd_options_screen.tick
        self.game.game_state_stack.append(s)
    case ["level_int", w, h, seed]:
        self.game.start_level((w,h), seed)
    case _:
        print("pattern not recognised")

class Pause(dummy_parent_screen):
    pass

class Options(dummy_parent_screen):
    pass

class GFX_Options(dummy_parent_screen):
    pass

class SND_Options(dummy_parent_screen):
    pass

class Scoreboard(dummy_parent_screen):
    pass

class Start(dummy_parent_screen):
    pass

class End(dummy_parent_screen):
    pass

class Level(dummy_parent_screen):
    def __init__(self, game, size, seed):
        super().__init__(game)
        print(f"level, size:{size}, seed:{seed} created")
```

Resolving Errors that prevent the code from being tested:

Before I can start testing the Main module, I need to get the module to a state where it actually runs; this means resolving syntax errors and other simple programming errors

Error 1:

```
D 96 |     self.main_screen.rescale()

Exception has occurred: AttributeError
'Game' object has no attribute 'main_screen'

  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\3Main>Main.py", line 96, in rescale
    self.main_screen.rescale()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\3Main>Main.py", line 20, in __init__
    self.rescale()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\3Main>Main.py", line 107, in <module>
    Game()
```

Problem 1:

The rescale method has been reused to initialise the screen, but this means that it is also trying to rescale all screens before they have been initialised.

```
# init video
self.rescale()

# init screens other than level
self.main_screen = MS.Main(self)
```

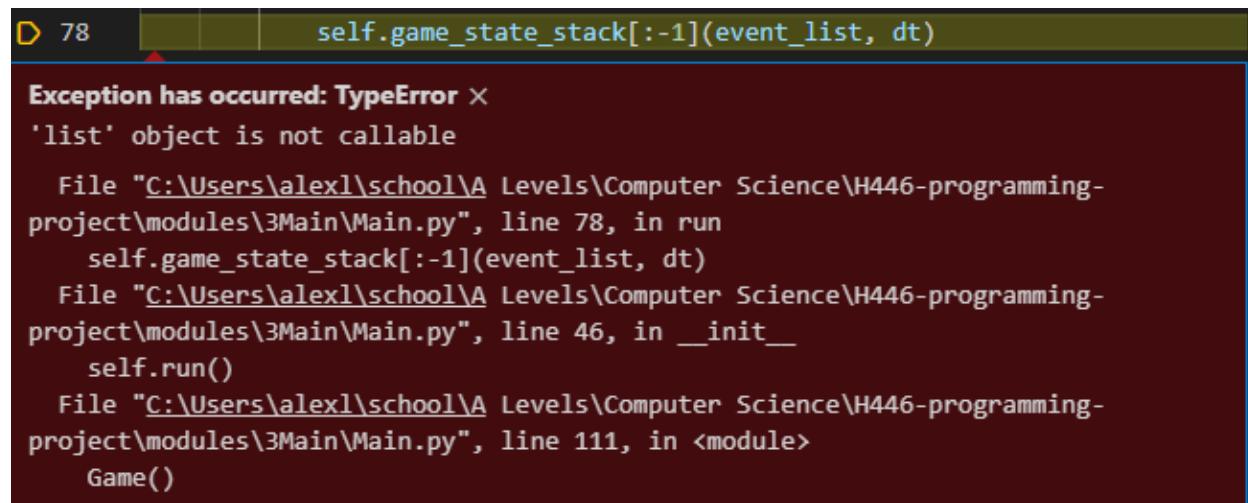
Solution 1:

Don't reuse rescaling code for initialising the screen:

```
# init video
if self.config.rescaleable:
    self.screen = pg.display.set_mode(self.config.resolution,
                                      pg.RESIZABLE)
else:
    self.screen = pg.display.set_mode(self.config.resolution)

# init screens other than level
self.main_screen = MS.Main(self)
```

Error 2:



D 78 self.game_state_stack[:-1](event_list, dt)

Exception has occurred: TypeError ×
'list' object is not callable

File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\3Main>Main.py", line 78, in run
 self.game_state_stack[:-1](event_list, dt)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\3Main>Main.py", line 46, in __init__
 self.run()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\3Main>Main.py", line 111, in <module>
 Game()

Problem 2:

`[:-1]` is used to create a list slice from the start to the end (the entire string), which still returns a list

Solution 2:

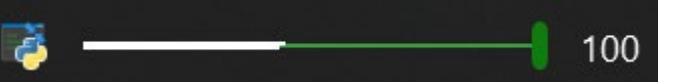
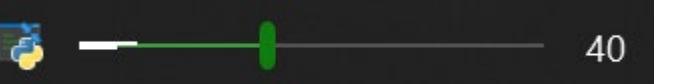
Used `[-1]` instead, which returns the item at the final index of the list

Now that the code can start and runs without instantly crashing, it can be tested on the tests set out be the design stage

Candidate Name: Alexander Mills Candidate Number: 9120

Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior	Resultant Behavior	Pass / Fail + Solution
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	pygame 2.1.2 (SDL 2.0.18, Python 3.10.1) Hello from the pygame community. https://www.pygame.org/contribute.html next screen: next screen: start Start screen	Pass
2	start screen execution	Input: start	normal	Console output saying it is on start screen	next screen: level Level screen	Pass
3	level screen execution	Input: level	normal	Console output saying it is on level screen	next screen: end End screen	Pass
4	end screen execution	Input: end	normal	Console output saying it is on end screen	next screen: scoreboard Scoreboard screen	Pass
5	scoreboard screen execution	Input: scoreboard	normal	Console output saying it is on scoreboard screen	next screen: pause Pause screen	Pass
6	pause screen execution	Input: pause	normal	Console output saying it is on pause screen	next screen: options Options screen	Pass
7	options screen execution	Input: options	normal	Console output saying it is on options screen	next screen: gfx GFX_Options screen	Pass
8	graphics options screen execution	Input: gfx	normal	Console output saying it is on graphics options screen	next screen: snd SND_Options screen	Pass
9	sound options screen execution	Input: snd	normal	Console output saying it is on sound options screen	next screen: level_init 30 50 12314515 level, size:(30, 50), seed:12314515 created Level screen	Pass
10	Level screen initialisation	Input:level init 30 50 12314515	normal	Console output saying level has been created with width 30, height 50 and seed 12314515 Console output saying it is on level screen	next screen: level_init -50 12314515 level, size:(-50, 12314515) created Level screen	Pass
11	Level screen initialisation	Input:level_init 30 -50 12314515	invalid	Console output saying failed to create level: invalid input	Fail: validation missing <code>def start_level(self, size, seed): # initialise new level self.level = MS.Level(self, size, seed)</code> Solution: implement validation <code>def start_level(self, size, seed): # validate if size[0] <= 0 or size[1] <= 0: print(f"invalid level size: {size}") return # initialise new level self.level = MS.Level(self, size, seed)</code>	
12	Music player	Start main file	normal	Music starts playing when file is loaded	 40 no sound.wav starts playing	Fail: music.wav is missing <code>self.music = self.snd_loader.get("music.wav")</code> Solution: add music file and change the loader request to match it: <code>self.music = self.snd_loader.get("Vexento - Lotus.wav")</code>

13	Music volume control	Music volume in config set to 0 then Input:snd_vol	Normal	Sound volume is set to mute	 14		Pass
14	Music volume control	Music volume in config set to 1 then Input:snd_vol	Normal	Sound volume is set to full	 100		Pass
15	Music volume control	Music volume in config set to -1 then Input:snd_vol	invalid	Sound volume is not set	 40		Pass
16	Rescaling	Input: rescale config.resolution = 640 * 480	Normal	Display surface changes size Console output from all screens saying they rescaled	Main rescaled Pause rescaled Options rescaled GFX_Options rescaled SND_Options rescaled Scoreboard rescaled Start rescaled End rescaled		Pass

STAGE 3: STAKEHOLDER REVIEW

This is an internal module, so the stakeholders will never see it therefore their input isn't needed to determine if this module meets its success criteria.

As this module has now passed all tests set out for it in the design section, it now meets its success criteria, and is thus considered complete and can be set aside until integration.

STAGE 4: MAZE GEN

This module is responsible for generating the maze the player will navigate. It generates a solvable maze layout, converts that to a format that can be rendered on screen and generates sprites to populate the maze, all while ensuring the level always remains solvable in all possible cases. This module also provides functionality for other sprites to navigate around the maze using Dijkstra's path finding algorithm.

STAGE 4: DEVELOPMENT GOALS:

- Declare a maze object which can be constructed given the maze size and seed
- Generate a layout of the correct dimensions that with walls in an array such that it represents a solvable maze with a maximised number of walls
- Convert the array of walls into a 2d array (board) which shows what there will be wall sprites and where there won't be
- Generate wall sprites from the board array
- Define a start and end for the maze
- Calculate the path through the maze
- Populate the maze with gateways and blocks such that it is still solvable
- Populate the maze with other sprites: the exit, checkpoints, keys, and enemies

STAGE 4: DEVELOPMENT PROCESS

First draft:

```
import pygame as pg
import random as rng
import Sprites_dummy as Sprites

class Maze():
    def __init__(self, game, msize, seed):
        # store maze size and seed
        self.game = game
```

```
self.msize = msize
self.seed = seed

# initialise sprite groups
self.all_sprites = pg.sprite.Group()
self.maze_walls = pg.sprite.Group()
self.gateway = pg.sprite.Group()
self.blocks = pg.sprite.Group()
self.enemies = pg.sprite.Group()
self.checkpoints = pg.sprite.Group()
self.keys = pg.sprite.Group()

# initialise RNG
rng.seed(self.seed)

# generate maze layout
self.generate_layout()

# converts board coords to pixel coords
self.pos_convert=lambda pos: [pos[0] * self.game.config.walls_width_px,
                               pos[1] * self.game.config.walls_height_px]

# convert layout to wall sprites
wall_gen = lambda pos: Sprites.Wall(self.game, self.pos_convert(pos))
self.layout_to_board(wall_gen)

# find start to end path
self.start_to_end_path = self.get_shortest_path(self.start, self.end)

# populate
self.populate()

def generate_layout(self):
    """generates a maze layout"""
    # uses kruskal's algorithm to generate maze layout
    # init layout array
    layout = [list([True, True, x + y * self.msize[0]])
              for x in range(0, self.msize[0]))
              for y in range(0, self.msize[1])]

    # generate list of all unchecked walls
    unchecked_walls = []
    # - 1 stops it from checking bottom most walls
    for y in range(0, self.msize[1] - 1):
```

These sprite groups store all the sprites generated by the maze, and allow the sprites easily be interacted with depending on type

Ensures identical random mazes can be generated

This anonymous function converts Board coordinates to the much larger world coordinates:

If the code were to check all walls, then it would cause index errors as it would try to check zones of the bottom of the maze

```

# - 1 stops it from checking right most walls
for x in range(0, self.msize[0] - 1):
    unchecked_walls.append([x,y,0])
    unchecked_walls.append([x,y,1])

# iterate over all walls randomly, removing them if possible
while len(unchecked_walls) > 0:
    # select random wall
    wall = rng.choice(unchecked_walls)
    x = wall[0]
    y = wall[1]
    if wall[2]: # is left right wall
        zone1 = layout[y][x][2]
        zone2 = layout[y][x+1][2]

        # check if this wall merges zones
        if zone1 != zone2:
            # delete this wall
            layout[y][x][1] = False
            layout[y][x+1][2] = zone1

        else: # is up down wall
            zone1 = layout[y][x][2]
            zone2 = layout[y+1][x][2]

            # check if this wall merges zones
            if zone1 != zone2:
                # delete this wall
                layout[y][x][0] = False
                layout[y+1][x][2] = zone1

        # remove wall from unchecked walls
        unchecked_walls.remove(wall)

    # store layout to attribute
    self.layout = layout

def layout_to_board(self, wall_gen):
    """converts the maze layout to a board and sprites"""
    # adjust wall gen to append created walls to the correct groups
    def wall_gen_group(start_pos):
        wall = wall_gen(start_pos)
        self.all_sprites.add(wall)
        self.maze_walls.add(wall)

```

Check if the wall separates 2 different zones; if it does, then remove it

This locally defined function; it runs the wall_gen function to create a wall, but then appends this wall to sprite groups before returning it

```

        return wall

# generate board array
# initialise blank array
bsize = [2 * self.msize[i] + 1 for i in (0,1)]
board = [list(False)
          for x in range(0, bsize[0])]
          for y in range(0, bsize[1])]

# place perimeter sprites on board
for x in range(0, bsize[0]): # top and bottom
    board[0][x] = wall_gen_group((x, 0))
    board[bsize[1]][x] = wall_gen_group((x, 1))

for y in range(1, bsize[1]-2): # side edges
    board[y][0] = wall_gen_group((0, y))
    board[y][bsize[0]] = wall_gen_group((bsize[0], y))
    board[bsize[1]-1][0] = wall_gen_group((0, bsize[1]-1))

# place corner sprites on board
for y in range(2, bsize[1], 2):
    for x in range(2, bsize[0], 2):
        board[y][x] = wall_gen_group((x,y))

# place edge sprites on board
for ly in range(self.msize[1]-1):
    for lx in range(self.msize[0]-1):
        by = 2*ly + 1
        bx = 2*lx + 1

        if self.layout[ly][lx][0]: # wall below
            board[by+1][bx] = wall_gen_group((bx, by+1))
        if self.layout[ly][lx][1]: # wall right
            board[by][bx+1] = wall_gen_group((bx, by+1))

self.board = board

# generate start
self.start = [1, 1]

# generate end
self.end = [bsize[0]-1, bsize[1]]
self.exit = Sprites.Exit(self.pos_convert(self.end))

```

The perimeter of the maze is made first, in 2 passes: a horizontal pass and then a vertical pass. There is a gap left at the bottom right of the maze for the exit to be placed

the walls that are at the corners of each junction are placed first as they don't change depending on layout

The walls that form the maze are placed next, filling in the gaps between the corners

The colour for the next block is selected from a list of colour indices that haven't already been used

```
def populate(self):
    """populates the maze with sprites"""
    # populate gateways and blocks
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)

    path_len = len(self.start_to_end_path)
    remaining_colours = [i for i in range(6)] # colours not used so far
    allowed_colours = [] # colours of blocks with no gateway

    node_index = path_len * self.game.config.maze_blocks_start_proportion

    while node_index < path_len and len(remaining_colours) > 0:
        # select current node from path
        node_index += int(rng.random() * \
                          self.game.config.maze_gateway_jitter)
        current_node = self.start_to_end_path[node_index]
        next_node = self.start_to_end_path[node_index + 1]

        # branch
        branch_node = self.branch(current_node, [next_node])

        # place block at branch_node
        block_colour = rng.choice(remaining_colours)
        remaining_colours.remove(block_colour)
        allowed_colours.append(block_colour)

        block = Sprites.Block(branch_node, block_colour)
        self.all_sprites.add(block)
        self.blocks.add(block)

        # conditionally set gateway to next node along path
        if rng.random() < self.game.config.maze_gateway_skip_threshold:
            gateway_colour = rng.choice(allowed_colours)
            allowed_colours.remove(gateway_colour)

            gateway = Sprites.Gateway(next_node)
            self.all_sprites.add(gateway)
            self.gateways.add(gateway)

        # increase node_index
        node_index += path_len * \
                      self.game.config.maze_blocks_distance_proportion
```

Gateways are placed if the value is less than the threshold

The place blocks and gateways algorithm from the design section

```
# populate keys
for _ in range(self.game.config.maze_key_count):
    pos = self.random_board_spot()
    while self.board[pos[1]][pos[0]] != False:
        pos = self.random_board_spot()

    key = Sprites.Key(pos)
    self.all_sprites.add(key)
    self.keys.add(key)

# populate checkpoints
for _ in range(self.game.config.maze_checkpoint_count):
    pos = self.random_board_spot()
    while self.board[pos[1]][pos[0]] != False:
        pos = self.random_board_spot()

    checkpoint = Sprites.Checkpoint(pos)
    self.all_sprites.add(checkpoint)
    self.checkpoints.add(checkpoint)

# populate enemies
for _ in range(self.game.config.maze_enemy_count):
    pos = self.random_board_spot()
    while self.board[pos[1]][pos[0]] != False:
        pos = self.random_board_spot()

    enemy = Sprites.Checkpoint(pos)
    self.all_sprites.add(enemy)
    self.enemies.add(enemy)

def get_shortest_path(self, start, end):
    """returns (list) path from start to end"""
    # dijkstra's algorithm

    # place start node in nodes to search
    nodes_to_search = [start]
    known_nodes = {start: False}

    while len(nodes_to_search) > 0:
        current_node_pos = nodes_to_search.pop(0)

        for offset in [(0,-1), (0,1), (1,0), (-1,0)]:
            neighbour = [current_node_pos[i] + offset[i] for i in (0,1)]
            # check neighbour is a wall
```

Random position is selected, and if it is already populated choose another one until it isn't populated

Standard Dijkstra's implementation using the board to get adjacencies

```

        if self.board[neighbour[1]][neighbour[0]] != False:
            continue

        # check neighbour has already been searched
        if neighbour in known_nodes.keys():
            continue

        # new node; add to known nodes and append to nodes_to_search
        known_nodes[neighbour] = current_node_pos
        nodes_to_search.append(neighbour)

    # use known nodes to construct a path from end to start
    end_to_start = []
    current_node = end
    while known_nodes[current_node] != False:
        end_to_start.append(current_node)
        current_node = known_nodes[current_node]

    # reverse end_to_start to get start_to_end
    return end_to_start[::-1]

def branch(self, start_node, known_nodes):
    """branches out from a start node to another node in the maze"""
    nodes_to_search = [start_node]

    while len(nodes_to_search) > 0 and \
        rng.random() < self.game.config.maze_branch_stop_threshold:

        current_node_pos = nodes_to_search.pop(0)

        for offset in [(0,-1), (0,1), (1,0), (-1,0)]:
            neighbour = [current_node_pos[i] + offset[i] for i in (0,1)]
            # check neighbour is a wall
            if self.board[neighbour[1]][neighbour[0]] != False:
                continue

            # check neighbour has already been searched
            if neighbour in known_nodes:
                continue

            # new node; add to known nodes and append to nodes_to_search
            known_nodes.append(current_node_pos)
            nodes_to_search.append(neighbour)

```

The branch algorithm in practice is very similar to Dijkstra's, but instead has a possibility of randomly halting, and it doesn't keep track of the path

```
        return current_node_pos

def random_board_spot(self):
    """returns (tuple) random point on the board"""
    return [rng.randint(0, self.msize[i]*2) for i in [0,1]]
```

Returns a random point on the board by simple generating a random x and random y

STAGE 4: UNIT TESTING

Issue Resolution:

Error 1:

```
D 32 | wall_gen = lambda pos: Sprites.Wall(self.game, self.pos_convert(pos))

Exception has occurred: TypeError
Wall.__init__() takes 2 positional arguments but 3 were given
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 32, in <lambda>
    wall_gen = lambda pos: Sprites.Wall(self.game, self.pos_convert(pos))
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 94, in wall_gen_group
    wall = wall_gen(start_pos)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 108, in layout_to_board
    board[0][x] = wall_gen_group((x, 0))
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 33, in __init__
    self.layout_to_board(wall_gen)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__
    self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>
    Game()
```

Problem 1:

The dummy sprites didn't have an argument in their constructor for the main game object:

```
class Parent(pg.sprite.Sprite):
    def __init__(self, start_pos):
        super().__init__()

        self.image = pg.surface.Surface((16,24))
        self.image.fill(self.colour)
        self.rect = self.image.get_rect()
        self.rect.topleft = start_pos
```

Solution 1:

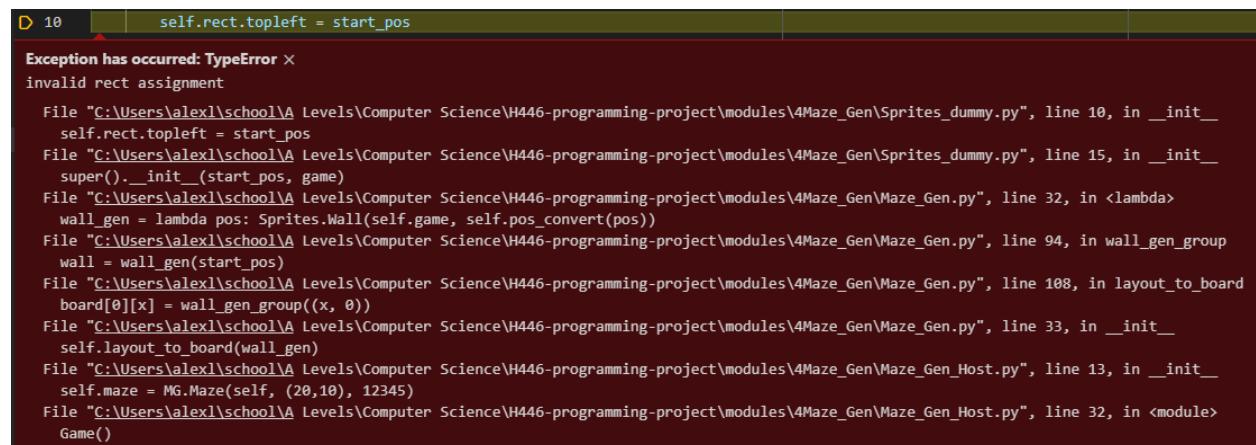
Give the dummy sprites a game argument

```
class Parent(pg.sprite.Sprite):
```

```
def __init__(self, game, start_pos):
    super().__init__()

    self.image = pg.surface.Surface((16,24))
    self.image.fill(self.colour)
    self.rect = self.image.get_rect()
    self.rect.topleft = start_pos
```

Error 2:



D 10 | self.rect.topleft = start_pos

Exception has occurred: TypeError ×
invalid rect assignment

File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Sprites_dummy.py", line 10, in __init__
 self.rect.topleft = start_pos
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Sprites_dummy.py", line 15, in __init__
 super().__init__(start_pos, game)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 32, in <lambda>
 wall_gen = lambda pos: Sprites.Wall(self.game, self.pos_convert(pos))
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 94, in wall_gen_group
 wall = wall_gen(start_pos)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 108, in layout_to_board
 board[0][x] = wall_gen_group((x, 0))
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 33, in __init__
 self.layout_to_board(wall_gen)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__
 self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>
 Game()

Problem 2:

Game arguments from solution 1 put in wrong place:

```
class Wall(Parent):
    def __init__(self, game, start_pos):
        self.colour = (32,32,32)
        super().__init__(start_pos, game)
```

Solution placed game arguments in correct place:

```
class Wall(Parent):
    def __init__(self, game, start_pos):
        self.colour = (32,32,32)
        super().__init__(game, start_pos)
```

Error 3:

```
D 109 | board[bsize[1]][x] = wall_gen_group((x, bsize[1])) |  
  
Exception has occurred: IndexError ×  
list index out of range  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 109, in layout_to_board  
    board[bsize[1]][x] = wall_gen_group((x, bsize[1]))  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 33, in __init__  
    self.layout_to_board(wall_gen)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>  
    Game()
```

Problem 3:

The size of an array is used as an index for that array; this is one off the end.

```
board[bsize[1]][x] = wall_gen_group((x, bsize[1]))
```

Solution 3:

Change indexing for arrays so that it is one less and thus aligns with the array:

```
board[bsize[1]-1][x] = wall_gen_group((x, bsize[1]-1))
```

Error 4:

```
D 141 | self.exit = Sprites.Exit(self.pos_convert(self.end)) |  
  
Exception has occurred: TypeError ×  
Exit.__init__() missing 1 required positional argument: 'start_pos'  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 141, in layout_to_board  
    self.exit = Sprites.Exit(self.pos_convert(self.end))  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 35, in __init__  
    self.layout_to_board(wall_gen)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>  
    Game()
```

Problem 4:

Exit constructor isn't being passed the game object:

```
self.exit = Sprites.Exit(self.pos_convert(self.end))
```

Solution 4:

Pass Exit constructor the game object. Several other sprite constructor calls also had this issue and were subsequently fixed aswell.

```
self.exit = Sprites.Exit(self.game, self.pos_convert(self.end))
```

Error 5:

```
D 223 known_nodes = [{start: False}]  
  
Exception has occurred: TypeError  
unhashable type: 'list'  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 223, in get_shortest_path  
    known_nodes = {start: False}  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 38, in __init__  
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>  
    Game()
```

Problem 5:

Using positions in list format as dictionary keys isn't allowed as they are unhashable:

```
> start: [1, 1]
```

```
known_nodes = {start: False}
```

Solution 5:

Convert position list format to a tuple: these are hashable and hold all the same data:

```
known_nodes = {tuple(start): False}
```

```
known_nodes: {(1, 1): False}
```

Error 6:

```
D 231 if self.board[neighbour[1]][neighbour[0]] != False:  
  
Exception has occurred: IndexError  
list index out of range  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 231, in get_shortest_path  
    if self.board[neighbour[1]][neighbour[0]] != False:  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 38, in __init__  
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>  
    Game()
```

Problem 6:

The algorithm is checking for neighbours on the edge of the board; this means that it is trying to access a slot on the board that doesn't exist. This usually doesn't happen because the walls around the outside of the maze stop it searching the perimeter slots of the board. However, where the exit is placed, the board was empty as the exit, while it had been created, hadn't been put into the board, thus providing a "break" in the wall where the search could check locations that don't exist:

```
# generate end  
self.end = [bsize[0]-1, bsize[1]-2]  
self.exit = Sprites.Exit(self.game, self.pos_convert(self.end))
```

Solution 6:

Place exit in board so that the algorithm can't query outside the board

```
# generate end
self.end = [bsize[0]-1, bsize[1]-2]
self.exit = Sprites.Exit(self.game, self.pos_convert(self.end))
# place exit in board
self.board[-2][-1] = self.exit
```

Error 7:

```
D 247 ▾ while known_nodes[current_node] != False:
Exception has occurred: TypeError ×
unhashable type: 'list'
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 247, in get_shortest_path
    while known_nodes[current_node] != False:
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 38, in __init__
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 13, in __init__
    self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 32, in <module>
    Game()
```

Problem 7:

Nodes (coordinates stored in list form) aren't hashable, meaning they can't be used as keys in a dictionary, much like problem 5:

```
> current_node: [40, 19]
```

```
current_node = end
while known_nodes[current_node] != False:
```

Solution 7:

Much like solution 5, the end position is converted to a tuple:

```
> current_node: (40, 19)
```

```
current_node = tuple(end)
while known_nodes[current_node] != False:
```

Error 8:

```
D 247 while known_nodes[current_node] != False:  
Exception has occurred: KeyError ×  
(40, 19)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 247, in get_shortest_path  
    while known_nodes[current_node] != False:  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 38, in __init__  
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>  
    Game()
```

Problem 8:

The end point is on the same board slot as the exit object; this means that it can't navigate to this point as it is obstructed by the exit object

```
# find start to end path  
self.start_to_end_path = self.get_shortest_path(self.start, self.end)
```

Solution 8:

Set end to the slot to the left of the exit object; this slot is possible to navigate to.

```
# find start to end path  
path_end = [self.end[0]-1, self.end[1]]  
self.start_to_end_path = self.get_shortest_path(self.start, path_end)
```

Error 9:

```
D 249 while known_nodes[tuple(current_node)] != False:  
Exception has occurred: KeyError ×  
(40, 19)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 249, in get_shortest_path  
    while known_nodes[tuple(current_node)] != False:  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 150, in populate  
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__  
    self.populate()  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>  
    Game()
```

Problem 9:

Start_to_end path is calculated twice for some reason, and the second one still gets it wrong as in error 8

```
def populate(self):  
    """populates the maze with sprites"""\n    # populate gateways and blocks  
    self.start_to_end_path = self.get_shortest_path(self.start, self.end)
```

Solution 9:

Remove erroneous start to end path calculation:

```
def populate(self):
    """populates the maze with sprites"""
    # populate gateways and blocks
```

Error 10:

D 160 | self.game.config.maze_gateway_jitter]

Exception has occurred: AttributeError
'Game' object has no attribute 'config'
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 160, in populate
 self.game.config.maze_gateway_jitter)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__
 self.populate()
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__
 self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>
 Game()

Problem 10:

Misspelled config:

```
self.game.config.maze_gateway_jitter)
```

Solution 10:

Spell config correctly:

```
self.game.config.maze_gateway_jitter)
```

Error 11:

157 | while node_index < path_len and len(remaining_colours) > 0:
158 | # select current node from path
159 | node_index += int(rng.random() * \n
160 | self.game.config.maze_gateway_jitter)
D 161 | current_node = self.start_to_end_path[node_index]

Exception has occurred: TypeError
list indices must be integers or slices, not float
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 161, in populate
 current_node = self.start_to_end_path[node_index]
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__
 self.populate()
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__
 self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>
 Game()

Problem 11:

Int function has been used as an attempt to round a value to an integer, but int has in this case returned a float:

```
node_index += int(rng.random() * \
                    self.game.config.maze_gateway_jitter)
```

```
node_index: 7.666666666666666
```

Solution 11:

Use round function to correctly round number to integer:

```
node_index += rng.random() * self.game.config.maze_gateway_jitter  
node_index = round(node_index)
```

```
node_index: 9
```

Error 12:

```
D 172 |     block = Sprites.Block(self.game, branch_node, block_colour)  
  
Exception has occurred: TypeError ×  
Block.__init__() takes 3 positional arguments but 4 were given  
  File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 172, in populate  
    block = Sprites.Block(self.game, branch_node, block_colour)  
  File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__  
    self.populate()  
  File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__  
    self.maze = MG.Maze(self, (20,10), 12345)  
  File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>  
    Game()
```

Problem 12:

The dummy block sprite constructor doesn't take a colour argument:

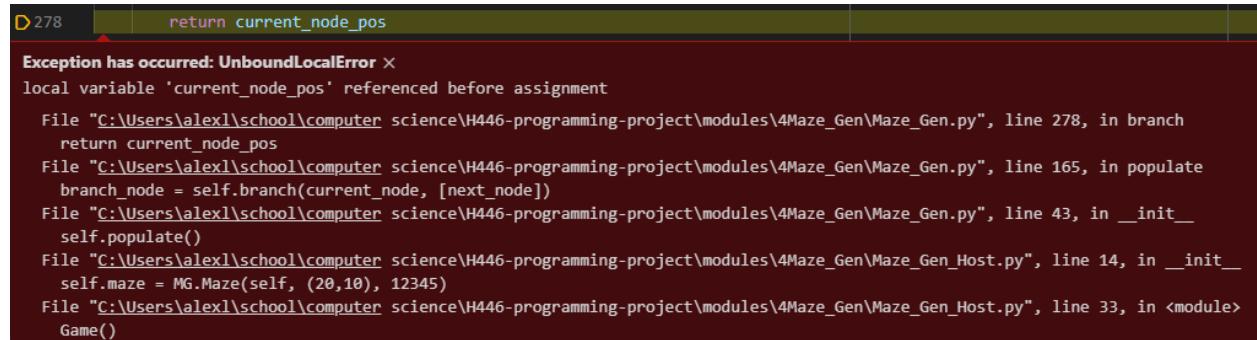
```
class Block(Parent):  
    def __init__(self, game, start_pos):  
        self.colour = (0,192,0)  
        super().__init__(game, start_pos)
```

Solution 12:

Add a colour argument to the dummy block sprite constructor: a similar fix was also applied for the dummy gateway constructor.

```
class Block(Parent):  
    def __init__(self, game, start_pos, colour):  
        self.colour = (0,192,0)  
        super().__init__(game, start_pos)
```

Error 13:



D 278 return current_node_pos

Exception has occurred: UnboundLocalError
local variable 'current_node_pos' referenced before assignment
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 278, in branch
 return current_node_pos
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 165, in populate
 branch_node = self.branch(current_node, [next_node])
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__
 self.populate()
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__
 self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>
 Game()

Problem 13:

The branch loop assigns current node at the start of each loop, but if it doesn't loop at all (no branching), then current node isn't assigned

```
while len(nodes_to_search) > 0 and \
        rng.random() < self.game.config.maze_branch_stop_threshold:

    current_node_pos = nodes_to_search.pop(0)
```

Solution 13:

Place the random chance of terminating branching at the end of the loop instead of the beginning, then the loop always runs at least once:

```
while len(nodes_to_search) > 0:

    current_node_pos = nodes_to_search.pop(0)

    for offset in [(0,-1), (0,1), (1,0), (-1,0)]:
        neighbour = [current_node_pos[i] + offset[i] for i in (0,1)]
        # check neighbour is a wall
        if self.board[neighbour[1]][neighbour[0]] != False:
            continue

        # check neighbour has already been searched
        if neighbour in known_nodes:
            continue

        # new node; add to known nodes and append to nodes_to_search
        known_nodes.append(current_node_pos)
        nodes_to_search.append(neighbour)

    if rng.random() > self.game.config.maze_branch_stop_threshold:
        break
```

Error 14:

```
D162 |     next_node = self.start_to_end_path[node_index + 1]

Exception has occurred: IndexError ×
list index out of range
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 162, in populate
    next_node = self.start_to_end_path[node_index + 1]
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__
    self.populate()
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__
    self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>
    Game()
```

Problem 14:

The current node is incremented by a small random value at the start of the while loop to provide randomness. This is in some cases increasing node index to a value such that when used to get the current node or the next node, it throws an index error.

```
while node_index < path_len and len(remaining_colours) > 0:
    # select current node from path
    node_index += rng.random() * self.game.config.maze_gateway_jitter
    node_index = round(node_index)
    current_node = self.start_to_end_path[node_index]
    next_node = self.start_to_end_path[node_index + 1]
```

Solution 14:

Place incrementing node index by a small number at the end of the loop, so that it can be checked by the while loops condition before trying to get current node and next node again

Error 15:

```
155 |     node_index = path_len * self.game.config.maze_blocks_start_proportion
156 |
157 |     while node_index + 1 < path_len and len(remaining_colours) > 0:
158 |
159 |         # select current node from path
160 |         current_node = self.start_to_end_path[node_index]

Exception has occurred: TypeError ×
list indices must be integers or slices, not float
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 161, in populate
    current_node = self.start_to_end_path[node_index]
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__
    self.populate()
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__
    self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>
    Game()
```

Problem 15:

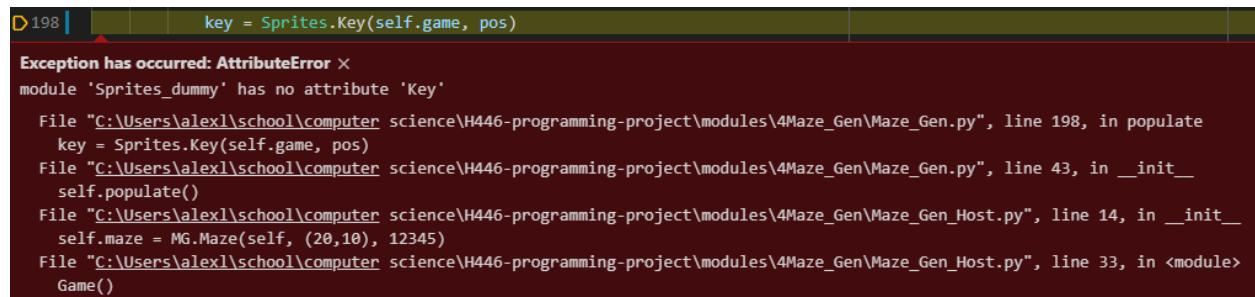
Node index is initially assigned to a float value, and must be rounded, but as solution 14 put rounding at the end of the loop, the first iteration is no longer rounded

Solution 15:

Place rounding back at the start of the loop

```
while node_index + 1 < path_len and len(remaining_colours) > 0:  
    node_index = round(node_index)  
  
    # select current node from path  
    current_node = self.start_to_end_path[node_index]
```

Error 16:



D 198 | key = Sprites.Key(self.game, pos)

Exception has occurred: AttributeError ×
module 'Sprites_dummy' has no attribute 'Key'
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 198, in populate
key = Sprites.Key(self.game, pos)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen.py", line 43, in __init__
self.populate()
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 14, in __init__
self.maze = MG.Maze(self, (20,10), 12345)
File "C:\Users\alex1\school\computer science\H446-programming-project\modules\4Maze_Gen\Maze_Gen_Host.py", line 33, in <module>
Game()

Problem 16:

Sprites dummy doesn't contain a Key class

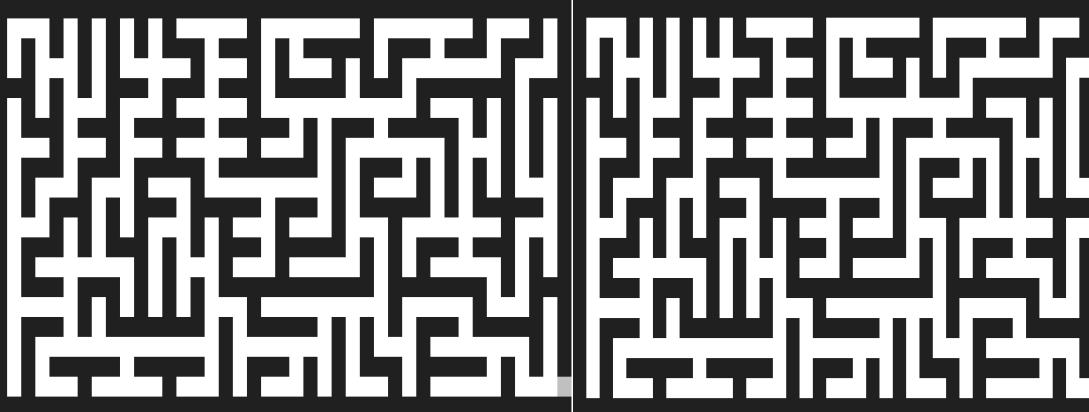
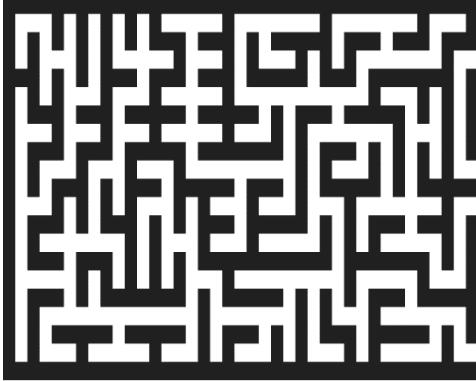
Solution 16:

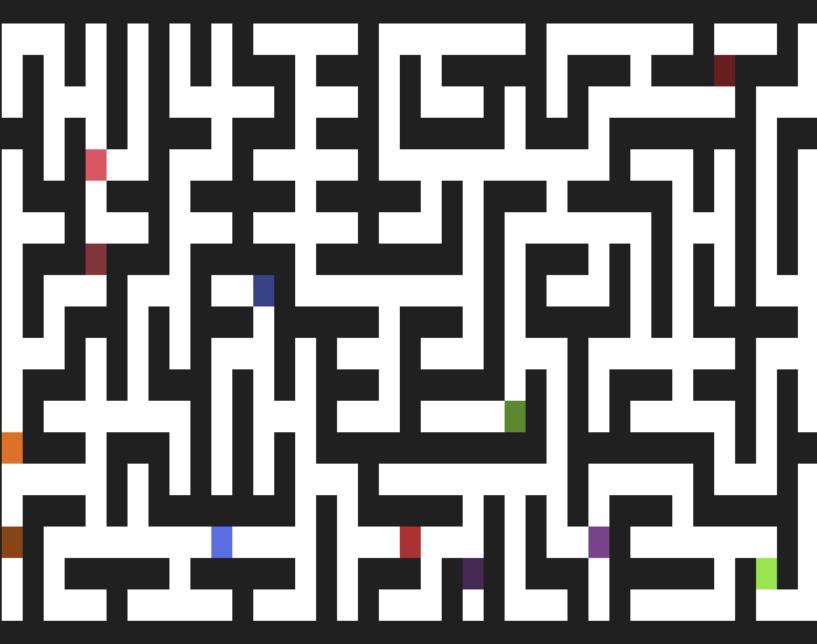
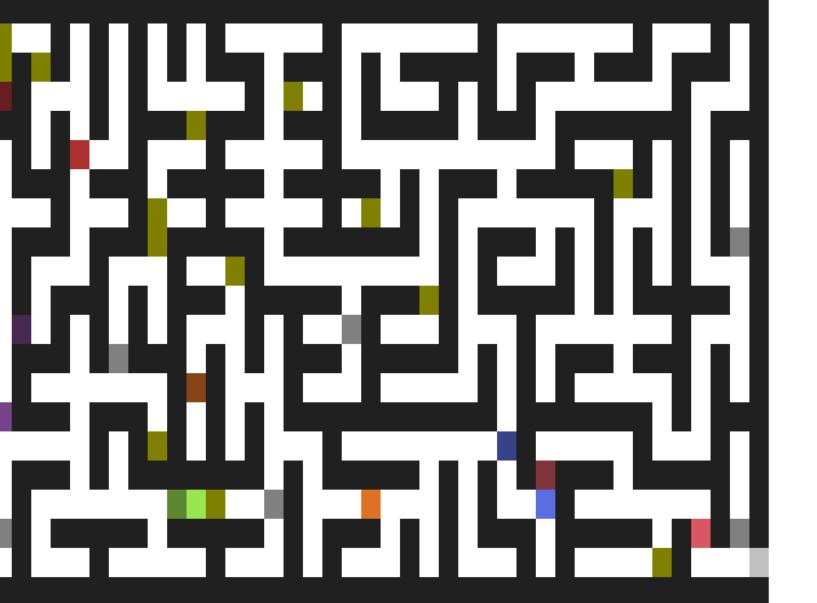
Implement a dummy key class in Sprites dummy

```
class Key(Parent):  
    def __init__(self, game, start_pos):  
        self.colour = (128,128,128)  
        super().__init__(game, start_pos)
```

Now that the code can start and runs without instantly crashing, it can be tested on the tests set out be the design stage.

Test Table:

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	 both mazes are identical	Pass
5	wall sprite generation	New maze object instantiated	Valid	Maze's maze_walls group is filled with sprites	<pre>✓ maze_walls: <Group(490 sprites)> > special variables > function variables > lostsprites: [] ✓ spritedict: {<Wall Sprite(in 2 groups)>: None, <Wall Sp > special variables > function variables <Wall Sprite(in 2 groups)>: None <Wall Sprite(in 2 groups)> (id: 1351763526560): None <Wall Sprite(in 2 groups)> (id: 1351763526704): None <Wall Sprite(in 2 groups)> (id: 1351763526848): None <Wall Sprite(in 2 groups)> (id: 1351763526902): None</pre>	all walls correctly generated pass
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	  maze is the same both on screen and in the console	Pass
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	<pre>> blocks: <Group(6 sprites)> > gateways: <Group(3 sprites)></pre>	Pass

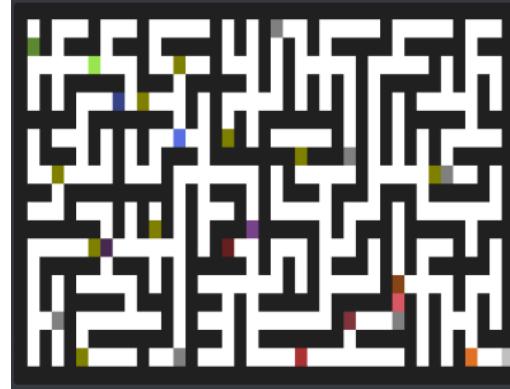
8	Gateway and block positioning	New maze object instantiated and board is printed	Valid	Blocks and gateways appear in same pattern on screen as they do on the board		<p>Fail: Problem: Gateways are before their corresponding blocks: branch can go forwards as well as backwards Neighbour is being declared as a list, but known nodes contains tuples</p> <pre>neighbour = [current_node_pos[i] + offset[i] for i in (0,1)] ↘ nodes_to_search: [[5, 3], [5, 5]] ↘ known_nodes: [(5, 5), (5, 4), (5, 4)] > special variables > function variables > 0: [5, 3] > 1: [5, 5] len(): 2</pre> <p>Solution: Convert neighbour to a tuple</p> <pre>neighbour = tuple(neighbour)</pre>
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	<pre>> exit: <Exit Sprite(in 1 groups)> > keys: <Group(6 sprites)> > enemies: <Group(4 sprites)></pre>	Pass
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		Pass

STAGE 4: STAKEHOLDER REVIEW

As the maze layout is critical to the gameplay, and thus the user experience, I have presented the following message to the stakeholders to see what their perspectives on the current game implementation:

Hi, this is a representation of typical procedural level generated for my game, where it is colour coded:
colours: lighter: gateways; locations the player can only pass through if they have the correct coloured block
colours: darker: blocks: must be picked up and brought to gateways to open them
dark grey: keys: these must all be collected to open the level exit
light grey (bottom right): exit - the main goal
yellow: checkpoints: places where the player can respawn
the start for the level is in the top left
what are your opinions on this kind of level structure? and what would you change. if you have any queries about rudimentary preview, please ask

(edited)



Ben's feedback focused on the colour scheme and presentation of this preview. He pointed out that the basic colour coding creates some ambiguity as to what is what, and this makes it harder to understand the level. These are concerns that will be resolved in the final game as all the sprites and graphics have yet to be completed. He also thinks that the colour scheme could do with some work: The red and pink colours are very similar, and with shades involved, they can be confused. As such, I have revised the colour scheme:

1: #AC3232	3: #99E550	5: #76428A
2: #DF7126	4: #0050EF	6: #00CCCC

Máté's feedback focused on the distribution of sprites around the maze. He says the general distribution is good as it adds progression to the game. He pointed out that the sprites tend to clump together, and this makes the level less satisfying than if each sprite were to be picked up individually. To rectify this, I will have to adjust the code that chooses a random point in the maze to put a sprite, and make it deliberately avoid choosing a space near already

populated blocks. He also said that there could be less checkpoints in the maze given the number of enemies; this is easy to change by adjusting the config file.

Now that the maze gen module passes all tests successfully and has been adjusted in response to the stakeholder's requests, it meets its success criteria and can thus be set aside until integration.

STAGE 5: SPRITES

This module contains all the sprites that will be rendered on the screen. As such, it contains the bulk of the game mechanics and a lot of the visuals, so it is critical that it meets or exceeds all its success criteria. It also makes strong use of inheritance as many sprites share similar functionality, and all of them must be rendered on the screen in the correct position relative to where the player is to ensure the player stays in the centre of the screen as they move around.

STAGE 5: DEVELOPMENT GOALS:

- All sprites can successfully load their assets and scale them to the correct resolution to render on screen
- All sprites are positioned correctly relative to the player by means of the camera object
- Camera object smoothly moves around the maze, following the player
- Player Controller allows movement around the maze and picking up and placing of blocks
- Player has collisions with walls that have no strange edge cases
- Enemies autonomously navigate randomly around the maze
- Enemies hurt the player when near them
- Gateways function, being obstructive until they have the corresponding block,
- Checkpoints function, allowing respawning at the correct place
- The exit functions: only opens when the player has all 6 keys
- Timer counts correctly, aligning with how long has been spent playing the level

STAGE 5: DEVELOPMENT PROCESS:

First draft:

```
import math
import random as rng
import time
from numpy import size
import pygame as pg

vec2 = pg.math.Vector2
colour_names = ["red", "orange", "green", "blue", "purple", "cyan"]

def collide_hit_rect(one, two):
    return one.hit_rect.colliderect(two.hit_rect)
```

```
class Renderable_Sprite(pg.sprite.Sprite):
    def __init__(self, game, start_pos=(0,0), start_rot=0):
        super().__init__()

        # initialise
        self.game = game
        self.camera = self.game.level.camera
        self.pos = vec2(start_pos)
        self.rot = start_rot
        self.layer = int(self.pos.y)

        # animations:
        self.imgs = []
        self.frame_index = 0
        self.frame_time = 100
        self.frame_countdown = 0
        self.culling = True

    def update(self, dt):
        pass

    def render(self, dt):
        # decrease frame_countdown
        self.frame_countdown -= dt
        # advance to next frame if less than 0
        if self.frame_countdown < 0:
            self.frame_countdown = self.frame_time
            self.frame_index = (self.frame_index + 1) % len(self.imgs)

        screen_pos = self.camera.wrld_2_sscrn_coord(self.pos)

        # retrieve correct img from imgs
        self.image = self.imgs[self.frame_index]

        # rotating and scaling images is expensive; only do it if the sprite
        # is visible on screen; this makes the game faster at higher zooms
        if self.culling == False or \
            screen_pos.x-300 < (ssize := self.game.screen.get_size())[0] and \
            screen_pos.x+300 > 0 and \
            screen_pos.y-300 < ssize[1] and \
            screen_pos.y+300 > 0:
```

Default values for variable are initialized

Advancing to the next frame of animation

Preparing this sprite's image for rendering involves rotating and scaling it correctly so that is the correct size and orientation for on screen this is very computationally intensive, especially the larger the images get, which happens at higher zoom levels. To increase performance, the position on screen of the sprite is checked, and if it is off screen, then it doesn't need to be rendered correctly, so these expensive steps are skipped. It still ends up being drawn to the display surface in an incorrect state, but this is far off screen, so isn't a problem. Stopping it being drawn would be too expensive and complicated to implement

```
        self.image = pg.transform.scale(self.image, [oord*self.camera.zoom
                                                    for oord in self.image.get_size()])

        # set rect position correctly
        self.rect = self.image.get_rect()
        self.rect.bottomleft = screen_pos

        # place hit_rect position correctly
        opp_corner = self.camera.wrlrd_2_scrn_coord(self.pos + vec2(1,-1))
        self.hit_rect = pg.Rect(0,0,self.rect[2], self.rect[2])
        self.hit_rect.bottomleft = screen_pos

    def get_facing_offset(self):
        """returns the direction the sprite is facing"""
        if 45 <= self.rot % 360 < 135:
            return vec2(1,0)
        elif 135 <= self.rot % 360 < 225:
            return vec2(0,-1)
        elif 225 <= self.rot % 360 < 315:
            return vec2(-1,0)
        else:
            return vec2(0,1)

class Player(Renderable_Sprite):
    def __init__(self, game, start_pos):
        # call parent constructor
        super().__init__(game, start_pos, 0)
        self.maze = self.game.level.maze

        self.colour = 0
        # set health
        self.health = game.config.player_max_health
        # set hurt cooldown
        self.hurt_cooldown = game.config.player_hurt_cooldown

        # kinematics
        self.vel = vec2(0,0)
        # set max speed
        self.max_speed = game.config.player_max_speed
        # set acc
        self.acc = game.config.player_acc
```

This utility method provides a vector to represent the direction the sprite is currently facing; this comes in useful in multiple sprites, so is in the parent

```

# animations
self.animation_state = "standing"
# load animation frames

right_imgs = [[self.game.img_loader.get(f"player_{colour}3")
               for colour in colour_names], \
              [self.game.img_loader.get(f"player_{colour}2")
               for colour in colour_names]]
right_imgs += right_imgs.copy()

left_imgs = [[pg.transform.flip(img, True, False)
              for img in frame]
             for frame in right_imgs]

down_imgs = [[self.game.img_loader.get(f"player_{colour}5")
              for colour in colour_names], \
              [self.game.img_loader.get(f"player_{colour}4")
               for colour in colour_names]]
down_imgs += [[pg.transform.flip(img, True, False)
              for img in frame]
             for frame in down_imgs]

up_imgs = [[self.game.img_loader.get(f"player_{colour}1")
            for colour in colour_names], \
           [self.game.img_loader.get(f"player_{colour}0")
            for colour in colour_names]]
up_imgs += [[pg.transform.flip(img, True, False)
            for img in frame]
           for frame in up_imgs]

self.standing_imgs = { (1,0) : [right_imgs[1]],
                      (-1,0) : [left_imgs[1]],
                      (0,-1) : [down_imgs[1]],
                      (0,1) : [up_imgs[1]] }
self.walking_imgs = { (1,0) : right_imgs,
                      (-1,0) : left_imgs,
                      (0,-1) : down_imgs,
                      (0,1) : up_imgs }

# sounds
self.walk_sounds = []
for i in range(1,9):
    self.walk_sounds.append(self.game.snd_loader.get( \

```

Player images are stored in a 2d array: the 1st axis is for the frame index, and then the second index is for the different colours. Disk space is saved by mirroring the right images to get the left images. This is handled by a lot of list comprehensions to deal with all the arrays

```

        f"stepdirt_{i}.wav"))
self.key_collect_snd = self.game.snd_loader.get("key_pickup.wav")
self.block_collect_snd = self.game.snd_loader.get("stepstone_4.wav")
self.block_place_snd = self.game.snd_loader.get("stepstone_1.wav")
self.gateway_open_snd = self.game.snd_loader.get("DoorOpen07.ogg")
self.respawn_snd = self.game.snd_loader.get("Hit_Hurt6.wav")
self.hurt_snd = self.game.snd_loader.get("Hit_Hurt6.wav")

self.step_sound_timer = self.game.config.player_step_snd_delay

# set up other mechanics
self.inventory = [False, False]
# stores the previous key state for edge detection
self.prev_slot_keys = [False, False]
# stores colour keys state:
self.colour_key_pressed = False

self.keys = 0
self.last_checkpoint = False

# render once to set up rect for collisions
self.walking = False
self.render(0)

def update(self, dt):
    # acceleration due to wasd
    keys = pg.key.get_pressed()
    walking_x = False
    walking_y = False
    if keys[pg.K_LEFT] or keys[pg.K_a]:
        # acc left
        self.vel.x -= self.acc * dt/1000
        walking_x = True
    elif keys[pg.K_RIGHT] or keys[pg.K_d]:
        # acc right
        self.vel.x += self.acc * dt/1000
        walking_x = True

    elif keys[pg.K_UP] or keys[pg.K_w]:
        # acc up
        self.vel.y -= self.acc * dt/1000
        walking_y = True
    elif keys[pg.K_DOWN] or keys[pg.K_s]:
        # acc down

```

Sounds are loaded by the game's sound loader and stored for use during the game

Collisions are in the update part of the main loop, but depend on all sprites having a hit_rect, which is computed at the render stage, so to prevent there being an error the first time the loop runs, all sprites are rendered once during initialization to set up their hit_rect

Movement input keys: implements 4 way movement for the player

```

        self.vel.y += self.acc * dt/1000
        walking_y = True

# decelerate back to vel = 0 if not walking
if not walking_x:
    self.vel.x -= min(self.acc * self.vel.x, self.vel.x*0.9,
                      key = lambda x: abs(x))
if not walking_y:
    self.vel.y -= min(self.acc * self.vel.y, self.vel.y*0.9,
                      key = lambda x: abs(x))
self.walking = walking_x or walking_y

# enforce max speed
self.vel = vec2([min(max(-self.max_speed, self.vel[i]), self.max_speed) \
    for i in (0,1)])\

# block picking and placing
slot_keys = [keys[pg.K_q], keys[pg.K_e]]
for slot_index in (0,1):
    # only trigger on the rising edge of the key press
    if slot_keys[slot_index] and not self.prev_slot_keys[slot_index]:
        # if the slot is empty, pick up
        if self.inventory[slot_index] == False:
            self.pick_up(slot_index)
        # if slot is full, place
        else:
            self.place(slot_index)
    self.prev_slot_keys = slot_keys

# get which keys are pressed for colour changing
colour_keys = [keys[k] for k in (pg.K_1, pg.K_2, pg.K_3,
                                  pg.K_4, pg.K_5, pg.K_6)]
# only change colour on rising edge a a key being pressed
if not self.colour_key_pressed:
    for key_index in range(0,6):
        if colour_keys[key_index]:
            self.colour = key_index

self.colour_key_pressed = max(colour_keys)

# collisions with keys
hits = pg.sprite.spritecollide(self,
                               self.maze.keys,
                               True,

```

If the player isn't walking in a direction, they decelerate back to not moving at all by decreasing their velocity

This checks if there are blocks in the player's inventory and acts accordingly when block placement and grabbing keys are pressed

This implements the player changing colour when the number keys are pressed

```

                collide_hit_rect)
self.keys += len(hits)
if len(hits):
    self.key_collect_snd.play()

# collisions with enemies
hits = pg.sprite.spritecollide(self,
                                self.maze.enemies,
                                False,
                                collide_hit_rect)
if [h for h in hits if h.colour != self.colour]:
    # only take damage if hurt cooldown has elapsed
    if self.hurt_cooldown <= 0:
        # reset hurt cooldown
        self.hurt_cooldown = self.game.config.player_hurt_cooldown
        # take damage
        self.health -= 1
        # play hurt sound
        self.hurt_snd.play()

# decrease hurt cooldown
self.hurt_cooldown = max(0, self.hurt_cooldown - dt)

# play step sound if step sound timer has elapsed
if self.walking:
    if self.step_sound_timer <= 0:
        rng.choice(self.walk_sounds).play()
        self.step_sound_timer = self.game.config.player_step_snd_delay
    else:
        self.step_sound_timer -= dt
else:
    self.step_sound_timer = 0

# update rot based on movement
if self.vel.length != 0:
    self.rot = self.vel.angle_to(vec2(0,1))

# collide with walls
self.collide()

# change position by velocity
self.pos += self.vel

# respawn

```

All collisions code uses the sprite's hit_rect instead of the draw rect as the draw rects should overlap because of the orthographic top down rendering

The walking sounds are randomly selected each time they play, which is at set intervals when the player is walking

```

if self.health == 0:
    self.respawn()

# change layer
self.maze.all_sprites.change_layer(self, int(self.pos.y))

def pick_up(self, slot_index):
    """picks up block in front of the player and stores in inventory"""
    # slot is already full
    if self.inventory[slot_index]:
        return

    # find block in front of the player
    facing_pos = (self.pos+vec2(0.25,0.75))/1 + self.get_facing_offset()
    facing_sprite = self.maze.board[int(facing_pos.y)][int(facing_pos.x)]

    # check if it is a block
    if type(facing_sprite).__name__ == "Block":
        # store this block to inventory
        self.inventory[slot_index] = facing_sprite

    # remove this block from board
    self.maze.board[int(facing_pos.y)][int(facing_pos.x)] = False

    # remove block from all sprites and blocks
    self.maze.blocks.remove(facing_sprite)
    self.maze.all_sprites.remove(facing_sprite)

    # play block collection sound
    self.block_collect_snd.play()

def place(self, slot_index):
    """places a block in front of the player from inventory slot"""
    # nothing to place
    if self.inventory[slot_index] == False:
        return

    # find block in front of the player
    facing_pos = (self.pos+vec2(0.25,0.75))/1 + self.get_facing_offset()
    facing_sprite = self.maze.board[int(facing_pos.y)][int(facing_pos.x)]

    # ensure the space is free before placing in a free space
    if facing_sprite == False:
        # remove block from inventory

```

All the sprites are stored in a layeredupdates group that ensures that all sprites are drawn in the correct order so that when they overlap, they overlap correctly to appear 3d. for this to work, the layer the player is in must be correct, and is thus updated every frame so that the player is drawn correctly amidst the walls

Calculating the position of the block in front of the player is trickier than I was expecting: the player's origin is located at the bottom left corner, so the correct offsets are needed to access its corresponding board position correctly

```

block = self.inventory[slot_index]
self.inventory[slot_index] = False

# set block's new position
block.pos = facing_pos//1

# store block in inventory there
self.maze.board[int(facing_pos.y)][int(facing_pos.x)] = block

# add block to all sprites and blocks
self.maze.all_sprites.add(block)
self.maze.blocks.add(block)

# change block's layer so it renders correctly
self.maze.all_sprites.change_layer(block, int(block.pos.y))

# play placing sound
self.block_place_snd.play()

# if space isn't free, check if it is a gateway
if type(facing_sprite).__name__ == "Gateway" and \
    facing_sprite.colour == self.inventory[slot_index].colour:
    # remove both the block and gateway
    self.inventory[slot_index].kill()
    self.inventory[slot_index] = False

    facing_sprite.kill()
    self.maze.board[int(facing_pos.y)][int(facing_pos.x)] = False
    self.gateway_open_snd.play()

def respawn(self):
    """respawns the player at the correct location when they die"""
    # play respawn sound
    self.respawn_snd.play()

    # set health to player max health
    self.health = self.game.config.player_max_health

    if self.last_checkpoint != False:
        self.pos = vec2(self.last_checkpoint.pos)
    else:
        self.pos = vec2(self.maze.start)

def collide(self):

```

An extra set of checks are run for if it is a gateway that is being placed on; there is the possibility that it needs to be deleted along with the block

```

# overcomplicated collisions to remove weird snapping
# - no dependency on velocities, which can cause problems

def check_collidable(sprite):
    return type(sprite).__name__ == "Wall" or \
           (type(sprite).__name__ == "Block" and
            sprite.colour != self.colour) or \
           type(sprite).__name__ == "Gateway" or \
           type(sprite).__name__ == "Exit" and self.keys != 6

board = self.maze.board

for sprite in self.maze.all_sprites:
    if collide_hit_rect(sprite, self) and check_collidable(sprite):

        if self.hit_rect.x+10 > sprite.hit_rect.right:
            # wall on left
            spot = sprite.pos//1 + vec2(1,0)

            # collide if outside the bounds or isn't a collideable
            if not(0 <= spot.x < self.maze.bsize[0]) or \
               not(check_collidable(board[int(spot.y)][int(spot.x)])):
                self.vel.x = max(0, self.vel.x)

        elif self.hit_rect.bottom-10 < sprite.hit_rect.y:
            # wall below
            spot = sprite.pos//1 + vec2(0,-1)

            # collide if outside the bounds or isn't a collideable
            if not(0 <= spot.y < self.maze.bsize[0]) or \
               not(check_collidable(board[int(spot.y)][int(spot.x)])):
                self.vel.y = min(0, self.vel.y)

        elif self.hit_rect.right-10 < sprite.hit_rect.x:
            # wall on right
            spot = sprite.pos//1 + vec2(-1,0)

            # collide if outside the bounds or isn't a collideable
            if not(0 <= spot.x < self.maze.bsize[0]) or \
               not(check_collidable(board[int(spot.y)][int(spot.x)])):
                self.vel.x = min(0, self.vel.x)

        elif self.hit_rect.y+10 > sprite.hit_rect.bottom:
            # wall above

```

The collisions code was difficult to develop to be satisfactory as it must account for a lot of behavior: it must figure out the direction of the collision, what that should do to the player's velocity, and if the collision is actually a valid collision or just an error in the collision detection: it achieves this by checking if the edges are actually exposed, which can be determined from the maze's board, and then it cancels out the correct component of velocity

```

spot = sprite.pos//1 + vec2(0,1)

# collide if outside the bounds or isn't a collideable
if not(0 <= spot.y < self.maze.bsize[0]) or \
not(check_collidable(board[int(spot.y)][int(spot.x)])):
    self.vel.y = max(0, self.vel.y)

def render(self, dt):
    """render the player sprite"""
    facing_dir = self.get_facing_offset()

    # retrieve correct set of imgs for player's current rotation
    if self.walking:
        self.imgs = self.walking_imgs[tuple(facing_dir)]
    else:
        self.imgs = self.standing_imgs[tuple(facing_dir)]
        self.frame_index = 0

    # decrease frame_countdown
    self.frame_countdown -= dt
    # advance to next frame if less than 0
    if self.frame_countdown < 0:
        self.frame_countdown = self.frame_time
        self.frame_index = (self.frame_index + 1) % len(self.imgs)

    # retrieve correct img from imgs
    self.image = self.imgs[self.frame_index][self.colour]
    self.image = pg.transform.scale(self.image,
                                   [oord * self.camera.zoom // 2 for
                                    oord in self.image.get_size()])

    # data for rendering of blocks in inventory
    b1_pos = {(0,-1) : vec2(3,6),
               (1,0) : vec2(0,6),
               (-1,0) : vec2(6,6)}
    b2_pos = {(0,-1) : vec2(3,4.5),
               (1,0) : vec2(0,4.5),
               (-1,0) : vec2(6,4.5)}

    # render blocks in inventory so that they scale correctly
    for block in self.inventory:
        if block:
            block.render(0)

```

Rendering the player requires adding on the blocks in the player's inventory, and thus polymorphism is used to support this extended functionality

The data for how the inventory should be drawn is stored in this dictionary

```

# no image if facing down
if facing_dir != (0,1):
    # work out the position and size of each block's image
    if block_1 := self.inventory[0]:
        block_1_image = pg.transform.scale(
            block_1.image, vec2(block_1.image.get_size())//8)

        block_1_pos = b1_pos[tuple(facing_dir)] * self.camera.zoom

    if block_2 := self.inventory[1]:
        block_2_image = pg.transform.scale(
            block_2.image, vec2(block_2.image.get_size())//8)

        block_2_pos = b2_pos[tuple(facing_dir)] * self.camera.zoom

# blit on top if facing up, otherwise blit behind
if facing_dir == (0,-1):
    if block_1:
        self.image.blit(block_1_image, block_1_pos)
    if block_2:
        self.image.blit(block_2_image, block_2_pos)
else:
    image = pg.surface.Surface(self.image.get_size(),
                                flags = pg.SRCALPHA)
    if block_1:
        image.blit(block_1_image,block_1_pos)
    if block_2:
        image.blit(block_2_image, block_2_pos)
    image.blit(self.image, (0,0))
    self.image = image

# set rect position correctly
self.rect = self.image.get_rect()
screen_pos = self.camera.wrld_2_scrn_coord(self.pos)
self.rect.bottomleft = screen_pos

# place hit_rect position correctly
opp_corner = self.camera.wrld_2_scrn_coord(self.pos + vec2(1,-1))
self.hit_rect = pg.rect.Rect(0,0,self.rect[2], self.rect[2])
self.hit_rect.bottomleft = screen_pos


class Enemy(Renderable_Sprite):
    def __init__(self, game, start_pos):

```

How the blocks are blitted to the player's image is dependent on the inventory and the player's facing direction, and thus there is different code for each situation so that it can be rendered in front or behind, and in different locations, depending on what is best

```

super().__init__(game, start_pos)
self.colour = rng.randint(0,5)
self.maze = self.game.level.maze

# load all animation frames from img loader

left_imgs = [[self.game.img_loader.get(f"enemy_{colour}{i}")]
             for colour in colour_names]
for i in range(4)

right_imgs = [[pg.transform.flip(img, True, False)
               for img in frame]
              for frame in left_imgs]

up_imgs = [[self.game.img_loader.get(f"enemy_{colour}6")
            for colour in colour_names], \
           [self.game.img_loader.get(f"enemy_{colour}7")]
            for colour in colour_names]]
up_imgs += [[pg.transform.flip(img, True, False)
             for img in frame]
            for frame in up_imgs]

down_imgs = [[self.game.img_loader.get(f"enemy_{colour}4")
              for colour in colour_names], \
             [self.game.img_loader.get(f"enemy_{colour}5")]
              for colour in colour_names]]
down_imgs += [[pg.transform.flip(img, True, False)
               for img in frame]
              for frame in down_imgs]

self.walking_imgs = {(-1,0) : left_imgs,
                     (1,0) : right_imgs,
                     (0,1) : down_imgs,
                     (0,-1) : up_imgs
                    }

# get a path
self.get_path()

def get_path(self):
    """calculates a new path for this sprite to follow"""

    current_pos = (self.pos+vec2(0.25,0.75))//1

```

The enemies are very similar to the player in many ways, such as asset loading and general behavior, though they are simpler as they have less to do. The main difference is that the enemy controller uses path finding to randomly navigate around the maze, following a calculated path

```

current_pos = (int(current_pos.x), int(current_pos.y))
destination = self.maze.branch(current_pos, [])

# find path to this location
self.target_path = self.maze.get_shortest_path(current_pos, destination)

def update(self, dt):
    # get direction to current target
    target = vec2(self.target_path[0]) + vec2(0.25, -0.25)
    target_delta = target - self.pos

    # if it has reached the target, remove it from target_path
    if target_delta.length() < 0.05:
        self.target_path.pop(0)

    # if the whole path has been followed, generate a new one
    if len(self.target_path) == 0:
        self.get_path()

    # recompute target
    target = vec2(self.target_path[0])
    target_delta = target - self.pos

    # move towards target
    if target_delta.length() != 0:
        self.vel = target_delta.normalize() * self.game.config.enemy_speed
        self.rot = self.vel.angle_to(vec2(0,1))
        self.pos += self.vel

    # change layer
    self.maze.all_sprites.change_layer(self, int(self.pos.y))

def render(self, dt):
    """render the enemy sprite"""
    facing_dir = self.get_facing_offset()
    self.imgs = self.walking_imgs[tuple(facing_dir)]

    # decrease frame_countdown
    self.frame_countdown -= dt
    # advance to next frame if less than 0
    if self.frame_countdown < 0:
        self.frame_countdown = self.frame_time
        self.frame_index = (self.frame_index + 1) % len(self.imgs)

```

When the enemy needs a new target, it uses the branch algorithm to search for possible targets. It then uses the get shortest path method (dijkstra's algorithm) to find a path to this location

This code checks if the enemy needs to advance onto the next target in their path, or if they have arrived at the end of the path, and thus need to calculate a new one

The enemy always moves in a strait line to it's target position at a set velocity. As the path is exclusively within the paths of the maze, they don't collide with the walls

```
# retrieve correct img from imgs
self.image = self.imgs[self.frame_index][self.colour]
self.image = pg.transform.scale(self.image,
                               [oord * self.camera.zoom // 1.5 for
                                oord in self.image.get_size()])

# set rect position correctly
self.rect = self.image.get_rect()
screen_pos = self.camera.wrld_2_scrn_coord(self.pos)
self.rect.bottomleft = screen_pos

# place hit_rect position correctly
opp_corner = self.camera.wrld_2_scrn_coord(self.pos + vec2(1,-1))
self.hit_rect = pg.Rect(0,0,self.rect[2], self.rect[2])
self.hit_rect.bottomleft = screen_pos
```

```
class Wall(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        # initialise assets
        self.imgs = [game.img_loader.get("brick dark grey")]
```

Most of the functionality is implemented by the player, meaning these sprites are very simple, and only need to initialize some assets to work in the maze

```
class Gateway(Renderable_Sprite):
    def __init__(self, game, start_pos, colour):
        super().__init__(game, start_pos)
        self.colour = colour

        # initialise assets
        self.imgs = [game.img_loader.get(
            f"gateway_{colour_names[self.colour]}")]
```

```
class Block(Renderable_Sprite):
    def __init__(self, game, start_pos, colour):
        super().__init__(game, start_pos)
        self.colour = colour
        self.culling = False

        # initialise assets
        match self.colour:
            case 0:
```

```
        self.imgs = [game.img_loader.get("crate light red")]
case 1:
    self.imgs = [game.img_loader.get("crate orange")]
case 2:
    self.imgs = [game.img_loader.get("crate lime")]
case 3:
    self.imgs = [game.img_loader.get("crate dark blue")]
case 4:
    self.imgs = [game.img_loader.get("crate purple")]
case 5:
    self.imgs = [game.img_loader.get("crate light blue")]

class Checkpoint(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        self.active = False

        # init active images
        self.active_imgs = []
        for img_index in range(0,6):
            img = self.game.img_loader.get("Flag{" + str(img_index) + "}")
            self.active_imgs.append(img)
        # init deactivate images
        self.deactive_imgs = []
        for img_index in range(0,2):
            img = self.game.img_loader.get("Flag_down{" + str(img_index) + "}")
            self.deactive_imgs.append(img)
        # innit
        self.imgs = self.deactive_imgs

    def update(self, dt):
        if not self.active:
            if (self.pos - self.game.level.player.pos).length() < 0.5:
                # deactivate currently active checkpoint
                if last_checkpoint := self.game.level.player.last_checkpoint:
                    last_checkpoint.deactivate()

                # activate this checkpoint
                self.activate()

    def activate(self):
        """activates this checkpoint"""


```

```

        self.active = True
        self.imgs = self.active_imgs
        self.game.level.player.last_checkpoint = self

    def deactivate(self):
        """deactivates this checkpoint"""
        self.active = False
        self.imgs = self.deactive_imgs
        self.frame_index = 0


class Key(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        key_frame_count = self.game.config.key_frame_count
        key_displacement = self.game.config.key_displacement

        # generate images
        key_image = self.game.img_loader.get("key cream")
        key_image = pg.transform.scale(key_image, vec2(key_image.get_size()))
        self.imgs = []

        for i in range(key_frame_count):
            displaced_img = pg.surface.Surface(
                key_image.get_size() + vec2(0, key_displacement),
                flags = pg.SRCALPHA)

            # calculate how far this image must be moved
            offset = (math.cos(2*math.pi * (i/key_frame_count) ) + 1) \
                     * key_displacement // 2

            # blit key image to displaced image in correct location
            displaced_img.blit(key_image, (0, offset))

            # append to imgs
            self.imgs.append(displaced_img)

    class Exit(Renderable_Sprite):
        def __init__(self, game, start_pos):
            super().__init__(game, start_pos)

            # load images

```

The checkpoints need to check if the player is near and activate if this is the case. They have to change the player's current checkpoint so that the player respawns at the correct checkpoint.

Keys implement the hovering animation during initialization by offsetting each frame by an amount dictated by a cosine wave

```

        self.state_imgs = []
        for img_index in range(0,7):
            self.state_imgs.append(
                self.game.img_loader.get(f"exit_locked{img_index}"))
        self.state_imgs.append(self.game.img_loader.get("exit_open"))
        self.imgs = [self.state_imgs[0]]

    def update(self, dt):
        keys = self.game.level.player.keys
        if keys == 6:
            self.imgs = self.state_imgs[-2:]
        else:
            self.imgs = [self.state_imgs[keys]]


class Camera():
    def __init__(self, game, target = False):
        self.game = game
        self.target = target
        self.pos = vec2(0,0) # this location is the centre of the screen
        self.zoom = self.game.config.camera_zoom

        # invisible default image to support being part of all sprites
        self.img = pg.surface.Surface((1,1))
        self.img.fill((0,0,0))
        self.rect = self.img.get_rect()
        self.rect.topleft = (-1000,-1000)

    def set_target(self, target):
        """sets the sprite which the camera should follow"""
        self.target = target

    def update(self, dt):
        """updates the position of the camera so that it tracks the player"""

        # adjust the camera pos
        target_pos = vec2(self.target.pos)
        target_pos_delta = -(target_pos - self.pos)
        self.pos = self.pos - 0.1*target_pos_delta

        # ensure camera never goes off screen
        unscaled_scrn_size = vec2(self.game.config.resolution)/ self.zoom / 16
        wrld_size = vec2(self.game.level.maze.bsize)

```

the camera must follow the player around the maze, but also not jolt around in a disorienting way; this is achieved by accelerating towards the target position, rather than just snapping to it

```
left_edge = unscaled_scrn_size.x/2
right_edge = wrld_size.x - unscaled_scrn_size.x
top_edge = unscaled_scrn_size.y/2 - 1.4
bottom_edge = wrld_size.y - unscaled_scrn_size.y

self.pos.x = min(max(left_edge, self.pos.x), right_edge)
self.pos.y = min(max(top_edge, self.pos.y), bottom_edge)

def wrld_2_scrn_coord(self, wrld_coord):
    """takes a world space coordinate and converts it to screenspace"""
    scrn_size = vec2(self.game.config.resolution)

    # ensures that the cameras position ends up at the centre of the screen
    scaled_wrld_coord = vec2(wrld_coord) * self.zoom * 16
    scaled_pos = self.pos * self.zoom * 16
    ss_coord = scaled_wrld_coord + scrn_size/2 - scaled_pos
    return ss_coord
```

To stop the camera from showing areas that are outside the maze, the edges of the maze in world space are calculated, and then used to constrain the position of the camera

```
class Timer():
    def __init__(self, game):
        self.game = game
        self.reset()

        # invisible default image to support being part of all sprites
        self.img = pg.surface.Surface((1,1))
        self.img.fill((0,0,0))
        self.rect = self.img.get_rect()
        self.rect.topleft = (-1000,-1000)

    def update(self, dt):
        # dt is in ms, but total time is in s so dt is scaled correctly
        self.total_time += dt/1000

    def reset(self):
        self.total_time = 0.0
        self.start_time = time.time()
```

The world space coordinates are converted to screen space coordinates using some linear transformations.

STAGE 5:UNIT TESTING:

Test Host Program:

To test the sprites, the test host program emulates the situation they will eventually be used in:

- It sets up a maze, which generates the sprites
- It updates the sprites in a main loop, just as they will be in the main game
- It provides the parts of program layout that the final game will have, allowing all the sprite's functionality to work

Test program code:

```
import config as cfg
import Maze_Gen as mg
import Asset_Loader as al
import pygame as pg
import Sprites as sprites
import random as rng

class Game():
    def __init__(self):
        # init pygame
        pg.init()

        # config
        self.config = cfg.Config()

        # init screen
        self.screen = pg.display.set_mode(self.config.resolution)

        # init asset loaders
        self.img_loader = al.Img_Loader(self)
        self.snd_loader = al.Snd_Loader(self)

        # run level level
        self.level = Level(self)
        self.level.setup()
        self.level.loop()

class Level():
    def __init__(self, game):
        self.game = game
        self.timer = sprites.Timer(self.game)

    def setup(self):
        """sets up the level"""
        # initialise camera
        self.camera = sprites.Camera(self.game)
```

```
# start setting up the maze
self.maze = mg.Maze(self.game, (20,10), rng.randint(0,10000))
# finishes generating the maze and sprites
self.maze.setup()
# initialise sprites in maze
for sprite in self.maze.all_sprites:
    sprite.render(0)

# initialise player
self.player = sprites.Player(self.game, (self.maze.start))
self.maze.all_sprites.add(self.player)
# set what the camera should follow
self.camera.set_target(self.player)
self.camera.pos = pg.Vector2(5,5)

def loop(self):

    clock = pg.time.Clock()
    while True:
        dt = clock.tick(75)

        for event in pg.event.get():
            if event.type == pg.QUIT:
                return
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_t:
                    print(self.timer.total_time)
                if event.key == pg.K_r:
                    self.timer.reset()

            if event.type == pg.MOUSEBUTTONDOWN:
                if event.button == 4:
                    self.camera.zoom = min(self.camera.zoom+1, 20)
                if event.button == 5:
                    self.camera.zoom = max(self.camera.zoom-1 , 1)

        # update all sprites
        self.maze.all_sprites.update(dt)
        self.camera.update(dt)
        self.timer.update(dt)

        # call all sprites render method
        for sprite in self.maze.all_sprites:
```

```
        sprite.render(dt)

        self.game.screen.fill((32,32,32))
        self.maze.all_sprites.draw(self.game.screen)

    for sprite in self.maze.all_sprites:
        pg.draw.rect(self.game.screen, (255,255,255), sprite.hit_rect, 1)

    pg.display.flip()

Game()
```

Issue resolution:

Error 1:

```
10 | v class Renderable_Sprite(pg.sprite.Sprite):
11 |     def __init__(self, game, start_pos=(0,0), start_rot=0):
12 |         super().__init__()
13 |
14 |         # initialise
15 |         self.game = game
D 16 |         self.maze = self.game.level.maze

Exception has occurred: AttributeError
'Game' object has no attribute 'level'
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 16, in __init__
  self.maze = self.game.level.maze
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 481, in __init__
  super().__init__(game, start_pos)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 30, in wall_gen
  return sprites.Wall(self.game, pos)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 107, in wall_gen_group
  wall = wall_gen(start_pos)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 121, in layout_to_board
  board[0][x] = wall_gen_group((x, 0))
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 31, in __init__
  self.layout_to_board(wall_gen)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite Host.py", line 29, in __init__
  self.maze = mg.Maze(self.game, (20,10), 12345)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite Host.py", line 23, in __init__
  self.level = Level(self)
File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite Host.py", line 56, in <module>
  Game()
```

Problem 1:

The maze's constructor calls to generate the sprites, but the sprite's constructors are dependent on the maze already existing:

```
class Renderable_Sprite(pg.sprite.Sprite):
    def __init__(self, game, start_pos=(0,0), start_rot=0):
        super().__init__()
```

```
# initialise
self.game = game
self.maze = self.game.level.maze
self.camera = self.game.level.camera
self.pos = vec2(start_pos)
self.rot = start_rot

# animations:
self.imgs = []
self.frame_index = 0
self.frame_time = 100
self.frame_countdown = 0
```

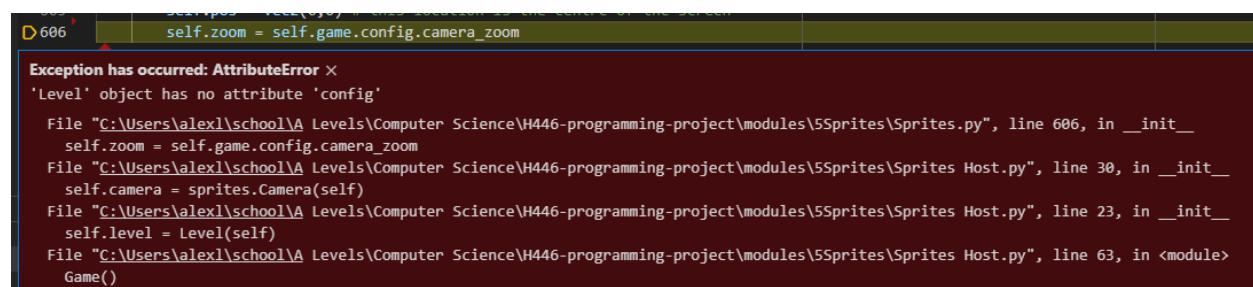
Solution 1: have the maze constructor be much simpler, just initialising key variables, then use a setup function to finish the setup. This way the maze exists by the time the sprites try to reference it, removing circular dependencies.

```
self.maze = mg.Maze(self.game, (20,10), 12345)
self.player = sprites.Player(self)
self.camera = sprites.Camera(self, self.player)

# populate the maze
self.maze.populate()
```

there was a similar circular dependency between the player and camera, and that has now been resolved in a very similar way, setting the target for the camera in a separate method after the camera has been initialised

Error 2:



D 606 self.zoom = self.game.config.camera_zoom

Exception has occurred: AttributeError
'Level' object has no attribute 'config'
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 606, in __init__
 self.zoom = self.game.config.camera_zoom
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 30, in __init__
 self.camera = sprites.Camera(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 23, in __init__
 self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 63, in <module>
 Game()

Problem 2:

the level is being passed in as an argument instead of the whole game object

```
class Level():
    def __init__(self, game):
        self.game = game
        # initialise camera
```

```
    self.camera = sprites.Camera(self)
```

Solution 2:

Pass in the game object, not the level object:

```
class Level():
    def __init__(self, game):
        self.game = game
        # initialise camera
        self.camera = sprites.Camera(self.game)
```

Error 3:

```
10  class Renderable_Sprite(pg.sprite.Sprite):
11      def __init__(self, game, start_pos=(0,0), start_rot=0):
12          super().__init__()
13
14          # initialise
15          self.game = game
D 16          self.camera = self.game.level.camera

Exception has occurred: AttributeError
'Game' object has no attribute 'level'

File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 16, in __init__
    self.camera = self.game.level.camera
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 483, in __init__
    super().__init__(game, start_pos)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 34, in wall_gen
    return sprites.Wall(self.game, pos)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 111, in wall_gen_group
    wall = wall_gen(start_pos)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 125, in layout_to_board
    board[0][x] = wall_gen_group((x, 0))
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 35, in setup
    self.layout_to_board(wall_gen)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite_Host.py", line 35, in __init__
    self.maze.setup()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite_Host.py", line 23, in __init__
    self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite_Host.py", line 63, in <module>
    Game()
```

Problem 3:

The sprites generated in the level constructor are dependent on the level already existing

```
class Level():
    def __init__(self, game):
        self.game = game
        # initialise camera
        self.camera = sprites.Camera(self.game)

        # start setting up the maze
        self.maze = mg.Maze(self.game, (20,10), 12345)
        # finishes generating the maze and sprites
        self.maze.setup()
        # initialise player
```

```
    self.player = sprites.Player(self, self.maze.start)
    # set what the camera should follow
    self.camera.set_target(self.player)
```

Solution 3:

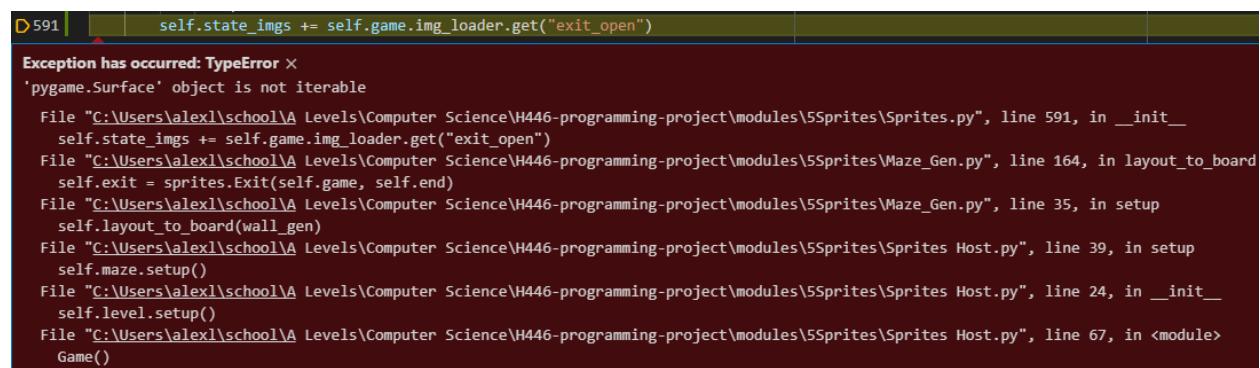
separate out sprite initialisation into a separate setup method so that the level already exists by the time the sprites are generated:

```
class Level():
    def __init__(self, game):
        self.game = game

    def setup(self):
        """sets up the level"""
        # initialise camera
        self.camera = sprites.Camera(self.game)

        # start setting up the maze
        self.maze = mg.Maze(self.game, (20,10), 12345)
        # finishes generating the maze and sprites
        self.maze.setup()
        # initialise player
        self.player = sprites.Player(self, self.maze.start)
        # set what the camera should follow
        self.camera.set_target(self.player)
```

Error 4:



```
D 591 |     self.state_imgs += self.game.img_loader.get("exit_open")
Exception has occurred: TypeError x
'pygame.Surface' object is not iterable
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 591, in __init__
    self.state_imgs += self.game.img_loader.get("exit_open")
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 164, in layout_to_board
    self.exit = sprites.Exit(self.game, self.end)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 35, in setup
    self.layout_to_board(wall_gen)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 39, in setup
    self.maze.setup()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 24, in __init__
    self.level.setup()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 67, in <module>
    Game()
```

Problem 4:

Incorrectly appending to a list:

```
    self.state_imgs = []
```

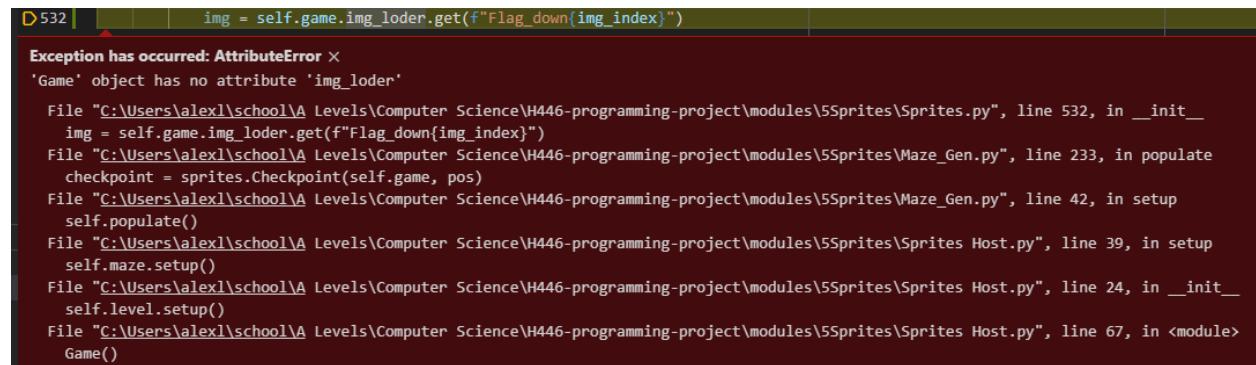
```
for img_index in range(0,7):
    self.state_imgs.append(
        self.game.img_loader.get(f"exit_locked{img_index}")
    )
self.state_imgs += self.game.img_loader.get("exit_open")
```

Solution 4:

Correctly append to a list:

```
self.state_imgs.append(self.game.img_loader.get("exit_open"))
```

Error 5:



```
D 532 |     img = self.game.img_loder.get(f"Flag_down{img_index}")
Exception has occurred: AttributeError
'Game' object has no attribute 'img_loder'
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 532, in __init__
    img = self.game.img_loder.get(f"Flag_down{img_index}")
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 233, in populate
    checkpoint = sprites.Checkpoint(self.game, pos)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 42, in setup
    self.populate()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite Host.py", line 39, in setup
    self.maze.setup()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite Host.py", line 24, in __init__
    self.level.setup()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprite Host.py", line 67, in <module>
    Game()
```

Problem 5:

Can't spell loader correctly:

```
img = self.game.img_loder.get(f"Flag_down{img_index}")
```

Solution 5:

Spell loader correctly:

```
img = self.game.img_loader.get(f"Flag_down{img_index}")
```

Error 6:

```
D 388 | self.colour = rng.choice(0,5)

Exception has occurred: TypeError
x
Random.choice() takes 2 positional arguments but 3 were given
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 388, in __init__
    self.colour = rng.choice(0,5)
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 244, in populate
    enemy = sprites.Enemy(self.game, pos)
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 42, in setup
    self.populate()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 39, in setup
    self.maze.setup()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 24, in __init__
    self.level.setup()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 67, in <module>
    Game()
```

Problem 6:

Wrong random method chosen:

```
self.colour = rng.choice(0,5)
```

Solution 6:

Choose correct random method:

```
self.colour = rng.randint(0,5)
```

Error 7:

```
D 397 | self.left_imgs = [[pg.transform.flip(img, True, False)]

Exception has occurred: TypeError
x
'pygame.Surface' object is not iterable
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 397, in <listcomp>
    self.left_imgs = [[pg.transform.flip(img, True, False)]
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 397, in __init__
    self.left_imgs = [[pg.transform.flip(img, True, False)]
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 244, in populate
    enemy = sprites.Enemy(self.game, pos)
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Maze_Gen.py", line 42, in setup
    self.populate()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 39, in setup
    self.maze.setup()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 24, in __init__
    self.level.setup()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites_Host.py", line 67, in <module>
    Game()
```

Problem 7:

Right_imgs isn't be initialised correctly: instead of a 2x6 array, it is starting as a 1x12 array:

```
self.right_imgs = [self.game.img_loader.get(f"enemy_{colour}2")
                  for colour in colour_names] +
                  [self.game.img_loader.get(f"enemy_{colour}3")
                   for colour in colour_names]
```

```
✓ right_imgs: [<Surface(100x100x32 SW) at 0x10000>]
  > special variables
  > function variables
  > 00: <Surface(100x100x32 SW) at 0x10000>
  > 01: <Surface(100x100x32 SW) at 0x10000>
  > 02: <Surface(100x100x32 SW) at 0x10000>
  > 03: <Surface(100x100x32 SW) at 0x10000>
  > 04: <Surface(100x100x32 SW) at 0x10000>
  > 05: <Surface(100x100x32 SW) at 0x10000>
  > 06: <Surface(100x100x32 SW) at 0x10000>
  > 07: <Surface(100x100x32 SW) at 0x10000>
  > 08: <Surface(100x100x32 SW) at 0x10000>
  > 09: <Surface(100x100x32 SW) at 0x10000>
  > 10: <Surface(100x100x32 SW) at 0x10000>
  > 11: <Surface(100x100x32 SW) at 0x10000>
  len(): 12
```

Solution 7:

fix right_imgs initialisation so that it is a 2x6 array

```
self.right_imgs = [self.game.img_loader.get(f"enemy_{colour}2")
                   for colour in colour_names], \
                   [self.game.img_loader.get(f"enemy_{colour}3")
                   for colour in colour_names]
```

```
✓ right_imgs: ([<Surface(100x100x32 SW) at 0x10000>, <Surface(100x100x32 SW) at 0x10000>], [<Surface(100x100x32 SW) at 0x10000>, <Surface(100x100x32 SW) at 0x10000>])
  > special variables
  > function variables
  > 0: [<Surface(100x100x32 SW) at 0x10000>, <Surface(100x100x32 SW) at 0x10000>]
  > 1: [<Surface(100x100x32 SW) at 0x10000>, <Surface(100x100x32 SW) at 0x10000>]
```

Error 8:

```
D 16 |     self.camera = self.game.level.camera
Exception has occurred: AttributeError
'Level' object has no attribute 'level'
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 16, in __init__
  self.camera = self.game.level.camera
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 41, in setup
  self.player = sprites.Player(self, self.maze.start)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 24, in __init__
  self.level.setup()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 67, in <module>
  Game()
```

Problem 8:

Player was passed level object instead of game object

```
    self.player = sprites.Player(self, self.maze.start)
```

```
> game: <__main__.Level object at 0x0000...
```

Solution 8:

Pass player the correct game object argument

```
    self.player = sprites.Player(self.game, self.maze.start)
```

Error 9:

```
D 594 | keys = self.game.level.player.keys

Exception has occurred: AttributeError ×
'Player' object has no attribute 'keys'

  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-
project\modules\5Sprites\Sprites.py", line 594, in update
    keys = self.game.level.player.keys
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-
project\modules\5Sprites\Sprites Host.py", line 56, in loop
    self.maze.all_sprites.update(dt)
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-
project\modules\5Sprites\Sprites Host.py", line 25, in __init__
    self.level.loop()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-
project\modules\5Sprites\Sprites Host.py", line 67, in <module>
    Game()
```

Problem 9:

The constructor for the player isn't being defined correctly(it's missing an underscore), so instead the parent constructor is being called, and this doesn't define some attributes such as keys

```
class Player(Renderable_Sprite):
    def __init__(self, game, start_pos):
```

Solution 9:

Correctly define the player constructor

```
class Player(Renderable_Sprite):
    def __init__(self, game, start_pos):
```

Error 10:

```
D 595 |     if self.keys == 6:  
  
Exception has occurred: AttributeError ×  
'Exit' object has no attribute 'keys'  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 595, in update  
    if self.keys == 6:  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 56, in loop  
    self.maze.all_sprites.update(dt)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 25, in __init__  
    self.level.loop()  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 67, in <module>  
    Game()
```

Problem 10:

The code tries to access player.keys, but is accessing Exit.keys instead:

```
def update(self, dt):  
    keys = self.game.level.player.keys  
    if self.keys == 6:
```

Solution 10:

Access player.keys correctly:

```
def update(self, dt):  
    keys = self.game.level.player.keys  
    if keys == 6:  
        self.imgs = self.state_imgs[-2:]  
    else:  
        self.imgs = self.state_imgs[keys]
```

Error 11:

```
D 537 |     if (self.pos - self.game.level.player.pos).length < 0.5:  
  
Exception has occurred: TypeError ×  
'<' not supported between instances of 'builtin_function_or_method' and 'float'  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 537, in update  
    if (self.pos - self.game.level.player.pos).length < 0.5:  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 56, in loop  
    self.maze.all_sprites.update(dt)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 25, in __init__  
    self.level.loop()  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 67, in <module>  
    Game()
```

Candidate Name: Alexander Mills Candidate Number: 9120

Problem 11:

Vector's length function not called:

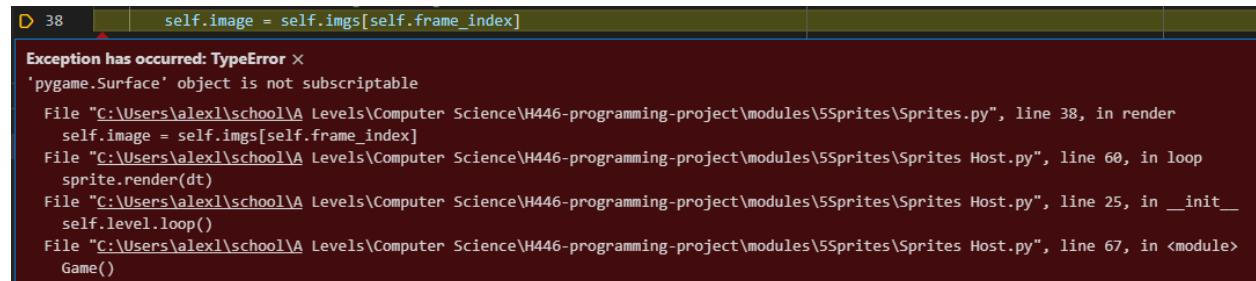
```
if (self.pos - self.game.level.player.pos).length < 0.5:
```

Solution 11:

Call vector's length function

```
if (self.pos - self.game.level.player.pos).length() < 0.5:
```

Error 12:



```
D 38     self.image = self.imgs[self.frame_index]

Exception has occurred: TypeError
'pygame.Surface' object is not subscriptable
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 38, in render
    self.image = self.imgs[self.frame_index]
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 60, in loop
    sprite.render(dt)
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 25, in __init__
    self.level.loop()
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 67, in <module>
    Game()
```

Problem 12:

Img attribute contains a surface, rather than a list containing a surface

```
class Wall(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        # initialise assets
        self.imgs = game.img_loader.get("brick dark grey")
```

Solution 12:

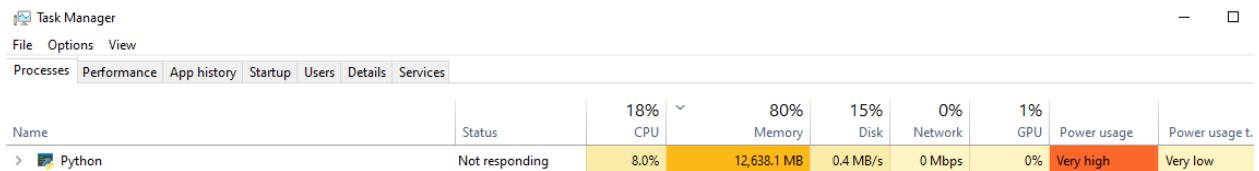
Ensure imgs attribute is a list

```
class Wall(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        # initialise assets
        self.imgs = [game.img_loader.get("brick dark grey")]
```

Error 13:

Game has a massive memory leak that fills up the entire computer's memory in under 30 seconds:



Problem 13:

The Asset loader fails to load any images, this means that every image is a default 100x100 purple square. The camera zoom is also set to 100 this results in images which are 10000x10000 pixels, which uses up **all** the RAM

```
self.camera_zoom = 100

# sprite cant be found
if not(image):
    image = pg.surface.Surface((100, 100)).convert_alpha()
    image.fill((255, 0, 255))
```

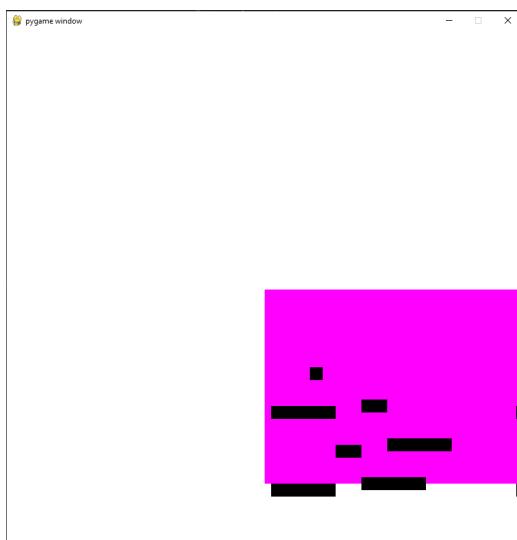
Solution 13: reduce camera zoom to 10 and reduce texture load fail square to 10x10

```
self.camera_zoom = 10

# sprite cant be found
if not(image):
    image = pg.surface.Surface((10, 10)).convert_alpha()
    image.fill((255, 0, 255))
```

Error 14:

Purple squares loaded instead of textures:



Candidate Name: Alexander Mills Candidate Number: 9120

Problem 14:

The asset loaders can't find the assets as the img path is incorrectly specified

```
self.img_path = 'img'  
self.snd_path = 'snd'
```

```
▼ Locals  
> game: <__main__.Game object at 0x0000026C02322E60>  
> img_path: WindowsPath('img')  
  img_path_name: 'img'  
> self: <Asset_Loader.Img_Loader object at 0x0000026C02321270>  
> Globals
```

Solution 14:

Give the asset loaders the correct path to load from:

```
self.img_path = (Path(__file__).parent / 'img').as_posix()  
self.snd_path = (Path(__file__).parent / 'snd').as_posix()
```

```
▼ Locals  
> game: <__main__.Game object at 0x0000016692AF2F20>  
> img_path: WindowsPath('C:/Users/alex1/school/A Levels/Computer Science/H446-programming-project/modules/5Sprites/img')  
> self: <Asset_Loader.Img_Loader object at 0x0000016692AF1300>  
> Globals
```

Error 15:

```
D 108 |     xml_tree_root = ET.parse(xml_path.as_posix()).getroot()  
  
Exception has occurred: ParseError X  
no element found: line 226, column 14  
  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
  project\modules\5Sprites\Asset_Loader.py", line 108, in load_xml  
    xml_tree_root = ET.parse(xml_path.as_posix()).getroot()  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
  project\modules\5Sprites\Asset_Loader.py", line 104, in __init__  
    self.load_xml(xml_path)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-  
  project\modules\5Sprites\Asset_Loader.py", line 18, in __init__  
    self.sprite_sheets.append(Sprite_Sheet(game, file_name[:-4]))  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites  
  Host.py", line 19, in __init__  
    self.img_loader = al.Img_Loader(self)  
  File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites  
  Host.py", line 67, in <module>  
    Game()
```

Problem 15:

Poorly formatted xml in one of the sprite sheets

Candidate Name: Alexander Mills Candidate Number: 9120

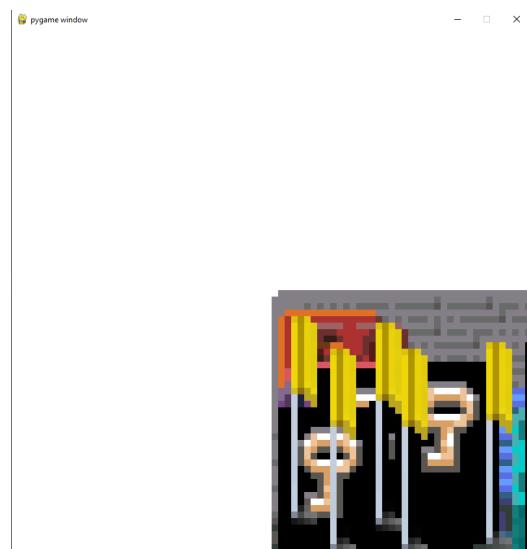
```
<Subtexture name="rampU dark grey" x="238" y="435" width="16" height="24" />
<TextureAtlas>
```

Solution 15:

Fix xml formatting:

```
<Subtexture name="rampU dark grey" x="238" y="435" width="16" height="24" />
</TextureAtlas>
```

Error 16:



Problem 16:

The camera doesn't convert world coordinates to screen coordinates correctly; it assumes the sprites are 1 pixel in size:

```
scaled_wrld_coord = vec2(wrld_coord) * self.zoom
```

Solution 16:

Multiply the coordinates by 16, as all sprites are 16 pixels wide

```
scaled_wrld_coord = vec2(wrld_coord) * self.zoom * 16
```

pygame window - □ ×



Now that the code can start and runs without instantly crashing and appears to work correctly, it can be tested on the tests set out be the design stage.

Test table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior	Resultant behavior	Pass / Fail + solution
1	Player – arrow inputs	Arrow keys are pressed	Valid	Player moves in the corresponding direction		<p>player is missing from the maze</p> <p>Fail: Player is not drawn Problem 1: Player and camera aren't part of maze.all_sprites, so aren't updated, rendered and drawn with everything else:</p> <pre># call all sprites render method for sprite in self.maze.all_sprites: sprite.render(dt) self.game.screen.fill((255,255,255)) self.maze.all_sprites.draw(self.game.screen)</pre> <p>Solution 1: update render and draw player and camera separately:</p> <pre># call all sprites render method for sprite in self.maze.all_sprites: sprite.render(dt) self.player.render(dt) self.game.screen.fill((255,255,255)) self.maze.all_sprites.draw(self.game.screen)</pre> <p>Problem 2:</p> <p>D 192 hits = pg.sprite.spritecollide(self, self.maze.keys, True)</p> <pre>Exception has occurred: AttributeError 'Player' object has no attribute 'rect' File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 192, in update hits = pg.sprite.spritecollide(self, self.maze.keys, True) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 58, in loop self.player.update(dt) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 25, in __init__ self.level.loop() File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 70, in <module> Game()</pre> <p>The player's sprite is only initialized once it has been rendered at least once. Solution 2: render the player once before updating it</p> <pre># render once to set up rect for collisions self.walking = False self.render(0)</pre> <p>Problem 3:</p> <p>D 359 image = self.imgs[self.frame_index][self.colour]</p> <pre>Exception has occurred: TypeError 'pygame.Surface' object is not subscriptable File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 359, in render image = self.imgs[self.frame_index][self.colour] File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 136, in __init__ self.render(0) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 41, in setup self.player = sprites.Player(self.game, self.maze.start) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 24, in __init__ self.level.setup() File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 70, in <module> Game()</pre> <p>Player's images aren't being stored correctly after they are loaded</p> <pre>self.standing_imgs = { (1,0) : right_imgs[0],</pre>

```
(-1,0) : left_imgs[0],  
(0,1) : down_imgs[0],  
(0,-1) : up_imgs[0]  
}  
self.walking_imgs = { (1,0) : right_imgs,  
(-1,0) : left_imgs,  
(0,1) : down_imgs,  
(0,-1) : up_imgs  
}
```

Solution 3: store ensure all the images are still in arrays with the same number of dimensions as they started with

```
self.standing_imgs = { (1,0) : [right_imgs[0]],  
(-1,0) : [left_imgs[0]],  
(0,1) : [down_imgs[0]],  
(0,-1) : [up_imgs[0]]  
}  
self.walking_imgs = { (1,0) : right_imgs,  
(-1,0) : left_imgs,  
(0,1) : down_imgs,  
(0,-1) : up_imgs  
}
```

Problem 4:

```
358 | # retrieve correct img from imgs  
D 359 | image = self.imgs[self.frame_index][self.colour]  
  
Exception has occurred: TypeError x  
list indices must be integers or slices, not tuple  
...  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 359, in render  
    image = self.imgs[self.frame_index][self.colour]  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 136, in __init__  
    self.render(0)  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 41, in setup  
    self.player = sprites.Player(self.game, self.maze.start)  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 24, in __init__  
    self.level.setup()  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 70, in <module>  
    Game()
```

The player's colour is stored as a tuple (the rgb value) rather than a colour index

```
self.colour = game.config.colours[0]
```

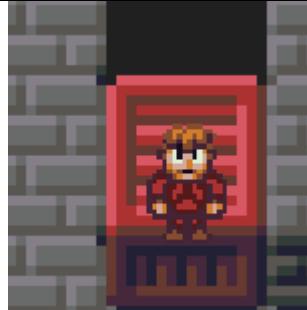
Solution: set player's colour to a colour index instead of an rgb value

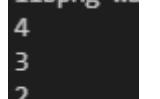
```
self.colour = 0
```

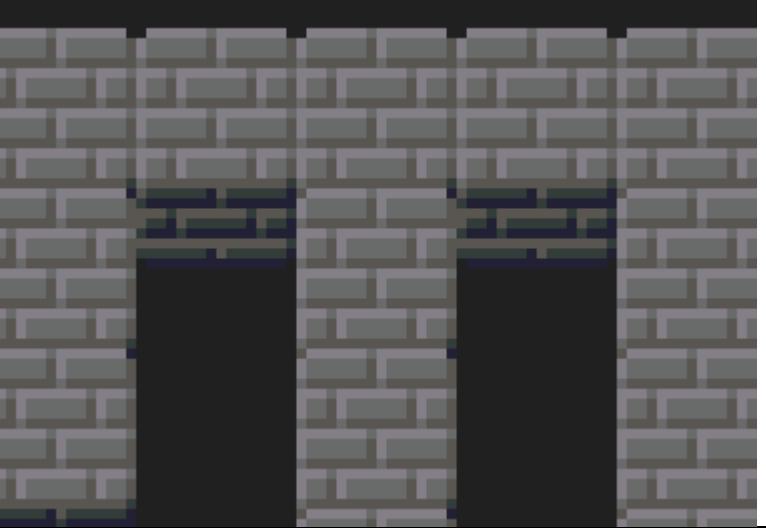
Problem 5:

						<pre> 195 # collisions with keys D196 hits = pg.sprite.spritecollide(self, self.maze.keys, True) Exception has occurred: AttributeError 'Key' object has no attribute 'rect' File "C:\Users\alex\school\A_ Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites.py", line 196, in update hits = pg.sprite.spritecollide(self, self.maze.keys, True) File "C:\Users\alex\school\A_ Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 58, in loop self.player.update(dt) File "C:\Users\alex\school\A_ Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 25, in __init__ self.level.loop() File "C:\Users\alex\school\A_ Levels\Computer Science\H446-programming-project\modules\5Sprites\Sprites Host.py", line 70, in <module> Game() </pre> <p>All sprites have their rect initialized when rendered, which is before they are updated this means the first frame they have no rect to use for updates such as</p> <p>collisions:</p> <pre> # set rect position correctly self.rect = self.image.get_rect() screen_pos = self.camera.wrld_2_scrn_coord(self.pos) self.rect.topleft = screen_pos </pre> <p>Solution 5: Render all sprites once after they have been initialized but before they are updated by the main loop:</p> <pre> # initialise sprites in maze for sprite in self.maze.all_sprites: sprite.render(0) </pre>
2	Player – wasd inputs	Wasd keys are pressed	Valid	Player moves in the corresponding direction	Player moves in the corresponding direction	pass
3	Player – smooth movement	Wasd keys are pressed	Valid	Player accelerates up to speed when moving in a direction	Player accelerates up to speed, but doesn't decelerate in an axis if the other is still moving	<p>Fail: Problem: One walking variable is used to track whether both x and y are changing, but this should be done per axis</p> <p>Solution: implement deceleration for each axis separately:</p> <pre> # decelerate back to vel = 0 if not walking if not walking_x: self.vel.x -= min(self.acc * self.vel.x, self.vel.x*0.9, key = lambda x: abs(x)) if not walking_y: self.vel.y -= min(self.acc * self.vel.y, self.vel.y*0.9, key = lambda x: abs(x)) self.walking = walking_x or walking_y </pre>
4	Player – standing animation	No keys are pressed	Valid	Player displays the standing animation	<p>Player stands there with walking animation:</p> 	Pass

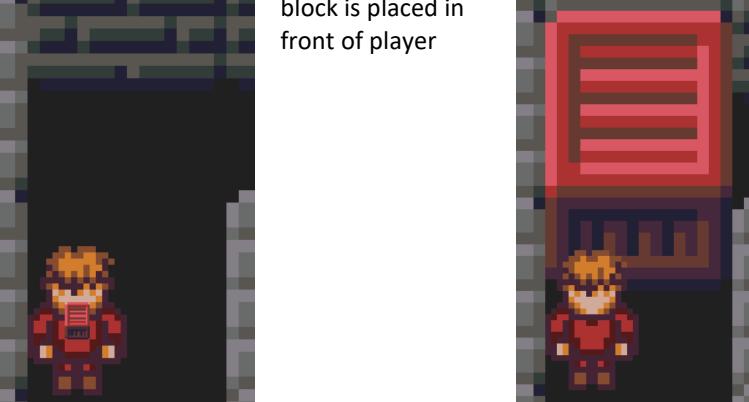
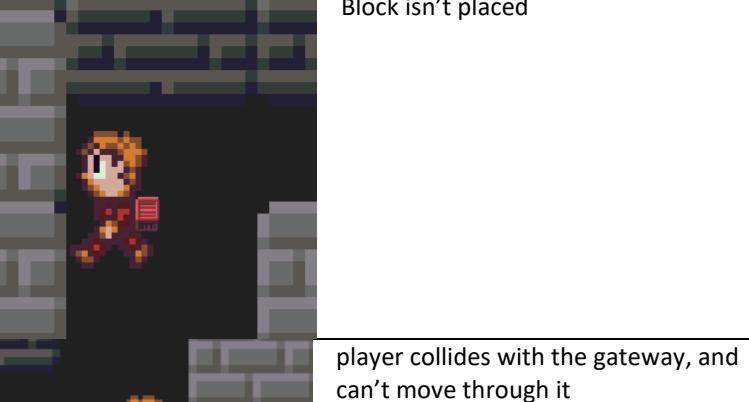
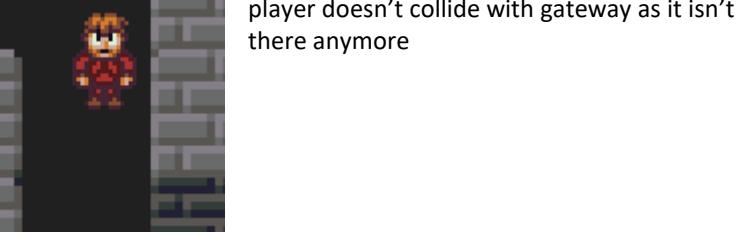
5	Player – walking animation	Wasd or arrow keys are pressed	Valid	Player displays walking animation	Player goes through walking animation while walking: 	<p>Fail : Problem 1:</p> <pre>D 360 self.image = self.imgs[self.frame_index][self.colour]</pre> <p>Exception has occurred: IndexError × list index out of range File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites.py", line 359, in render self.image = self.imgs[self.frame_index][self.colour] File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites Host.py", line 68, in loop self.player.render(dt) File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites Host.py", line 25, in __init__ self.level.loop() File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites Host.py", line 75, in <module> Game() <p>Transitioning back to standing animation sometimes results in frame_index being 1 when using standing imgs, which causes an index error Solution 1: set frame index to zero when transitioning back</p> <pre>self.imgs = self.standing_imgs[tuple(facing_dir)] self.frame_index = 0</pre> </p>
6	Player – orientation	Wasd or arrow keys are pressed	Valid	Player turns around to face in whichever direction they are moving	Player faces in the direction they are moving	Pass
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		<p>Fail: Problem: wall collisions aren't registering because the type checking isn't correct:</p> <pre>if collide_hit_rect(sprite, self) and (type(sprite) == "Wall" or type(sprite) == "Block" and sprite.colour != self.colour or type(sprite) == "Exit" and self.keys != 6):</pre> <p>Solution: Correctly check the type of sprite the player is colliding with:</p> <pre>if collide_hit_rect(sprite, self) and (type(sprite).__name__ == "Wall" or (type(sprite).__name__ == "Block" and sprite.colour != self.colour) or type(sprite).__name__ == "Exit" and self.keys != 6):</pre>
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	Player collides with the sides of walls that aren't exposed:	<p>Fail: Problem: collisions don't care if the side is exposed or not</p> <pre># wall on left self.vel.x = max(0, self.vel.x)</pre> <p>Solution check if the side is exposed before colliding with it</p> <pre>spot = sprite.pos//1 + vec2(1,0) # collide if outside the bounds or isn't a collideable if not(0 <= spot.x < self.maze.bsize[0]) or \ not(check_collidable(board[int(spot.y)][int(spot.x)])): self.vel.x = max(0, self.vel.x)</pre>

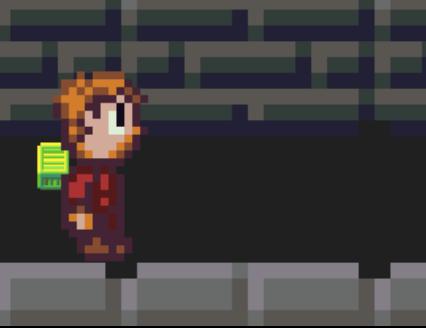
9	Player – colour changing	Number keys 1 to 6	Valid	Player sprite changes to reflect the current colour		player changes colour corresponding to which key is pressed	Pass
10	Player – no collisions with blocks of the same colour	Player is moved towards a block who's colour equals that of the player	Valid	Player passes through block		player passes through the red block	pass
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		player doesn't pass through the red block	pass
12	Enemies – navigation	Enemy spawned in the maze	Valid	Enemy moves towards an open space in the maze	Game locks up: it is stuck in an infinite loop	<p>Fail: Problem: enemy can't choose a target location they can navigate to:</p> <pre>def get_path(self): """calculates a new path for this sprite to follow""" while True: try: # find new destination to try to go to destination = self.maze.random_board_spot() while self.maze.board[destination[1]][destination[0]] != False: destination = self.maze.random_board_spot() # find path to this location self.target_path = self.maze.get_shortest_path(self.pos//1, destination) except: # failed to find a location that can be navigated to, try again pass</pre> <p>Solution: Select a location using the maze's branch method: then the location will be accessible</p>	

13	Enemies – smooth movement	Enemy spawned in the maze	Valid	As enemy goes round turns, it follows a smooth path, not jittering around		enemy smoothly goes round the corner	Pass
14	Enemies – navigation around maze	Enemy spawned in the maze	Valid	Enemy doesn't intersect with the maze as it is moving		enemy only intersects with top surfaces of some walls, but is always rendered under them: this is due to the orthographic viewpoint, so isn't a problem	Pass
15	Enemies – reallocating objective	Enemy has moved to target position	Valid	Enemy selects a new target and moves towards it	Enemy continuously moves around the maze, ever getting stuck in one location, so is successfully selecting new objectives		Pass
16	Enemies – attacking player	Player moves to same location as an enemy	Valid	Player health is decremented		player health is decremented	Pass
17	Enemies – attack cooldown	Player moves to same location as an enemy	Valid	Player health is decremented at regular intervals, not every frame		player health is decremented every 500 ms	pass
18	Enemies – walking animation	Enemy is spawned in the maze	Valid	Enemy animates as it moves along		enemy animates as it goes along	Pass
19	Enemies - orientation	Enemy is spawned in the maze	Valid	Enemy turns such that it points in the direction it moves		enemy is facing in the direction it is going	pass

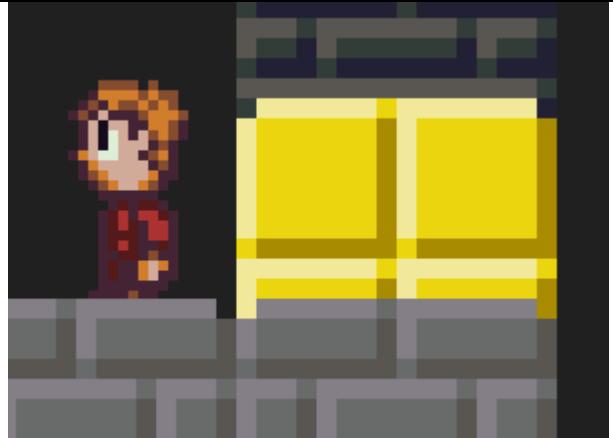
20	Walls - animation	Walls are spawned in the maze	Valid	Wall sprite displays it's animation on screen	Walls are seen in the maze: 	Pass
21	Walls – collisions with player	Player is moved to collide with the wall	Valid	Player can collide with multiple walls and walls are unaffected	Walls don't move during collision	Pass
22	Blocks – colours	Block is spawned in the maze	Valid	Blocks have colours randomly selected from all colours	 block colours are randomly selected	pass
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	Nothing happens	Fail: Problem 1: edge detection doesn't work <code>if slot_keys[slot_index] and not slot_keys[slot_index]:</code> Solution 1: fix edge detection code <code>if slot_keys[slot_index] and not self.prev_slot_keys[slot_index]:</code> Problem 2: it isn't correctly checking if there is a block in front of the player: <code>if type(facing_sprite) == "Block":</code> Solution 2; check type's name is equal to block instead: <code>if type(facing_sprite).__name__ == "Block":</code>
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	Block is picked up into player's right inventory slot	Pass

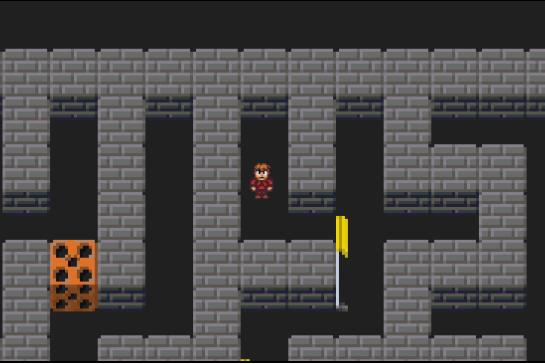
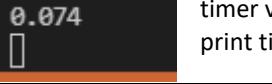
25	Blocks – rendering in player inventory	Blocks stored in both slots of player's inventory	Valid	Blocks are rendered on player's backpack, with colours clearly visible		inventory not visible	<p>Fail: Problem 1: Positioning code doesn't work correctly, so the sprite position isn't in a visible location</p> <pre># no image if facing down if facing_dir != (0,1): if block_1 := self.inventory[0]: block_1_image = pg.transform.scale(block_1.image, (16, 24)) block_1_pos = self.camera.wrld_2_scrn_coord(b1_pos[tuple(facing_dir)]) self.image.blit(block_1_image, block_1_pos) if block_2 := self.inventory[1]: block_2_image = pg.transform.scale(block_2.image, (16, 24)) block_2_pos = self.camera.wrld_2_scrn_coord(b2_pos[tuple(facing_dir)]) self.image.blit(block_2_image, block_2_pos)</pre> <p>Solution 1: correctly calculate the position of the sprite</p> <pre>if facing_dir != (0,1): # work out the position and size of each block's image if block_1 := self.inventory[0]: block_1_image = pg.transform.scale(block_1.image, vec2(block_1.image.get_size())//8) block_1_pos = b1_pos[tuple(facing_dir)] * self.camera.zoom if block_2 := self.inventory[1]: block_2_image = pg.transform.scale(block_2.image, vec2(block_2.image.get_size())//8) block_2_pos = b2_pos[tuple(facing_dir)] * self.camera.zoom</pre> <p>Problem 2: The player renders with a black box around them because the transparency is missing as the sprite it is rendered onto doesn't have per pixel transparency:</p> <pre>image = pg.surface.Surface(self.image.get_size())</pre> <p>Solution 2: Enable per pixel transparency for the surface:</p> <pre>image = pg.surface.Surface(self.image.get_size(), flags = pg.SRCALPHA)</pre>
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			Pass

						Block is placed in front of the player	
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		block is placed in front of player	
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		Block isn't placed	
29	Gateway – collisions when closed	player moves up to edge of a gateway	Valid	Player can't move through the gateway		player collides with the gateway, and can't move through it	
30	Gateway – collisions when open	Player moves towards edge of a gateway	Valid	Player can move through gateway		player doesn't collide with gateway as it isn't there anymore	

31	Gateway – opening with correct block	Player places block of correct colour in a gateway	Valid	Gateway changes from closed state to open state	Block can't be placed on gateway	<p>Fail: Problem 1: Placing can't detect if it is trying to place on a gateway due to incorrect type checking: <code>if type(facing_sprite) == "Gateway"</code></p> <p>Solution 1: correctly get the name of the type it is trying to place on <code>if type(facing_sprite).__name__ == "Gateway"</code></p> <p>Problem 2: the block can only be placed on a gateway of the same colour as the player <code>if type(facing_sprite).__name__ == "Gateway" and \ facing_sprite.colour == self.colour:</code></p> <p>solution 2: check of the gateway is the same colour as the block, not the same colour as the player <code>if type(facing_sprite).__name__ == "Gateway" and \ facing_sprite.colour == self.inventory[slot_index].colour:</code></p>	
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		Gateway doesn't open	Pass
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		gateway disappears	pass
34	Checkpoint – player detection	Player walks past checkpoint	Valid	The checkpoint activates	Game crashes	<p>Fail: Problem: <code>D 663 self.game.level.player.last_checkpoint.deactivate()</code></p> <p>Exception has occurred: AttributeError × 'bool' object has no attribute 'deactivate' File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites\Sprites.py", line 663, in update self.game.level.player.last_checkpoint.deactivate() File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites\Sprites Host.py", line 68, in loop self.maze.all_sprites.update(dt) File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites\Sprites Host.py", line 25, in __init__ self.level.loop() File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\SSprites\Sprites Host.py", line 80, in <module> Game()</p> <p>If the player has no previous checkpoint, then there is no checkpoint to deactivate: Solution: check if there is a previous checkpoint before deactivating it <code>if last_checkpoint := self.game.level.player.last_checkpoint: last_checkpoint.deactivate()</code></p>	

35	Checkpoint – visual indication of activation	Player walks past checkpoint	Valid	The checkpoint sprite changes to visually show that it has been activated		checkpoint sprite becomes a waving flag	pass
36	Checkpoint – respawning	Player dies	Valid	Player respawns at the checkpoint, not at start		Checkpoint is now “stuck” to the player and follows it around	<p>Fail: Problem: vectors are mutable objects, so this means that the player and checkpoint end up with the same memory location storing both of their positions</p> <pre>if self.last_checkpoint != False: self.pos = self.last_checkpoint.pos</pre> <p>solution: copy position before assignment to break link between them</p> <pre>if self.last_checkpoint != False: self.pos = vec2(self.last_checkpoint.pos)</pre>
37	Second checkpoint – activation	One checkpoint is activated, then another, and then the player dies	Valid	Player respawns at most recently activated checkpoint	Player teleports back to most recent checkpoint, and starts with full health again		Pass
38	Key - rendering	Key is spawned	Valid	Key animates, moving up and down on the spot		key animates ,but is surrounded by a black box	<p>Fail: Problem: the surface the key is blitted onto doesn't support per pixel transparency:</p> <pre>displaced_img = pg.surface.Surface(key_image.get_size() + vec2(0,key_displacement))</pre> <p>solution enable per pixel transparency for the surface:</p> <pre>displaced_img = pg.surface.Surface(key_image.get_size() + vec2(0,key_displacement), flags = pg.SRCALPHA)</pre>
39	Key - collection	Player is close to key	Valid	Key disappears and the player's key count increases	Key disappears, key collection sound is played and key count increases		Pass

40	Exit – player wins	Player is close to the exit with all keys	Valid	Level closes and end screen shows		when player has all keys, the exit opens	Pass
41	Exit – player doesn't have all the keys	Player is close to exit without all keys	Valid	Level keeps playing		Level keeps playing	Pass
42	Camera – player tracking	Player moves around	Valid	Camera moves around, following player		camera follows the player round the maze	pass
43	Camera – positioning on screen	Player moves around	Valid	Camera follows player such that they are located centrally on screen		Player is 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		Edge of screen can be seen:	<p>Fail: Problem: the edge of the screen positioning calculations don't work; they don't account for the fact that sprites are 16x16 pixels, not 1x1</p> <pre>unscaled_scrn_size = vec2(self.game.config.resolution) / self.zoom</pre> <p>Solution: Account for sprite dimensions:</p> <pre>unscaled_scrn_size = vec2(self.game.config.resolution) / self.zoom / 16</pre>
45	Timer – initialization	timer object is created	Valid	timer object created, with start time set to system time, without crashing	Timer object created and starts to count	Pass	
46	Timer – reset	Timer is reset	Valid	Timer sets start time to current system time and total time is 0	 timer value goes back to zero (+ time it takes to print time)	Pass	
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	 The timer is accurate; the time it contains is equal to the time it has been going (+ the small amount of error I've introduced)	Pass	

STAGE 5: STAKEHOLDER REVIEW:

Now that there are the fundamentals of the game, with all the intended gameplay and most of the finished graphics, it must be reviewed by the stakeholders to ensure it still meets the success criteria. I have taken the files for this module and sent them in a zip folder to both the stakeholders and asked them for their opinions.

Máté's feedback is very positive, complimenting the overall aesthetic design of the sprites and the overall gameplay. His main point of criticism is about the player movement controller; it is only designed to support 4 way movement, but he has suggested it could make the game more playable if it supported 8 way movement. This can be implemented by replacing an elif with an if in the player movement code. He has also found a slight bug with the collisions code that means that if the player is up against a wall, then moves laterally away from the wall off a protruding corner, they can't laterally move back to this position. This is due to the collision code only cancelling the player velocity the frame after collision, so they slightly move into the wall before stopping. This is quite hard to fix as it would require setting the player's position correctly so that they are adjacent to the wall without intersecting it at all, while not getting the player caught on the wall., so this bug will have to remain as I don't have the time to fix it.

Ben has also identified the bug in the collisions code as well, meaning this could be a significant problem getting in the way of the game being playable. He also suggests that the player's colour should be selectable by scroll wheel, not just number keys, which will make the game much more playable, so that will be implemented.

As the code has passed all the tests set out in the design section, it is complete and fully functional. Now that the stakeholders have also given their input and the code has been changed to meet their requests, it fully meets its success criteria. This means that this module is complete for now and can be set aside until integration.

STAGE 6: MENU SPRITES

The user interacts with the user interface through a set of standard UI elements, such as buttons, text entry boxes, sliders and static text. This functionality is implemented by a set of objects which can be instantiated and placed around the multiple screens that the player will navigate through, allowing their functionality to be reused. This also ensures there is a cohesive aesthetic to the UI as all elements are separate

STAGE 6: DEVELOPMENT GOALS

- A text sprite that can be rendered on the screen.
- An input box that can be typed into
- A toggle button that can be toggled true or false
- A slider that can be dragged to different values
- A spinner with 2 buttons that can be pressed to cycle between different options
- A button that can be pressed to cause things to happen
- All sprites rescale to fit the size of the screen

STAGE 6: DEVELOPMENT PROCESS

```
import pygame as pg
from pathlib import Path

k2c_numeric = { pg.K_0 : "0",
                pg.K_1 : "1",
                pg.K_2 : "2",
                pg.K_3 : "3",
                pg.K_4 : "4",
                pg.K_5 : "5",
                pg.K_6 : "6",
                pg.K_7 : "7",
                pg.K_8 : "8",
                pg.K_9 : "9",
            }
k2c_alpha = { pg.K_a : "a",
               pg.K_b : "b",
               pg.K_c : "c",
               pg.K_d : "d",
               pg.K_e : "e",
               pg.K_f : "f",
               pg.K_g : "g",
               pg.K_h : "h",
               pg.K_i : "i",
               pg.K_j : "j",
               pg.K_k : "k",
               pg.K_l : "l",
               pg.K_m : "m",
               pg.K_n : "n",
               pg.K_o : "o",
               pg.K_p : "p",
               pg.K_q : "q",
               pg.K_r : "r",
               pg.K_s : "s",
               pg.K_t : "t",
               pg.K_u : "u",
               pg.K_v : "v",
               pg.K_w : "w",
               pg.K_x : "x",
               pg.K_y : "y",
               pg.K_z : "z",
            }
```

```
class Text(pg.sprite.Sprite):
    def __init__(self, game, start_rect, text):
        # store text to attribute
        self.game = game
        self.text = text

        # get font name
        font_name = self.game.config.text_font_name
        font_path = Path(self.game.config.img_path) / font_name

        # load font
        self.font = pg.font.Font(font_path.as_posix())
        self.text_colour = self.game.config.text_colour

        # rescale to generate image
        self.rect = start_rect
        self.rescale()

    def update(dt, events):
        # empty function to ensure functionality
        pass

    def rescale(self):
        # render text to image
        text_img = self.font.render(self.text, False, self.text_colour)
        scale_factor = self.rect.height / text_img.get_height()
        text_size = [text_img.get_width() * scale_factor, self.rect.height]
        text_img = pg.transform.scale(text_img, text_size)
        text_img_rect = text_img.get_rect()

        # scale image to width and height of rect
        self.image = pg.surface.Surface(self.rect.size, keys=pg.SRCALPHA)
        text_img_rect.center = self.rect.center
        self.image.blit(text_img, text_img_rect.topleft)

class Input_Box(pg.sprite.Sprite):
    def __init__(self, game, start_rect, default_text, keys_to_chars ):
        # store attributes
        self.game = game
        self.default_text = default_text
        self.text = ""
        self.keys_to_chars = keys_to_chars
```

```
        self.selected = False

        # get font name
        font_name = self.game.config.text_font_name
        font_path = Path(self.game.config.img_path) / font_name

        # load font
        self.font = pg.font.Font(font_path.as_posix())
        self.text_colour = self.game.config.text_colour

        # rescale to generate image
        self.rect = pg.rect.Rect(start_rect)
        self.rescale()

    def update(self, dt, events):
        for event in events:
            # set selected to true if cursor in rect
            # and left mouse button clicked, otherwise false
            if event.type == pg.MOUSEBUTTONDOWN and event.key == 0:
                mouse_pos = pg.mouse.get_pos()
                if self.rect.collidepoint(*mouse_pos):
                    self.selected = True
                else:
                    self.selected = False
                self.rescale()

            # register key presses
            if event.type == pg.KEYDOWN:
                # if key is backspace, remove last char from text
                if event.key == pg.K_BACKSPACE:
                    self.text = self.text[:-1]
                # if key is in keys_to_chars, append to keys
                if event.key in self.keys_to_chars.keys():
                    self.text += self.keys_to_chars[event.key]
                self.rescale()

    def rescale(self):
        # if self.text isn't empty, render text
        if len(self.text) > 0:
            render_text = self.text
            # if self.selected, append _ to the end of the text to show this
            if self.selected:
                render_text += "_"
        # if self.text is empty, render default text
```

```
        else:
            render_text = self.default_text

            # render text to image
            text_img = self.font.render(render_text, False, self.text_colour)
            scale_factor = self.rect.height / text_img.get_height()
            text_size = [text_img.get_width() * scale_factor, self.rect.height]
            text_img = pg.transform.scale(text_img, text_size)
            text_img_rect = text_img.get_rect()

            # scale image to width and height of rect
            self.image = pg.surface.Surface(self.rect.size)
            text_img_rect.center = self.rect.center
            self.image.blit(text_img, text_img_rect.topleft)

class Toggle(pg.sprite.Sprite):
    def __init__(self, game, start_rect):
        self.game = game

        # initialise ticked to false
        self.ticked = False

        self.ticked_img = self.game.img_loader.get("toggle_ticked")
        self.unticked_img = self.game.img_loader.get("toggle_unticked")

        # init rect and render
        self.rect = pg.rect.Rect(start_rect)
        self.rescale()

    def update(self, dt, events):
        for event in events:
            if event.type == pg.MOUSEBUTTONDOWN and event.key == 0:
                self.ticked = not self.ticked
                self.rescale()

    def rescale(self):
        # if ticked, render the ticked image
        if self.ticked:
            self.image = pg.transform.scale(self.ticked_img, self.rect.size)
        # if not ticked, render unticked image
        else:
            self.image = pg.transform.scale(self.unticked_img, self.rect.size)
```

```
class Slider(pg.sprite.Sprite):
    def __init__(self, game, start_rect):
        self.game = game

        # initialise val to 0
        self.val = 0
        self.grabbed = False

        # load assets
        self.slider_img = self.game.img_loader.get("slider")
        self.thumb_img = self.game.img_loader.get("slider thumb")

        # initialise rect and render
        self.rect = pg.rect.Rect(start_rect)
        self.rescale()

    def update(self, dt, events):
        mouse_pos = pg.mouse.get_pos()

        for event in events:
            # if mouse is clicked and hovering over thumb rect, grab
            if event.type == pg.MOUSEBUTTONDOWN and event.button == 0:
                if self.scrn_thumb_rect.collidepoint(mouse_pos):
                    self.grabbed = True
            # if mouse is unclicked, release
            elif event.type == pg.MOUSEBUTTONUP and event.button == 0:
                self.grabbed = False

            # update val based of it is grabbed and mouse movement
            if self.grabbed:
                self.val = (self.mouse_pos - self.rect.left) / self.rect.width
                self.val = min(max(0, self.val), 1)
                self.rescale()

    def rescale(self):
        # blit slider image
        self.image = pg.surface.Surface(self.rect.size, flags = pg.SRCALPHA)
        pg.transform.scale(self.slider_img, self.rect.size, self.image)

        # find thumb collide rect
        self.scrn_thumb_rect = self.thumb_img.get_rect()
        self.scrn_thumb_rect.centre = \
            (self.rect.left + self.val * self.rect.width,
             self.rect.centery)
```

```
# blit thumb to image
thumb_rect = self.thumb_img.get_rect()
thumb_rect.centre = (self.val * self.rect.width,
                     self.rect.centery)
self.image.blit(self.thumb_img, thumb_rect)

class Spinner(pg.sprite.Sprite):
    def __init__(self, game, start_rect, options):
        self.game = game
        self.options = options
        self.index = 0

        # get font name
        font_name = self.game.config.text_font_name
        font_path = Path(self.game.config.img_path) / font_name

        # load font
        self.font = pg.font.Font(font_path.as_posix())
        self.text_colour = self.game.config.text_colour

        # load images and sounds
        self.arrows_img = self.game.img_loader.get("spinner arrows")
        self.end_hit_sound = self.game.snd_loader.get("spinner end.wav")

        self.rect = start_rect
        self.rescale()

    def update(self, dt, events):
        mouse_pos = pg.mouse.get_pos()
        for event in events:
            if event.type == pg.MOUSEBUTTONDOWN and event.button == 0:
                # if mouse is clicked over left button, decrease index
                if self.left_button_rect.collidepoint(mouse_pos):
                    if (self.index == 0):
                        self.end_hit_sound.play()
                    else:
                        self.index -= 1
                        self.rescale()
            # if mouse is clicked over right button, increase index
            elif self.right_button_rect.collidepoint(mouse_pos):
                if (self.index == len(self.options)-1):
                    self.end_hit_sound.play()
```

```
        else:
            self.index += 1
            self.rescale()

def rescale(self):
    # blit image
    self.image = pg.transform.scale(self.arrows_img, self.rect.size,
                                    self.image)

    text_img = self.font.render(self.options[self.index],
                                False, self.text_colour)
    # rescale text to fit screen
    scale_factor = self.rect.height / text_img.get_height()
    text_size = [text_img.get_width() * scale_factor, self.rect.height]
    text_img = pg.transform.scale(text_img, text_size)
    text_img_rect = text_img.get_rect()

    # button rects
    self.left_button_rect = pg.rect(self.rect.top,
                                    self.rect.left,
                                    self.rect.width * 1/12,
                                    self.rect.height)
    self.right_button_rect = pg.rect(self.rect.top,
                                    self.rect.left+self.rect.width * 11/12,
                                    self.rect.width * 1/12,
                                    self.rect.height)

    # blit text
    text_img_rect.center = self.rect.center
    self.image.blit(text_img, text_img_rect)

class Button(pg.sprite.Sprite):
    def __init__(self, game, start_rect, text):
        self.game = game
        self.text = text

        # get font name
        font_name = self.game.config.text_font_name
        font_path = Path(self.game.config.img_path) / font_name

        # load font
        self.font = pg.font.Font(font_path.as_posix())
        self.text_colour = self.game.config.text_colour
```

```
self.pressed = [False * 3]
self.rising_edges = [False * 3]
self.falling_edges = [False * 3]

self.rect = start_rect
self.rescale()

def update(self, dt, events):
    self.rising_edges = [False * 3]
    self.falling_edges = [False * 3]

    if self.rect.collidepoint(pg.mouse.get_pos()):
        for event in events:
            # detect rising edges
            if event.type == pg.MOUSEBUTTONDOWN:
                self.rising_edges[event.button] = True
                self.pressed[event.button] = True

            # detect falling edges
            if event.type == pg.MOUSEBUTTONUP:
                self.falling_edges[event.button] = True
                self.pressed[event.button] = False
        else:
            self.pressed = [False * 3]

def rescale(self):
    # rescale image to rect
    self.image = pg.surface.Surface(self.rect.size)
    self.image.fill((0,0,0))
    pg.draw.rect(self.image, (255,255,255), (0,0, self.rect.right-1,
                                                self.rect.bottom-1), 3)
    self.image = pg.transform.scale(self.image, self.rect.size)

    # render text
    text_img = self.font.render(self.text, False, self.text_colour)
    # rescale text to fit screen
    scale_factor = self.rect.height / text_img.get_height()
    text_size = [text_img.get_width() * scale_factor, self.rect.height]
    text_img = pg.transform.scale(text_img, text_size)
    text_img_rect = text_img.get_rect()

    # blit text
    text_img_rect.center = self.rect.center
```

```
    self.image.blit(text_img, text_img_rect)
```

STAGE 6: UNIT TESTING

The Test Host Program:

To test the menu sprites function correctly, the test host program uses them in a simple menu screen that behaves as the ones in the final game will, as to test all of the updating and rescaling functionality:

The test Host program:

- 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
- Allows the screen to be rescaled, and rescales the menu sprites

Test Host Program Code:

```
import pygame as pg
import config as cfg
import Asset_Loader as AL
import Menu_Sprites as MS

vec2 = pg.math.Vector2


class Game():
    def __init__(self):
        # init pygame
        pg.init()

        # config
        self.config = cfg.Config()

        # init screen
        self.screen = pg.display.set_mode(self.config.resolution,
                                         flags = pg.RESIZABLE)

        # init asset loaders
        self.img_loader = AL.Img_Loader(self)
        self.snd_loader = AL.Snd_Loader(self)

    # run level level
```

```
self.level = Level(self)

clock = pg.time.Clock()
while True:
    dt = clock.tick(75)
    events = pg.event.get()
    for event in events:
        if event.type == pg.QUIT:
            return
        if event.type == pg.VIDEORESIZE:
            self.level.rescale()

    self.level.tick(dt, events)

    pg.display.flip()

class Level():
    def __init__(self, game):
        self.game = game
        self.sprites = pg.sprite.Group()

        self.text = MS.Text(self.game, pg.rect.Rect(0,0,100,100), "text1")
        self.sprites.add(self.text)

        self.input_box = MS.Input_Box(self.game,
                                      pg.rect.Rect(0,0,300,100),
                                      "box 1",
                                      MS.k2c_numeric)
        self.sprites.add(self.input_box)

        self.toggle = MS.Toggle(self.game, pg.rect.Rect(0,0,50,50))
        self.sprites.add(self.toggle)

        self.slider = MS.Slider(self.game, pg.rect.Rect(0,0,300,50))
        self.sprites.add(self.slider)

        self.spinner = MS.Spinner(self.game, pg.rect.Rect(0,0,300,100),
                                  ["o1","o2","o3"])
        self.sprites.add(self.spinner)

        self.button = MS.Button(self.game, pg.rect.Rect(0,0,300,100), "b1")
        self.sprites.add(self.button)

    self.rescale()
```

```
def tick(self, dt, events):
    for sprite in self.sprites:
        sprite.update(dt, events)

    self.game.screen.fill((32,32,32))
    self.sprites.draw(self.game.screen)

def rescale(self):
    screen_size = vec2(pg.display.get_window_size())
    # reposition all the sprites

    self.text.rect.center = (screen_size.y * 1/4, screen_size.x * 1/6)
    self.input_box.rect.center = (screen_size.y * 1/4, screen_size.x * 3/6)
    self.toggle.rect.center = (screen_size.y * 1/4, screen_size.x * 5/6)

    self.slider.rect.center = (screen_size.y * 3/4, screen_size.x * 1/6)
    self.spinner.rect.center = (screen_size.y * 3/4, screen_size.x * 3/6)
    self.button.rect.center = (screen_size.y * 3/4, screen_size.x * 5/6)

    for sprite in self.sprites:
        sprite.rescale()

Game()
```

Issue Resolution:

Error 1:

```
D 54     self.font = pg.font.Font(font_path.as_posix())

Exception has occurred: TypeError
function takes exactly 2 arguments (1 given)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 54, in __init__
    self.font = pg.font.Font(font_path.as_posix())
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 47, in __init__
    self.text = MS.Text(self.game, pg.Rect(0,0,100,100), "text1")
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 26, in __init__
    self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module>
    Game()
```

Problem 1:

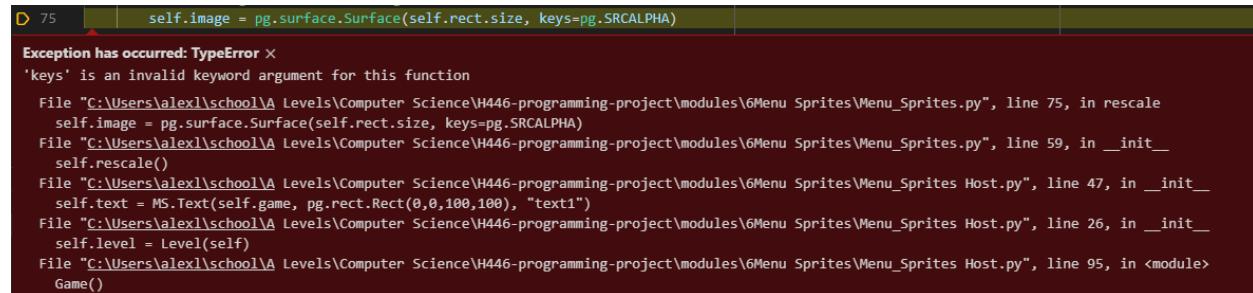
Font initialising needs a font size as well as font name:

```
self.font = pg.font.Font(font_path.as_posix())
```

Solution 1: supply font size as well

```
self.font = pg.font.Font(font_path.as_posix(), 20)
```

Error 2:



D 75 self.image = pg.surface.Surface(self.rect.size, keys=pg.SRCALPHA)

Exception has occurred: TypeError ×
'keys' is an invalid keyword argument for this function
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 75, in rescale
 self.image = pg.surface.Surface(self.rect.size, keys=pg.SRCALPHA)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 59, in __init__
 self.rescale()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 47, in __init__
 self.text = MS.Text(self.game, pg.Rect(0,0,100,100), "text1")
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 26, in __init__
 self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module>
 Game()

Problem 2:

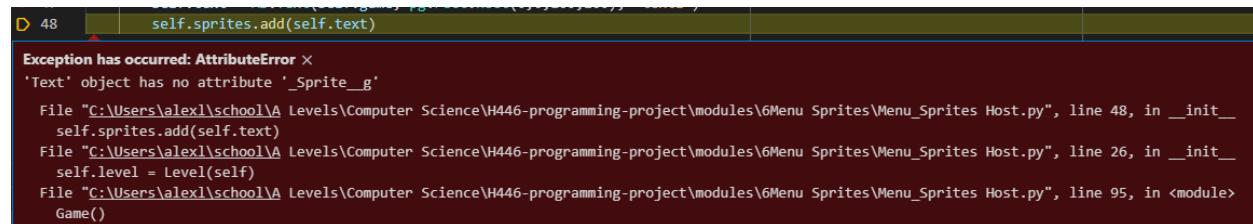
Surface constructor doesn't take keys keyword argument

```
self.image = pg.surface.Surface(self.rect.size, keys=pg.SRCALPHA)
```

Solution 2:

Surface constructor takes flags keyword argument

Error 3:



D 48 self.sprites.add(self.text)

Exception has occurred: AttributeError ×
'Text' object has no attribute '_Sprite_g'
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 48, in __init__
 self.sprites.add(self.text)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 26, in __init__
 self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module>
 Game()

Problem 3:

Parent constructor for all menu sprites hasn't been run

```
class Text(pg.sprite.Sprite):  
    def __init__(self, game, start_rect, text):  
        # store text to attribute  
        self.game = game  
        self.text = text
```

Solution 3:

Run parent constructor:

```
class Text(pg.sprite.Sprite):
    def __init__(self, game, start_rect, text):
        super().__init__()
        # store text to attribute
        self.game = game
```

Problem 4:

```
D 275 |     self.image = pg.transform.scale(self.arrows_img, self.rect.size)
Exception has occurred: AttributeError
'Spinner' object has no attribute 'image'
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 276, in rescale
    self.image)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 252, in __init__
    self.rescale()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 62, in __init__
    self.spinner = MS.Spinner(self.game, pg.Rect(0,0,300,100),
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 26, in __init__
    self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module>
    Game()
```

Error 4:

Too many arguments given to scale function

```
# blit image
self.image = pg.transform.scale(self.arrows_img, self.rect.size,
                                self.image)
```

Solution 4: give the correct number of arguments

```
self.image = pg.transform.scale(self.arrows_img, self.rect.size)
```

Error 5:

```
D 286 |     self.left_button_rect = pg.rect(self.rect.top,
Exception has occurred: TypeError
'module' object is not callable
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 286, in rescale
    self.left_button_rect = pg.rect(self.rect.top,
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 252, in __init__
    self.rescale()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 62, in __init__
    self.spinner = MS.Spinner(self.game, pg.Rect(0,0,300,100),
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 26, in __init__
    self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module>
    Game()
```

Problem 5:

Pg.rect is a module, not a class

```
self.left_button_rect = pg.rect(self.rect.top,
```

Solution 5:

Call the rect constructor correctly:

```
self.left_button_rect = pg.rect.Rect(self.rect.top,
```

Error 6:

```
D 347     self.image = pg.transform.scale(self.img, self.rect.size)

Exception has occurred: AttributeError
'Button' object has no attribute 'img'

File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 347, in rescale
    self.image = pg.transform.scale(self.img, self.rect.size)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 321, in __init__
    self.rescale()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 66, in __init__
    self.button = MS.Button(self.game, pg.Rect(0,0,300,100), "b1")
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 26, in __init__
    self.level = Level(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module>
    Game()
```

Problem 6:

residual code that loads an image left in:

```
def rescale(self):
    # rescale image to rect
    self.image = pg.surface.Surface(self.rect.size)
    self.image.fill((0,0,0))
    pg.draw.rect(self.image, (255,255,255), (0,0, self.rect.right-1,
                                                self.rect.bottom-1), 3)
    self.image = pg.transform.scale(self.img, self.rect.size)
```

Solution 6:

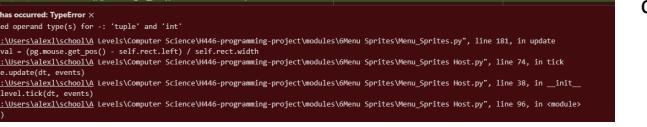
Remove residual code

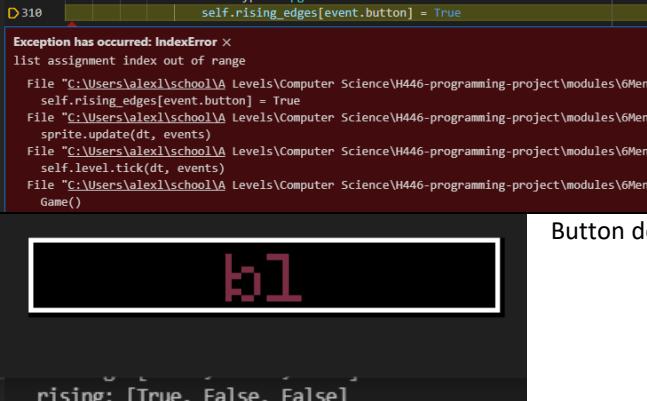
Now that the code gets to the point where it runs without instantly crashing, I can be tested for functionality using the tests set out in the design section:

Test Table:

No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior	Resultant behaviour	Pass / Fail + Solution
1	Text – initialization	A text element is instantiated with "text1" and rendered	Valid	Text 1 appears on screen, using the font from font .config	 text doesn't appear	<p>Fail: Problem: not correctly calculating the text's position on this sprite's image: <code>text_img_rect.center = self.rect.center</code></p> <p>Solution: correctly calculate text's position on the image: <code>text_img_rect.center = vec2(self.rect.size) / 2</code></p>
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	 Text is rendered on screen in the same position, scaled accordingly	Pass
3	Input_box - initialization	Input box is instantiated with no default text	Valid	Input box appears on screen	 Box renders on screen	Pass
4	Input_box – default text	Input box is instantiated with default text "input box 1"	Valid	Default text renders within input box	 default text renders in the box	Pass
5	Input_box - selection	cursor is hovered over input_box and left clicked	Valid	Input_box is highlighted, indicating it will accept text input	<p>Crashes</p> <pre>D 109 if event.type == pg.MOUSEBUTTONDOWN and event.key == 0: Exception has occurred: AttributeError 'Event' object has no attribute 'key' File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 109, in update if event.type == pg.MOUSEBUTTONDOWN and event.key == 0: File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 74, in tick sprite.update(dt, events) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 38, in __init__ self.level.tick(dt, events) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module> Game()</pre>	<p>Fail: Problem: when processing mouse events, it checks which button on the mouse has been pressed, but it is trying to check event.key: <code>if event.type == pg.MOUSEBUTTONDOWN and event.key == 0:</code></p> <p>Solution: Access event.button for the mouse buttons: <code>if event.type == pg.MOUSEBUTTONDOWN and event.button == 0:</code></p>
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	Nothing happens:	<p>Fail: Problem: an keydown events aren't in the expected format: expected variables like pg.K_left, but they provide Unicode characters instead:</p>

					<pre>> event: <Event(768-KeyDown { 'unicode': '\x08', 'key': 8, 'mod': 0, 'scancode': 42, 'type': 768, 'unicode': '\x08', 'window': None })</pre>	Solution: adjust key recognition code to account for the event format:	
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		No text appears	Pass
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		box is deselected and it doesn't take any more key strokes	Pass
9	Input_box – rescaling	Screen is rescaled	Valid	Input box rescales as to maintain its proportions on screen		rescales to fit the screen	Pass
10	Toggle – initialization	New toggle box is initialised	Valid	New toggle object is created and rendered to screen with no crashing		toggle image is missing	Fail: Problem: loader can't find toggle image as it isn't named correctly: <pre>self.ticked_img = self.game.img_loader.get("toggle_ticked") self.unticked_img = self.game.img_loader.get("toggle_unticked")</pre> Solution: load correct image name: <pre>self.ticked_img = self.game.img_loader.get("toggle ticked") self.unticked_img = self.game.img_loader.get("toggle unticked")</pre>
11	Toggle – rendering	New toggle box is initialised	Valid	Toggle box appears on screen unticked		toggle appears unticked	Pass
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		toggle changes state, even when cursor isn't over it	Fail: Problem: update code doesn't check cursor is over the toggle <pre>for event in events: if event.type == pg.MOUSEBUTTONDOWN and event.button == 1: self.ticked = not self.ticked self.rescale()</pre>

						Solution : add checking mouse position to update code: <pre>for event in events: if self.rect.collidepoint(pg.mouse.get_pos()): if event.type == pg.MOUSEBUTTONDOWN and event.button == 1: self.ticked = not self.ticked self.rescale()</pre>
13	Toggle – rescaling	Screen is rescaled	Valid	Toggle rescales such that it maintains the same proportions on screen	 toggle rescales	pass
14	Slider – initialization	New slider is initialized	Valid	New slider object appears on screen and causes no crashing		slider object appears
15	Slider - grabbing	cursor hovered over slider and left button is pressed	Valid	Slider thumb moves around so that it follows the mouse cursor		crashes
16	Slider – thumb stays on slider	Slider grabbed and cursor is moved around	Valid	If the cursor goes off the end of the slider, the thumb stops at the corresponding end until cursor returns		slider stops at the end and doesn't go any further
17	Slider – releasing	Left mouse button is released	Valid	Thumb stays in the same place it was in before left button was released		slider stays in the same location
18	Slider – rescaling	Screen is rescaled	Valid	Slider rescales such that its proportions on screen are maintained		slider rescales to a different size
19	Spinner - initialization	A new spinner object is initialized with options: o1, o2, o3	Valid	New spinner object is created and appears on screen		appears on screen
20	Spinner – right button	Cursor hovers over right button and right click is pressed	Valid	Spinner moves to next option		moves to next option
21	Spinner – left button	Cursor hovers over left button and left click is pressed	Valid	Spinner moves to previous option		spinner moves to previous option

22	Spinner - ends	Left and right buttons are pressed	Valid	Spinner moves to the final option and then doesn't go any further and doesn't wrap around		spinner stays at the end and doesn't wrap around	pass
23	Spinner – rescaling	Screen is rescaled	Valid	Spinner rescaled as to maintain the same proportions on screen		spinner rescales accordingly	pass
24	Button – initialization	A new button object is initialized with text b1	Valid	Button object appears on screen with correct text and font		button appears on screen	pass
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	 <pre>D:\10 self.rising_edges[event.button] = True Exception has occurred: IndexError list assignment index out of range File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites.py", line 310, in update self.rising_edges[event.button] = True File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 74, in tick sprite.update(dt, events) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 38, in __init__ self.level.tick(dt, events) File "C:\Users\alex\school\A Levels\Computer Science\H446-programming-project\modules\6Menu Sprites\Menu_Sprites Host.py", line 95, in <module> Game()</pre>	Fail: Problem: falling and rising edge arrays not initialized correctly: self.falling_edges = [False * 3] Solution: correctly initialize arrays: self.falling_edges = [False] * 3	pass
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		Button detects rising edges	Pass
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		button detects falling edges	pass
28	Button – clicking elsewhere on the screen	Cursor is moved elsewhere on screen and mouse buttons are pressed	Valid			nothing happens	pass
29	Button - rendering	Cursor hovers over button and mouse buttons are pressed	Valid	Button visually indicates that has been pressed		button indents to indicate it has been pushed	pass

30	Button – rescaling	Screen is rescaled	Valid	Button rescales to maintain same proportions on screen		Button rescales	pass
----	--------------------	--------------------	-------	--	---	-----------------	------

STAGE 6: STAKEHOLDER REVIEW

The user interface is very critical to the user experience as it is what the user interacts with to control and configure the game. As such, it must behave exactly as expected, with no strange bugs or quirky behaviour; this would make the game annoying to player, stopping it meeting its success criteria. To ensure that it behaves as the user expects, I have sent the UI test program to the stakeholders to gather their feedback about anything that could be changed to improve the overall experience.

Máté's has found the checkbox, slider, and spinner to work as he would expect, though has found a bug with the button where it remains stuck down if the mouse cursor moves away from the button before releasing; this makes the button harder to use. This has been fixed by checking for the mouse button being released even when the cursor isn't over the button. He also thinks that there should be more sound effects, for example the button pressed. This has been subsequently implemented.

Ben's feedback

As the code has passed all the tests set out in the design section, it is complete and fully functional. Now that the stakeholders have also given their input and the code has been changed to meet their requests, it fully meets its success criteria. This means that this module is complete for now and can be set aside until integration

STAGE 7 : MENU SYSTEM

The menu system provides an environment for all the rest of the game's processes to be displayed in; it has multiple screens that each display different menus or gameplay. They are responsible for handling menu navigation by changing the game state stack, allowing the user to navigate between menus. It is thus responsible for making the UI flow correctly by having navigation buttons clearly labelled and predictable; most of this has been set out in the UI section of the design, so the functionality only has to be correctly implemented now.

STAGE 7: DEVELOPMENT GOALS

- Main menu
 - Background image to represent the game
 - Title text
 - Can be used to open options, scoreboard or start a game
- Options
 - Can open either graphics or sound options
- Graphics options
 - Allow configuring of graphical options
 - Changes take effect after apply button is pressed
- Sound options
 - Allows configuring of sound options
 - Changes take effect immediately

- Scoreboard screen
 - Shows top 10 previous scores
- Pause screen
 - Provides access to options
 - Pauses game timer
- Start screen
 - Allows configuration of a new level
- End screen
 - Shows level statistics
 - Allows user to add themselves to the scoreboard

STAGE 7: DEVELOPMENT PROCESS

First writeup:

```
import pygame as pg
import Menu_Sprites as MS
import random as rng

vec2 = pg.math.Vector2
default_rect = lambda x : pg.rect.Rect(0,0,1,1)

class Ui_Screen():
    def __init__(self, game):
        self.game = game

        self.elements = pg.sprite.Group()
        self.background = False

        self.screen = self.game.screen

    def tick(self, event_list, dt):
        self.elements.update(event_list, dt)
        if self.background:
            self.game.screen.blit(self.bg_img, self.bg_rect)
        self.elements.draw(self.game.screen)

    def rescale(self):
        screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
        screen_size = vec2(screen_rect.size)

        # rescale the background if it is present
```

Initialises the variables required for a screen to function

Calls all element's update functions, then draws them on top of the background

```
if self.background:  
    # choose scale factor to fill screen  
    match_width_SF = screen_rect.width / self.background.get_width()  
    match_height_SF = screen_rect.height / self.background.get_height()  
    scale_factor = max(match_height_SF, match_width_SF)  
  
    # rescale image  
    self.bg_img = pg.transform.scale(self.background,  
                                    screen_size * scale_factor)  
    # position image  
    self.bg_rect = self.bg_img.get_rect()  
    self.bg_rect.center = screen_rect.center  
  
    # rescale all elements  
    for elements in self.elements:  
        elements.rescale()  
  
class Main(Ui_Screen):  
    def __init__(self, game):  
        super().__init__(game)  
  
        # init elements  
        self.title_text = MS.Text(game, default_rect(),  
                                "Colour Between The Lines")  
        self.start_b = MS.Button(game, default_rect(), "Start")  
        self.options_b = MS.Button(game, default_rect(), "Options")  
        self.scoreboard_b = MS.Button(game, default_rect(), "Scoreboard")  
        self.close_b = MS.Button(game, default_rect(), "Close")  
        # load background image  
        self.background = game.img_loader.get("main background")  
  
        # call rescale to render image  
        self.rescale()  
  
    def tick(self, event_list, dt):  
        # call parent tick function  
        super().tick(event_list, dt)  
        # push tick functions onto GSS for button presses  
  
        # start  
        if self.start_b.falling_edges[0]:  
            self.game.game_state_stack.append(self.game.start_screen.tick)  
        # options  
        elif self.options_b.falling_edges[0]:
```

To support all aspect ratios, the background image is scaled to fit the screen in larger of the two axes, and then it is cropped to fit the

A lot of the constructors look very similar; they all call the parent constructor, initialize the ui elements, load the background, then call rescale to render it all

The tick functions run the functionality of the screens, so are fairly unique, but they all follow a similar pattern: run parent tick method, which updates all elements, then check the buttons and enact that buttons functionality

```
        self.game.game_state_stack.append(self.game.options_screen.tick)
# scoreboard
elif self.scoreboard_b.falling_edges[0]:
    self.game.game_state_stack.append(self.game.scoreboard_screen.tick)
# close
elif self.options_b.falling_edges[0]:
    self.game.game_state_stack = []

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.title_text.rect.width = screen_size.x / 2
    self.title_text.rect.height = screen_size.y / 8
    self.title_text.rect.centerx = screen_rect.centerx
    self.title_text.rect.centery = screen_size.y / 4

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 10
    button_spacing = screen_size.y / 8

    button_pos_y = screen_size.y * 2 / 6

    # start button
    self.start_b.rect.width = button_width
    self.start_b.rect.height = button_height
    self.start_b.rect.centerx = screen_rect.centerx
    self.start_b.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # options button
    self.options_b.rect.width = button_width
    self.options_b.rect.height = button_height
    self.options_b.rect.centerx = screen_rect.centerx
    self.options_b.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # scoreboard button
    self.scoreboard_b.rect.width = button_width
    self.scoreboard_b.rect.height = button_height
    self.scoreboard_b.rect.centerx = screen_rect.centerx
    self.scoreboard_b.rect.centery = button_pos_y
```

The rescale methods all do almost exactly the same thing; they assign the rects of all the sprites to arrange them on screen. This means calculating on screen proportions as fractions of the screen's dimensions

```
button_pos_y += button_spacing

# close button
self.close_b.rect.width = button_width
self.close_b.rect.height = button_height
self.close_b.rect.centerx = screen_rect.centerx
self.close_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# call parent rescale method
super().rescale()

class Pause(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.pause_text = MS.Text(game, default_rect(), "Pause")
        self.resume_b = MS.Button(game, default_rect(), "Resume")
        self.options_b = MS.Button(game, default_rect(), "Options")
        self.exit_b = MS.Button(game, default_rect(), "Exit")

        # load background
        self.background = game.img_loader.get("menu background")

        # call rescale to render image
        self.rescale()

    def tick(self, events, dt):
        # call parent tick method
        super().tick(events, dt)

        # resume button and ESC key
        if self.resume_b.falling_edges[0] or \
           max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
            self.game.game_state_stack.pop(-1)

        # options button
        elif self.options_b.falling_edges[0]:
            self.game.game_state_stack.append(self.game.options_screen.tick)

        # exit button
        elif self.exit_b.falling_edges[0]:
```

```
        self.game.game_state_stack = [self.game.main_screen.tick]

def rescale(self):
    screen_rect = pg.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # position title text
    self.pause_text.rect.width = screen_size.x / 2
    self.pause_text.rect.height = screen_size.y / 8
    self.pause_text.rect.centerx = screen_rect.centerx
    self.pause_text.rect.centery = screen_size.y / 4

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 10
    button_spacing = screen_size.y / 8
    button_pos_y = screen_size.y * 1 / 6

    # resume button
    self.resume_b.rect.width = button_width
    self.resume_b.rect.height = button_height
    self.resume_b.rect.centerx = screen_rect.centerx
    self.resume_b.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # options button
    self.options_b.rect.width = button_width
    self.options_b.rect.height = button_height
    self.options_b.rect.centerx = screen_rect.centerx
    self.options_b.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # exit button
    self.exit_b.rect.width = button_width
    self.exit_b.rect.height = button_height
    self.exit_b.rect.centerx = screen_rect.centerx
    self.exit_b.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # call parent rescale method
    super().rescale()

class Options(Ui_Screen):
    def __init__(self, game):
```

```
super().__init__(game)

# init elements
self.options_text = MS.Text(game, default_rect(), "Options")
self.gfx_b = MS.Button(game, default_rect(), "Graphics")
self.snd_b = MS.Button(game, default_rect(), "Sound")
self.exit_b = MS.Button(game, default_rect(), "Exit")

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick function
    super().tick(events, dt)

    # gfx button
    if self.gfx_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.gfx_options_screen.tick)

    # snd button
    if self.snd_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.snd_options_screen.tick)

    # exit button and ESC key
    if self.exit_b.falling_edges[0] or \
       max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
        self.game.game_state_stack.pop(-1)

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.options_text.rect.width = screen_size.x / 2
    self.options_text.rect.height = screen_size.y / 8
    self.options_text.rect.centerx = screen_rect.centerx
    self.options_text.rect.centery = screen_size.y / 4

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 10
```

```
button_spacing = screen_size.y / 8
button_pos_y = screen_size.y * 1 / 6

# gfx button
self.gfx_b.rect.width = button_width
self.gfx_b.rect.height = button_height
self.gfx_b.rect.centerx = screen_rect.centerx
self.gfx_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# snd button
self.snd_b.rect.width = button_width
self.snd_b.rect.height = button_height
self.snd_b.rect.centerx = screen_rect.centerx
self.snd_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# exit button
self.exit_b.rect.width = button_width
self.exit_b.rect.height = button_height
self.exit_b.rect.centerx = screen_rect.centerx
self.exit_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# call parent rescale method
super().rescale()
```

```
class GFX_Options(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.gfx_text = MS.Text(game, default_rect(), "Graphics")
        self.res_sp = MS.Spinner(game, default_rect(), ["640x480",
                                                       "800x600",
                                                       "1280x720",
                                                       "1366x768",
                                                       "1600x900",
                                                       "1920x1080",
                                                       "Rescalable"])

        if game.config.rescaleable:
            res_n = "Rescalable"
        else:
```

The graphics options has a list of allowed resolutions stored in the spinner's options

```
    res_n = f"{game.config.resolution[0]}x{game.config.resolution[1]}"
    self.res_sp.index = self.res_sp.options.index(res_n)

    self.fullscreen_text = MS.Text(game, default_rect(), "Fullscreen")
    self.Vsync_text = MS.Text(game, default_rect(), "Vsync")

    self.fullscreen_tg = MS.Toggle(game, default_rect())
    self.fullscreen_tg.ticked = game.config.fullscreen

    self.Vsync_tg = MS.Toggle(game, default_rect())
    self.Vsync_tg.ticked = game.config.vsync

    self.apply_b = MS.Button(game, default_rect(), "Apply")
    self.exit_b = MS.Button(game, default_rect(), "Exit")

    # load background
    self.background = game.img_loader.get("menu background")

    # call rescale to render image
    self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # change settings if apply pressed
    if self.apply_b.falling_edges[0]:
        # resolution
        res_option = self.res_sp.options[self.res_sp.index]
        if res_option == "Rescalable":
            self.game.config.rescaleable = True
        else:
            self.game.config.rescaleable = False
            self.game.config.resolution = [int(i) for i in
                                           res_option.split("x")]

        # fullscreen
        self.game.config.fullscreen = self.fullscreen_tg.ticked

        # vsync
        self.game.config.vsync = self.Vsync_tg.ticked

        # save changes
        self.game.config.save()
```

When the apply button is pressed, the data is taken from the ui elements and transferred to config

```
# exit button and ESC key
if self.exit_b.falling_edges[0] or \
    max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
    self.game.game_state_stack.pop(-1)

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.gfx_text.rect.width = screen_size.x / 2
    self.gfx_text.rect.height = screen_size.y / 8
    self.gfx_text.rect.centerx = screen_rect.centerx
    self.gfx_text.rect.centery = screen_size.y / 4

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 10
    button_spacing = screen_size.y / 8
    button_pos_y = screen_size.y * 1 / 6

    # res spinner
    self.res_sp.rect.width = button_width
    self.res_sp.rect.height = button_height
    self.res_sp.rect.centerx = screen_rect.centerx
    self.res_sp.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # fullscreen text
    self.fullscreen_text.rect.width = button_width * 2/3
    self.fullscreen_text.rect.height = button_height
    self.fullscreen_text.rect.centerx = screen_rect.centerx-button_width*1/3
    self.fullscreen_text.rect.centery = button_pos_y

    # fullscreen toggle
    self.fullscreen_tg.rect.width = button_width * 1/6
    self.fullscreen_tg.rect.height = button_width * 1/6
    self.fullscreen_tg.rect.centerx = screen_rect.centerx+button_width*1/2
    self.fullscreen_tg.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # Vsync text
    self.Vsync_text.rect.width = button_width * 2/3
```

```
        self.Vsync_text.rect.height = button_height
        self.Vsync_text.rect.centerx = screen_rect.centerx-button_width*1/3
        self.Vsync_text.rect.centery = button_pos_y

        # Vsync toggle
        self.Vsync_tg.rect.width = button_width * 1/6
        self.Vsync_tg.rect.height = button_width * 1/6
        self.Vsync_tg.rect.centerx = screen_rect.centerx+button_width*1/2
        self.Vsync_tg.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # apply button
        self.apply_b.rect.width = button_width
        self.apply_b.rect.height = button_height
        self.apply_b.rect.centerx = screen_rect.centerx
        self.apply_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = button_width
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # call parent rescale method
        super().rescale()

class SND_Options(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.snd_text = MS.Text(game, default_rect(), "Sound")
        self.game_text = MS.Text(game, default_rect(), "Game Volume:")
        self.music_text = MS.Text(game, default_rect(), "Music Volume:")
        self.game_slider = MS.Slider(game, default_rect())
        self.game_slider.val = game.config.game_vol
        self.music_slider = MS.Slider(game, default_rect())
        self.music_slider.val = game.config.music_vol
        self.exit_b = MS.Button(game, default_rect(), "Exit")

        # load background
        self.background = game.img_loader.get("menu background")
```

```
# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # store sound vols to config
    self.game.config.game_vol = self.game_slider.val
    self.game.config.music_vol = self.music_slider.val

    # exit button and ESC key
    if self.exit_b.falling_edges[0] or \
        max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
        self.game.game_state_stack.pop(-1)

    # call game's load sound function
    self.game.load_snd_vol()

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.snd_text.rect.width = screen_size.x / 2
    self.snd_text.rect.height = screen_size.y / 8
    self.snd_text.rect.centerx = screen_rect.centerx
    self.snd_text.rect.centery = screen_size.y / 4

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 10
    button_spacing = screen_size.y / 8
    button_pos_y = screen_size.y * 1 / 6

    # game volume text
    self.game_text.rect.width = button_width
    self.game_text.rect.height = button_height / 2
    self.game_text.rect.centerx = screen_rect.centerx
    self.game_text.rect.centery = button_pos_y

    # game volume slider
    self.game_slider.rect.width = button_width
```

```
        self.game_slider.rect.height = button_height / 2
        self.game_slider.rect.centerx = screen_rect.centerx
        self.game_slider.rect.centery = button_pos_y + screen_size.y / 12
        button_pos_y += button_spacing

        # music volume text
        self.music_text.rect.width = button_width
        self.music_text.rect.height = button_height / 2
        self.music_text.rect.centerx = screen_rect.centerx
        self.music_text.rect.centery = button_pos_y

        # music volume slider
        self.music_slider.rect.width = button_width
        self.music_slider.rect.height = button_height / 2
        self.music_slider.rect.centerx = screen_rect.centerx
        self.music_slider.rect.centery = button_pos_y + screen_size.y / 12
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = button_width
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

    # call parent rescale method
    super().rescale()

class Scoreboard(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.sb_text = MS.Text(game, default_rect(), "Scoreboard")
        self.exit_b = MS.Button(game, default_rect(), "Exit")

        # score boxes
        self.score_boxes = []
        for _ in range(11):
            box = MS.Text(game, default_rect(), "")
            self.score_boxes.append(box)
            self.elements.add(box)
```

The scoreboard renders it's table using text, so there is a text element for each row

```

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.load()
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().__init__(events, dt)

    # exit button and ESC key
    if self.exit_b.falling_edges[0] or \
       max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
        self.game.game_state_stack.pop(-1)

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.sb_text.rect.width = screen_size.x / 2
    self.sb_text.rect.height = screen_size.y / 8
    self.sb_text.rect.centerx = screen_rect.centerx
    self.sb_text.rect.centery = screen_size.y / 4

    # render top 10 scores
    table_width = screen_size.x * 3/4
    table_row_height = screen_size.y * 1/3
    table_entry_height = screen_size.y * 1/20

    # table heading
    box = self.score_boxes[0]
    box.rect.width = table_width
    box.rect.height = table_entry_height
    box.rect.centerx = screen_rect.centerx
    box.rect.centery = table_row_height
    table_row_height += table_entry_height
    box.text = \
        f"# {'Name':^20}|{'time':^6}|{'width':^8}|{'height':^8}|{'seed':^8}"

    # sort by shortest time
    sorted_scores = sorted(self.scoreboard_data, key = lambda x : x[1])

```

each row in the table is generated in a for loop, accessing the data from a sorted list. To format the data correctly so it lines up, f strings are used.

```
# set data for each box
for i in range(1,11):
    # set row position
    box = self.score_boxes[i]
    box.rect.width = table_width
    box.rect.height = table_entry_height
    box.rect.centerx = screen_rect.centerx
    box.rect.centery = table_row_height
    table_row_height += table_entry_height

    # set row text
    r = sorted_scores[i-1]
    box.text = \
        f"{{i:^2}}|{{r[0]:^20}}|{{r[1]:^6}}|{{r[2]:^8}}|{{r[3]:^8}}|{{r[4]:^8}}"

    # exit button
    self.exit_b.rect.width = screen_size.x / 3
    self.exit_b.rect.height = screen_size.y / 10
    self.exit_b.rect.centerx = screen_rect.centerx
    self.exit_b.rect.centery = screen_size.y * 5/6

    # call parent rescale method
    super().rescale()

def load(self):
    # load scoreboard file
    scores_file = open(self.game.config.scoreboard_path, "r")

    # split by lines
    self.scoreboard_data = []
    for line in scores_file[:-1]:
        self.scoreboard_data.append(line.split(","))

    # close scoreboard file
    scores_file.close()

def save(self):
    # convert data to a string
    output_str = ""
    for row in self.scoreboard_data:
        for column in row:
            output_str += f"{column},"
    output_str += "\n"
```

As the data is stored in a CSV, it is simple to import it and export it using some iteration

```
# load scoreboard file
scores_file = open(self.game.config.scoreboard_path, "w")

# write to file
scores_file.write(output_str)

# close file
scores_file.close()

def add_score(self, name, time, width, height, seed):
    # load data before appending to it
    self.load()

    # add new data
    self.scoreboard_data.append([name, time, width, height, seed])

    # save new data and apply changes to screen
    self.save()
    self.rescale()

class Start(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.start_text = MS.Text(game, default_rect(), "Start Level")
        self.label_text = MS.Text(game, default_rect(), "Maze Size:")
        self.width_text = MS.Text(game, default_rect(), "Width:")
        self.height_text = MS.Text(game, default_rect(), "Height:")
        self.cross_text = MS.Text(game, default_rect(), "X")
        self.seed_text = MS.Text(game, default_rect(), "Seed:")
        self.width_i = MS.Input_Box(game, default_rect(), "20", MS.k2c_numeric)
        self.height_i = MS.Input_Box(game, default_rect(), "10", MS.k2c_numeric)
        self.seed_i = MS.Input_Box(game, default_rect(), "0", MS.k2c_numeric)
        self.start_b = MS.Button(game, default_rect(), "Start Level")
        self.exit_b = MS.Button(game, default_rect(), "Cancel")

        # load background
        self.background = game.img_loader.get("menu background")

        # call rescale to render image
        self.rescale()
```

When scores are added, the file is loaded first to make sure there are no unforeseen changes to the file that are overwritten

```
def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # start level button
    if self.start_b.falling_edges[0]:
        # get width
        if len(self.width_i.text) > 0:
            width = int(self.width_i.text)
        else:
            width = int(self.width_i.default_text)

        # get height
        if len(self.height_i.text) > 0:
            height = int(self.height_i.text)
        else:
            height = int(self.height_i.default_text)

        # get seed
        if len(self.seed_i.text) > 0:
            seed = int(self.seed_i.text)
        else:
            seed = int(self.seed_i.default_text)

        # start level
        self.game.start_level((width, height), seed)

    # exit button and ESC key
    if self.exit_b.falling_edges[0] or \
       max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
        self.game.game_state_stack.pop(-1)

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.start_text.rect.width = screen_size.x / 2
    self.start_text.rect.height = screen_size.y / 8
    self.start_text.rect.centerx = screen_rect.centerx
    self.start_text.rect.centery = screen_size.y / 4

    # position buttons
    button_width = screen_size.x / 3
```

```
button_height = screen_size.y / 10
button_spacing = screen_size.y / 8
button_pos_y = screen_size.y * 1 / 6

# maze size text
self.label_text.rect.width = button_width
self.label_text.rect.height = button_height
self.label_text.rect.centerx = screen_rect.centerx
self.label_text.rect.centery = button_pos_y
button_pos_y += button_spacing

# width and height text
width_x = screen_rect.centerx - screen_size.x / 6
height_x = screen_rect.centerx + screen_size.x / 6
dim_width = button_width * 3/8

self.width_text.rect.width = dim_width
self.width_text.rect.height = button_height
self.width_text.rect.centerx = width_x
self.width_text.rect.centery = button_pos_y

self.height_text.rect.width = dim_width
self.height_text.rect.height = button_height
self.height_text.rect.centerx = height_x
self.height_text.rect.centery = button_pos_y
button_pos_y += button_spacing

# width and height inputs
self.width_i.rect.width = dim_width
self.width_i.rect.height = button_height
self.width_i.rect.centerx = width_x
self.width_i.rect.centery = button_pos_y

self.height_i.rect.width = dim_width
self.height_i.rect.height = button_height
self.height_i.rect.centerx = height_x
self.height_i.rect.centery = button_pos_y

# X
self.cross_text.rect.width = screen_size.x / 20
self.cross_text.rect.height = button_height
self.cross_text.rect.centerx = screen_rect.centerx
self.cross_text.rect.centery = button_pos_y
button_pos_y += button_spacing
```

```
# seed input
self.seed_text.rect.width = button_width * 1/4
self.seed_text.rect.height = button_height
self.seed_text.rect.centerx = screen_rect.centerx - button_width * 3/8
self.seed_text.rect.centery = button_pos_y

self.seed_i.rect.width = button_width * 3/4
self.seed_i.rect.height = button_height
self.seed_i.rect.centerx = screen_rect.centerx + button_width * 1/8
self.seed_i.rect.centery = button_pos_y
button_pos_y += button_spacing

# randomise seed
rng.seed()
self.seed_i.default_text = rng.randint(0,999999)

# start button
self.start_b.rect.width = button_width
self.start_b.rect.height = button_height
self.start_b.rect.centerx = screen_rect.centerx
self.start_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# exit button
self.exit_b.rect.width = button_width
self.exit_b.rect.height = button_height
self.exit_b.rect.centerx = screen_rect.centery
self.exit_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# call parent rescale method
super().rescale()

class End(Ui_Screen):
    def __init__(self, game, time, width, height, seed):
        super().__init__(game)
        self.score_added = False
        self.time = time
        self.width = width
        self.height = height
        self.seed = seed
```

```
# init elements
self.end_text = MS.Text(game, default_rect(), "Level Complete!")
self.dim_text = MS.Text(game, default_rect(), f"{{width:^8}}X{{height:^8}}")
self.seed_text = MS.Text(game, default_rect(), f"Seed:{seed:^8}")
self.Time_text = MS.Text(game, default_rect(), f"Time:{time:^8}")

self.sb_text = MS.Text(game, default_rect(),
                      "Put Your Name on The Scoreboard")
self.name_i = MS.Input_Box(game, default_rect(), "Enter Name",
                           MS.k2c_numeric +
                           MS.k2c_alpha_lower +
                           MS.k2c_alpha_upper)
self.add_b = MS.Button(game, default_rect(), "Add To Scoreboard")

self.exit_b = MS.Button(game, default_rect(), "Return to Main Menu")

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # add to scoreboard
    if not self.score_added and self.add_b.falling_edges[0]:
        if len(self.name_i.text) > 0:
            self.score_added = True
            self.game.scoreboard_screen.add_score(self.name_i.text,
                                                   self.time,
                                                   self.width,
                                                   self.height,
                                                   self.seed)
            self.add_b.text_colour = (128,128,128)

    # exit button and ESC key
    if self.exit_b.falling_edges[0] or \
       max([e.type == pg.KEYUP and e.key == pg.K_ESCAPE for e in events]):
        self.game.game_state_stack.pop(-1)

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
```

```
screen_size = vec2(screen_rect.size)

# positon title text
self.end_text.rect.width = screen_size.x / 2
self.end_text.rect.height = screen_size.y / 8
self.end_text.rect.centerx = screen_rect.centerx
self.end_text.rect.centery = screen_size.y / 4

# position buttons
button_width = screen_size.x / 3
button_height = screen_size.y / 10
button_spacing = screen_size.y / 8
button_pos_y = screen_size.y * 1 / 6

left_column = screen_size.x * 1/4
right_column = screen_size.x * 3/4

# maze size text
self.dim_text.rect.width = button_width
self.dim_text.rect.height = button_height
self.dim_text.rect.centerx = left_column
self.dim_text.rect.centery = button_pos_y

# add to scoreboard text
self.sb_text.rect.width = button_width
self.sb_text.rect.height = button_height
self.sb_text.rect.centerx = right_column
self.sb_text.rect.centery = button_pos_y
button_pos_y += button_spacing

# seed text
self.seed_text.rect.width = button_width
self.seed_text.rect.height = button_height
self.seed_text.rect.centerx = left_column
self.seed_text.rect.centery = button_pos_y

# name input
self.name_i.rect.width = button_width
self.name_i.rect.height = button_height
self.name_i.rect.centerx = right_column
self.name_i.rect.centery = button_pos_y
button_pos_y += button_spacing

# time text
```

```
        self.Time_text.rect.width = button_width
        self.Time_text.rect.height = button_height
        self.Time_text.rect.centerx = left_column
        self.Time_text.rect.centery = button_pos_y

        # add name button
        self.add_b.rect.width = button_width
        self.add_b.rect.height = button_height
        self.add_b.rect.centerx = right_column
        self.add_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = button_width
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y

        # call parent rescale method
        super().rescale()

class Level(Ui_Screen):
    def __init__(self, game, size, seed):
        super().__init__(game)
        print(f"level, size:{size}, seed:{seed} created")

    def tick(self, events, dt):
        super().tick(events, dt)

        for event in events:
            if event.type == pg.pg.K_DOWN and \
               event.key == pg.K_SPACE:
                self.game.end_screen = End(self.game, "987", "20", "10", "1234")
                self.game.game_state_stack.append(self.game.end_screen.tick)
```

The level screen is just a simple dummy that allows progressing onto the end screen; the final one will be done in integration

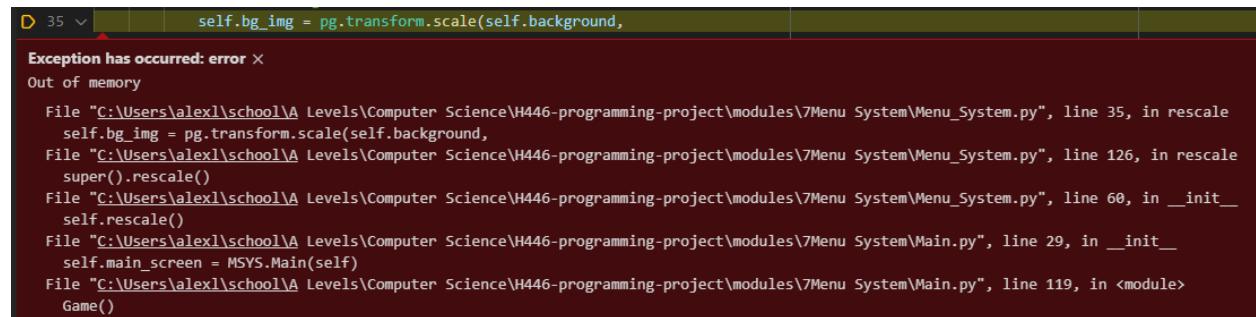
STAGE 7: UNIT TESTING

Test Host environment:

As this module is designed to couple with the main program developed in stage3, that has been used to run the program and provide the functionality needed for testing

Issue resolution:

Error 1:



D 35 ▾ self.bg_img = pg.transform.scale(self.background,
Exception has occurred: error X
Out of memory
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 35, in rescale
self.bg_img = pg.transform.scale(self.background,
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 126, in rescale
super().rescale()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 60, in __init__
self.rescale()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System>Main.py", line 29, in __init__
self.main_screen = MSYS.Main(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System>Main.py", line 119, in <module>
Game()

Problem 1:

In determining the resolution for the background image, it is multiplying the scale factor by the screen size, not the background image size, resulting in attempting to create 256000 by 144000 image, which causes an out of memory exception.

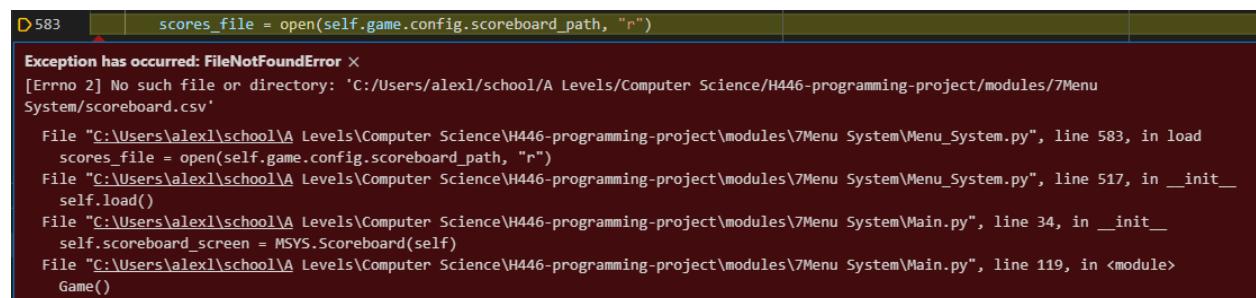
```
# rescale image
self.bg_img = pg.transform.scale(self.background,
                                  screen_size * scale_factor)
# position image
```

Solution 1:

Correctly calculate the backgrounds size from the unscaled size, not the screen size

```
# rescale image
background_size = vec2(self.background.get_size()) * scale_factor
self.bg_img = pg.transform.scale(self.background, background_size)
```

Error 2:



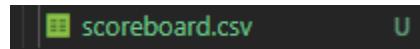
D 583 ▾ scores_file = open(self.game.config.scoreboard_path, "r")
Exception has occurred: FileNotFoundError X
[Errno 2] No such file or directory: 'C:/Users/alex1/school/A Levels/Computer Science/H446-programming-project/modules/7Menu System/scoreboard.csv'
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 583, in load
scores_file = open(self.game.config.scoreboard_path, "r")
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 517, in __init__
self.load()
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System>Main.py", line 34, in __init__
self.scoreboard_screen = MSYS.Scoreboard(self)
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System>Main.py", line 119, in <module>
Game()

Problem 2:

There is no scoreboard file

Solution 2:

Create a scoreboard file



Error 3:

```
D 587 for line in scores_file[:-1]:  
  
Exception has occurred: TypeError ×  
'_io.TextIOWrapper' object is not subscriptable  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 587, in  
load  
    for line in scores_file[:-1]:  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 517, in  
__init__  
    self.load()  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Main.py", line 34, in __init__  
    self.scoreboard_screen = MSYS.Scoreboard(self)  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Main.py", line 119, in <module>  

```

Problem 3:

Not correctly accessing the lines of a file:

```
for line in scores_file[:-1]:
```

Solution 3:

Call the correct function to access the lines of a file:

```
for line in scores_file.readlines()[:-1]:
```

Error 4:

```
D 568 r = sorted_scores[i-1]  
  
Exception has occurred: IndexError ×  
list index out of range  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 568, in  
rescale  
    r = sorted_scores[i-1]  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Menu_System.py", line 518, in  
__init__  
    self.rescale()  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Main.py", line 34, in __init__  
    self.scoreboard_screen = MSYS.Scoreboard(self)  
File "C:\Users\alex1\school\A Levels\Computer Science\H446-programming-project\modules\7Menu System\Main.py", line 119, in <module>  

```

Problem 4:

It attempts to render all 10 lines of the table even when there aren't 10 entries

```
r = sorted_scores[i-1]  
box.text = \  
f"{{i:^2}}|{{r[0]:^20}}|{{r[1]:^6}}|{{r[2]:^8}}|{{r[3]:^8}}|{{r[4]:^8}}"
```

Solution 4:

Check if there is a row to render before trying to render it:

```
# set row text
if len(sorted_scores) > i-1:
    r = sorted_scores[i-1]
    box.text = \
f"{i:^2}|{r[0]:^20}|{r[1]:^6}|{r[2]:^8}|{r[3]:^8}|{r[4]:^8}"
```

Error 5:

Nothing appears on the screen

Problem 5:

No of the ui elements have been appended to the element groups, thus they aren't rendered

```
# init elements
self.title_text = MS.Text(game, default_rect(),
                           "Colour Between The Lines")
self.start_b = MS.Button(game, default_rect(), "Start")
self.options_b = MS.Button(game, default_rect(), "Options")
self.scoreboard_b = MS.Button(game, default_rect(), "Scoreboard")
self.close_b = MS.Button(game, default_rect(), "Close")
# load background image
self.background = game.img_loader.get("main background")
```

Solution 5:

Add all ui elements to the elements groups

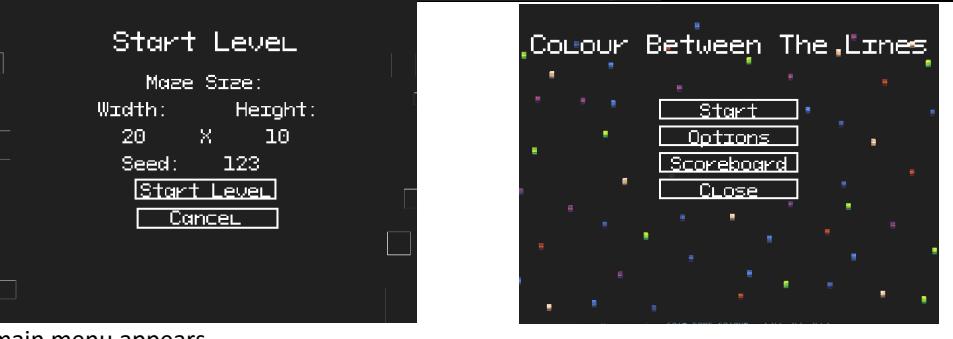
Now that the code works without crashing instantly, It can be tested on the tests set out in the design stage

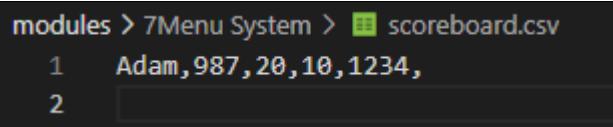
Candidate Name: Alexander Mills Candidate Number: 9120

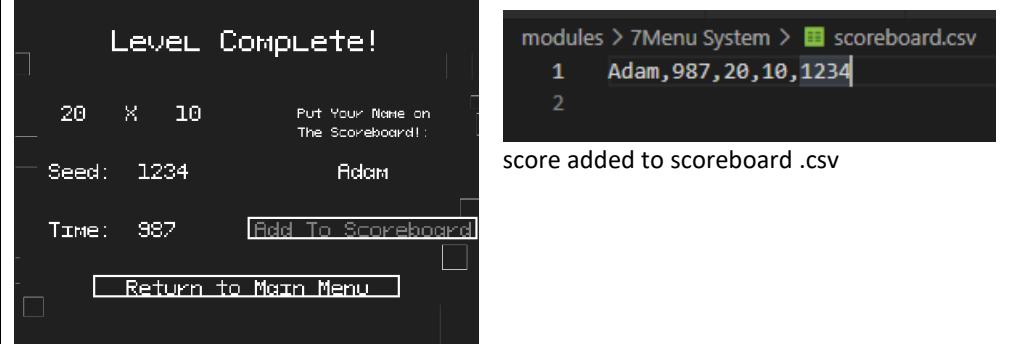
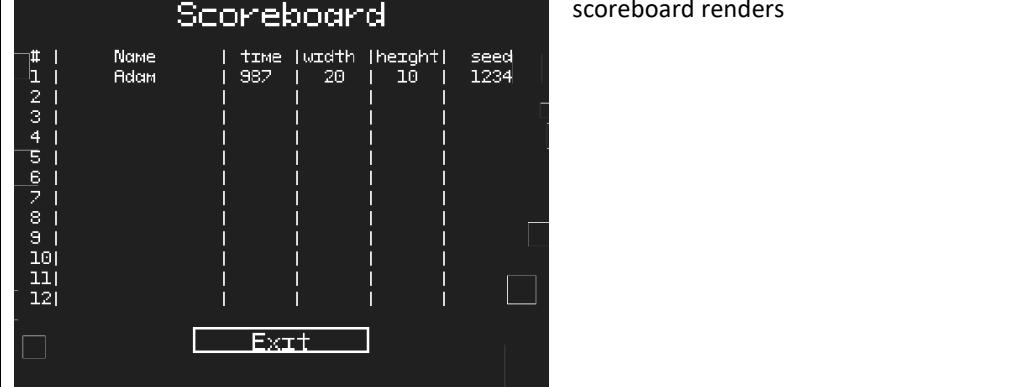
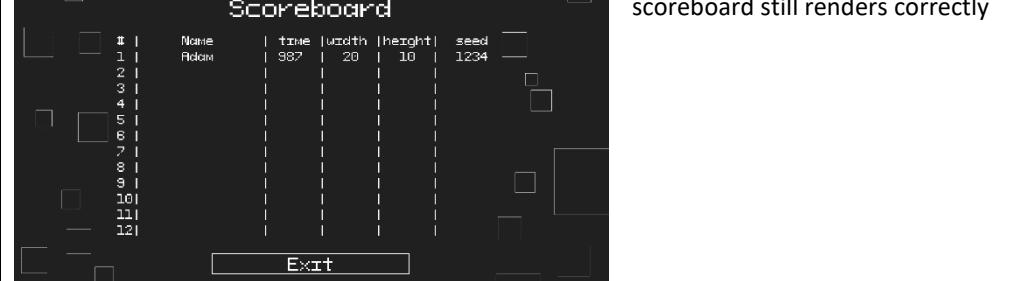
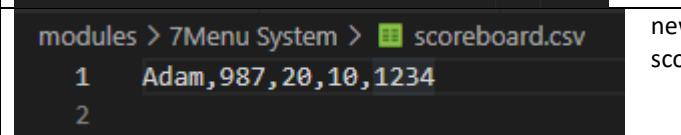
Test Table:

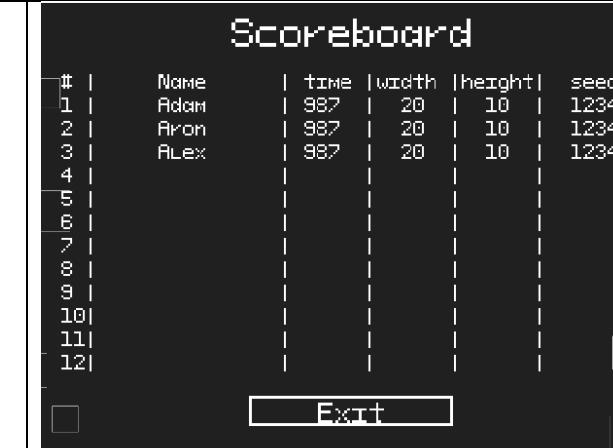
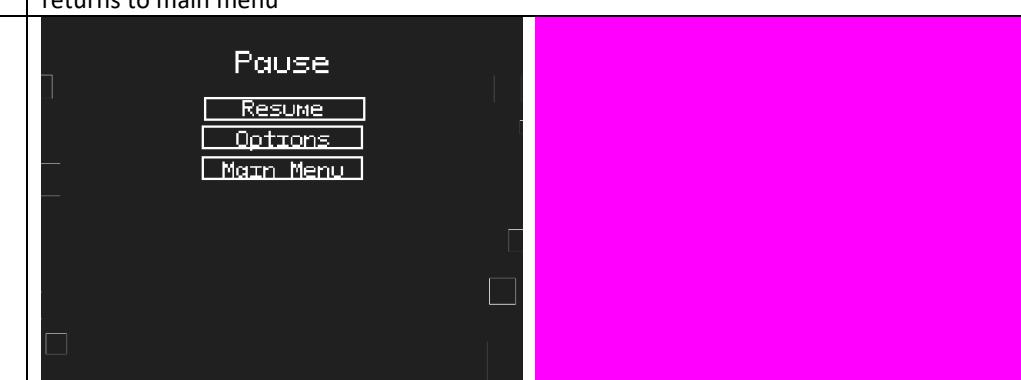
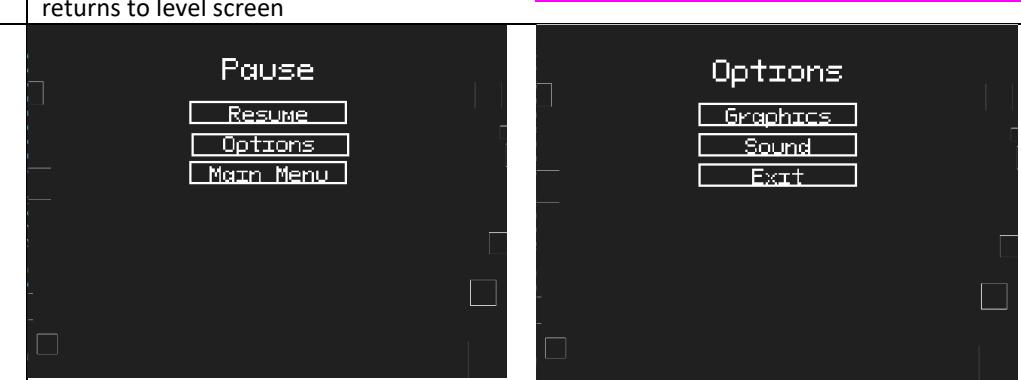
No.	Tested Functionality	Test conditions/ input	Test type	Expected behavior	Resultant behavior	Pass / Fail + solution
1	Main menu - start button	Start button pressed	Normal	Start screen appears	  start button opens start menu	Pass
2	Main menu – options button	Options button pressed	Normal	Options screen appears	  options menu opens	Pass
3	Main menu – scoreboard button	Scoreboard button pressed	Normal	Scoreboard screen appears	  Scoreboard opens	Pass
4	Main menu – close button	Close button pressed	Normal	Game exits	Game exits	Pass

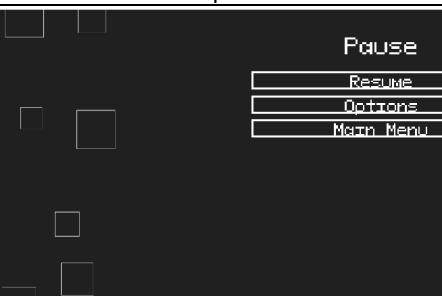
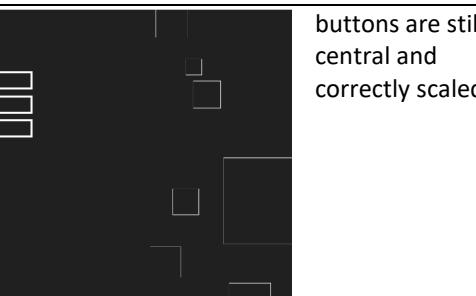
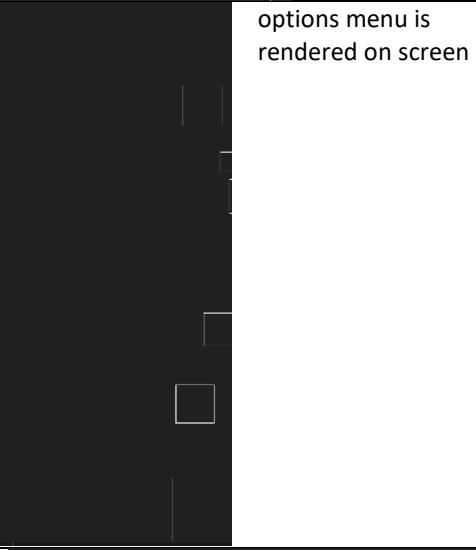
5	Start screen – width box	Input number into width box: 123	Normal	Number can be entered		numbers can be entered	pass
6	Start screen – width box	Input characters into width box: abc	invalid	Characters can't be entered		letters can't be entered	Pass
7	Start screen – height box	Input number into height box: 123	Normal	Number can be entered		numbers can be entered	Pass
8	Start screen – height box	Input characters into height box: abc	invalid	Characters can't be entered		letters can't be entered	

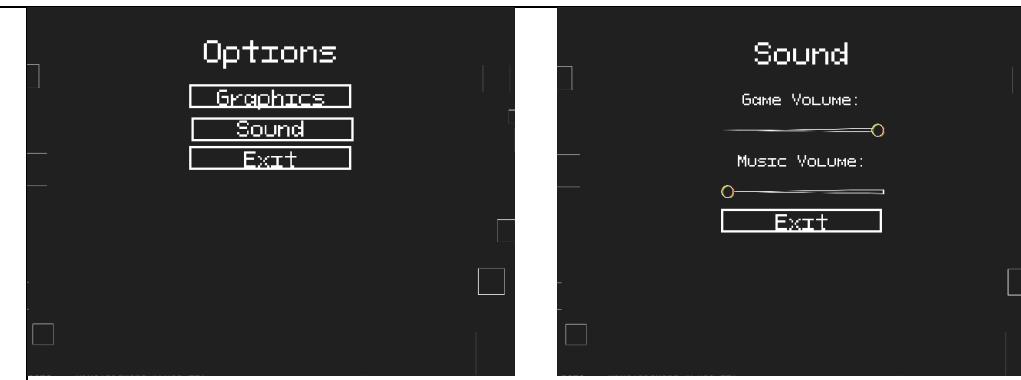
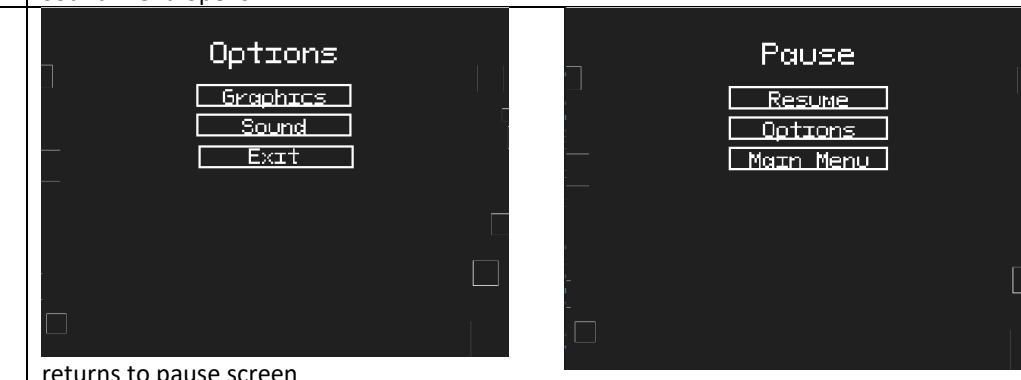
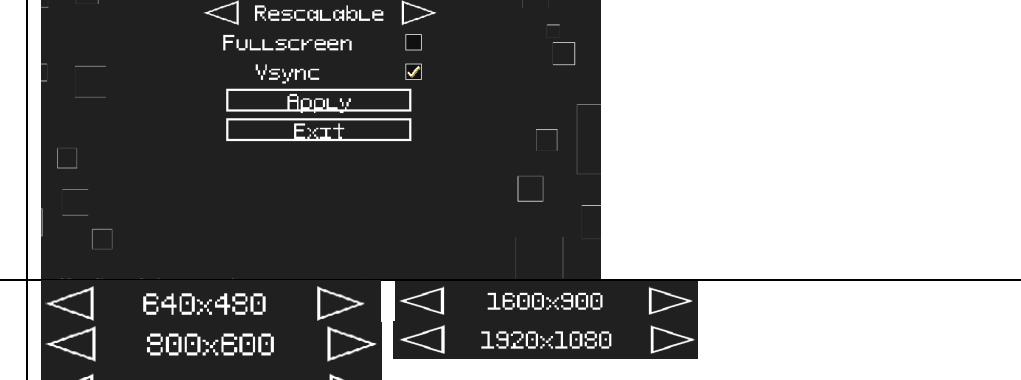
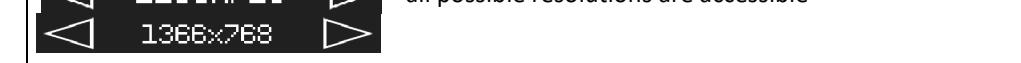
9	Start screen – seed box	Input string into width box: string123	Normal	Any string can be entered		numbers can be entered	Pass
10	Start screen – exit button	Exit button pressed	Normal	Main menu screen appears		main menu appears	Pass
11	Start screen – start button	Start button pressed	Normal	Dummy level screen appears		dummy level screen appears	Pass

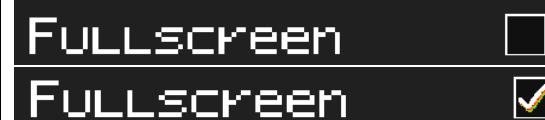
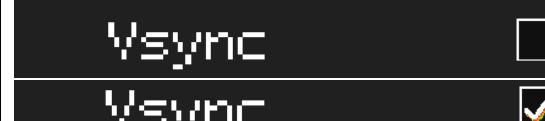
12	End screen – name box	Name string can be entered: Adam	Normal	Name string can be entered		name string can be inputted	Pass
13	End screen – name box	Strings with invalid chars can't be entered: ';/#\'	Invalid	Non name string can't be entered		Non name string can't be inputted	Pass
14	End screen – add to scoreboard button	Add to scoreboard button pressed	Normal	Scoreboard stores that row to scoreboard file		overwrites scoreboard file	<p>Fail: it hasn't maintained all previous scores Problem: when reading from a file, the last line is discarded, as it is just a \n, but the get_lines function already does this, so the last score is being removed every time: <code>for line in scores_file.readlines()[:-1]:</code></p> <p>Solution: don't remove the last line twice: <code>for line in scores_file.readlines():</code></p>
15	End screen – exit button	Exit button pressed	Normal	Main menu screen appears	 <p>Returns to level screen</p>		<p>Fail: not going to main menu screen Problem: the exit button is popping the top off the stack rather than clearing the stack and pushing the main menu screen</p> <pre># exit button if self.exit_b.falling_edges[0]: self.game.game_state_stack.pop(-1) # ESC key for event in events: if event.type == pg.KEYUP and event.key == pg.K_ESCAPE: self.game.game_state_stack.pop(-1) break</pre> <p>Solution: clear the stack and push main menu screen</p> <pre># exit button if self.exit_b.falling_edges[0]: self.game.game_state_stack = [self.game.main_screen.tick] # ESC key for event in events: if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:</pre>

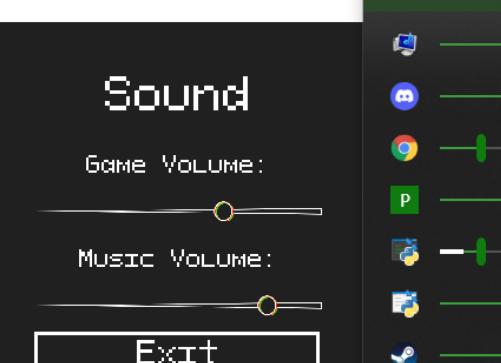
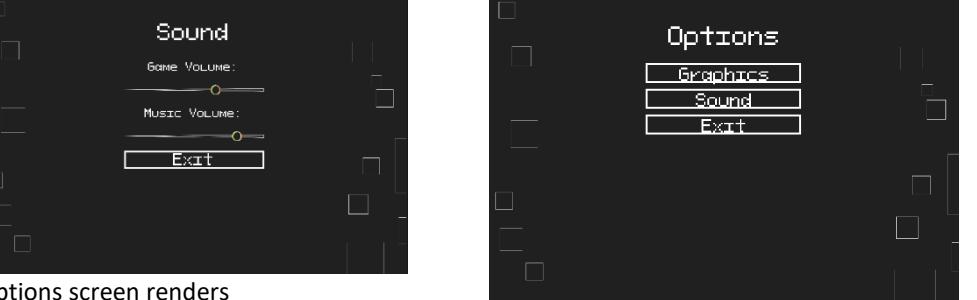
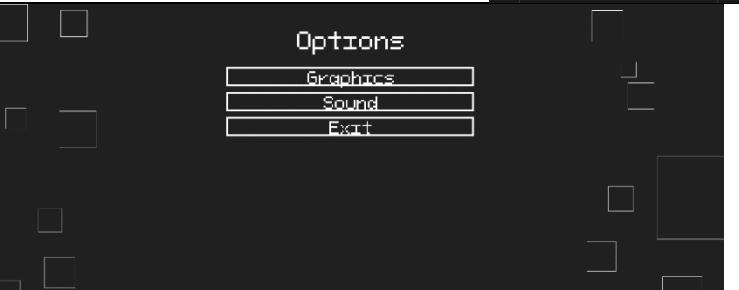
						<pre>self.game.game_state_stack = [self.game.main_screen.tick] break</pre>	
16	Scoreboard – add score	End screen's add score button is pressed with user name Adam,	Normal	Score is added to scores .csv file		Pass	
17	Scoreboard – rendering	Scoreboard button on main menu is pressed	Normal	Scoreboard appears on screen, listing scores of all previous players		Pass	
18	Scoreboard - rescaling	Screen is rescaled	Normal	Scoreboard is re rendered such that it is still centered on screen		Pass	
19	Scoreboard - saving	Game is closed	Normal	New entry is present in scoreboard.csv		new score is present in scoreboard.csv	Pass

20	Scoreboard - loading	Game is reopened and scoreboard button is pressed	Normal	All previous entries are present in the scoreboard		all entries are loaded	pass
21	Scoreboard – exit button	Exit button is pressed	normal	Returns to main menu	 returns to main menu	Pass	
22	Pause – resume button	Resume button is pressed	Normal	Returns to gameplay	 returns to level screen	Pass	
23	Pause – options button	Options button is pressed	Normal	Options screen opens	 options menu opens	Pass	

24	Pause – exit button	Exit button is pressed	Normal	Main menu screen appears	  Main menu screen opens	Pass
25	Pause screen - rescaling	Screen is rescaled	Normal	Elements are still arranged centrally on screen	  buttons are still central and correctly scaled	Pass
26	Options - rendering	Options screen is opened	Normal	Options text and buttons are on screen	  options menu is rendered on screen	Pass
27	Options – opening graphics menu	Graphics button is pressed	Normal	Graphics options screen opens	  Graphics options menu opens	Pass

28	Options – opening sound menu	Sound button is pressed	Normal	Sound options screen opens	 Sound menu opens	Pass
29	Options – exit button	Exit button is pressed	Normal	Returns to pause screen	 returns to pause screen	Pass
30	Options – rescaling	Screen is rescaled	Normal	Options text and buttons are placed on screen such that they are still central	 rescales so that the buttons are still centered on screen	Pass
31	GFX options – rendering	Graphics screen is opened	Normal	Graphics text, spinners, toggles and buttons appear centrally arranged on screen	 ui appears as expected	Pass
32	GFX options – resolution selection works	Spinner buttons are pressed	Normal	Resolution spinner can cycle between all available resolution options	 all possible resolutions are accessible	Pass

33	GFX options – fullscreen toggle works	Fullscreen toggle is pressed	Normal	Fullscreen toggle toggles between being ticked and unticked		Toggle works	Pass
34	GFX options – vsync toggle works	vsync toggle is pressed	Normal	vsync toggle toggles between being ticked and unticked		Toggle works	Pass
35	GFX options – apply buttons	Apply button is pressed	Normal	Resolution, fullscreen and vsync settings are applied, without the game crashing		all setting successfully applied	Pass
36	GFX options – exit button	Exit button is pressed	Normal	Returns to options screen		Options	Pass
37	GFX options - rescaling	Screen is rescaled	Normal	Graphics text, spinner, toggle, and buttons still appear in center of the screen.		rescales correctly to maintain centered layout	Pass
38	SND options - rendering	Sound options are opened	Normal	Sound options text, sliders and buttons render		sound options screen renders correctly	Pass

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		audio feedback is given to indicate volume of the music sound	Pass
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		audio feedback is given to indicate the volume of the game sound	Pass
41	SND options – exit button	Exit button is pressed	Normal	Options screen renders	 options screen renders	options screen renders	Pass
42	SND options – rescaling	Windows is rescaled	Normal	Sound options text, sliders and buttons render		buttons are rescaled and repositions so that they are still centered on screen	Pass
	Level – will be tested during integration testing as it is dependent on many other modules						

STAGE 7: STAKEHOLDER REVIEW

In this stage, the rest of the user interface, including navigation and flow between different menus has been developed. This functionality is very important to the overall experience of the game as it is the user interface which presents the game's functionality to the user, as it is the interface between the user and the game mechanics. To ensure that it behaves in line with how the user expects the UI to work, I have sent a copy of the test demo program to the stakeholders so they can experiment with the menu system and give their feedback:

Máté's feedback is very positive, complementing the easy to understand navigation and the UI's general responsiveness. He has uncovered a bug to do with the buttons: releasing the mouse over a button causes it to trigger even if it wasn't initially pressed down over the button, which means the user can sometimes activate multiple UI elements at once. This is caused by the button setting the falling edges value whenever the mouse button is released with the cursor over the button. To solve this, I had to add a check to see if the button was already pressed before setting the falling edges to True, now it only changes if the button was previously pressed, resolving the issue. He also spotted an issue with the sliders; they only play the release sound if the cursor is on the thumb, not if it is above or below it. This minor issue was introduced when I was trying to work around the previous bug because at the time I didn't fully understand it, and as such is easy to fix by removing a check for the cursor position being on the thumb that was implemented.

Ben's feedback

As the code has passed all the tests set out in the design section, it is complete and fully functional. Now that the stakeholders have also given their input and the code has been changed to meet their requests, it fully meets its success criteria. This means that this module is complete for now and can continue on to the next stage: integration, where it will be joined with all the other modules to create the final game!

STAGE 8: INTEGRATION

Now that all 7 modules are fully developed and tested to the satisfaction of their success criteria, it is time to integrate them into the final polished game. This will involve collecting the files that will become the final project, then working through the sections where they are to link together to enable the remaining functionality that depends on their interoperation

TASK1 : COLLECTING THE FILES:

As task 5 and 7 have significant tested and verified features, they maintain most of the functionality for the gameplay and the menu system respectively, and as such make up the bulk of the code that will go into the final build of the game. Each file that is present in both will make use of stage 7's version as that one is more up to date, with more developed within it to support all the features of the game. As such, some files will have to be modified to mitigate incompatibilities due to the changes in the more recent versions of other files.

Final build file composition:

- img folder – Stage 7
- snd folder – Stage 7
- Asset_Loader.py – Stage 7
- config.py – Stage 7
- Main.py – Stage 7
- Maze_Gen.py – Stage 5
- Menu_Sprites.py – Stage 7
- Sprites.py – Stage 5
- Scoreboard.csv – Stage 7
- Menu_System.py – Stage 7
- Sprites_Host.py – Stage 5

Sprites Host from stage 5 implements the required main loop functionality to support the game mechanics during level gameplay, so will be needed, but it will be merged into Menu_System.py: this is task 2.

TASK 2: MERGING SPRITES HOST WITH MENU SYSTEM

At the end of stage 7, menu system's Level screen was a very simple dummy implementation that just showed a magenta screen that moved onto the level complete screen if the space bar was pressed; this must now implement everything that the sprites host does, but in the framework of the ui system.

The constructor and setup method have just been copied across, now with the maze initialisation parameters passed into the constructor rather than being hard coded:

New Merged constructor and setup:

```
def __init__(self, game, size, seed):
    super().__init__(game)
    self.size = size
    self.seed = seed
    self.timer = sprites.Timer(self.game)

    self.background = self.game.img_loader.get("background")

def setup(self):
    """sets up the level"""
    # initialise camera
    self.camera = sprites.Camera(self.game)

    # start setting up the maze
    self.maze = mg.Maze(self.game, self.size, self.seed)
    # finishes generating the maze and sprites
    self.maze.setup()
    # initialise sprites in maze
    for sprite in self.maze.all_sprites:
        sprite.render(0)

    # initialise player
    self.player = sprites.Player(self.game, (self.maze.start))
    self.maze.all_sprites.add(self.player)
    # set what the camera should follow
    self.camera.set_target(self.player)
```

Sprite Host constructor and setup:

```
def __init__(self, game):
    self.game = game
    self.timer = sprites.Timer(self.game)

def setup(self):
    """sets up the level"""
    # initialise camera
    self.camera = sprites.Camera(self.game)

    # start setting up the maze
    self.maze = mg.Maze(self.game, (20,10), rng.randint(0,1000))
    # finishes generating the maze and sprites
    self.maze.setup()
    # initialise sprites in maze
    for sprite in self.maze.all_sprites:
        sprite.render(0)

    # initialise player
    self.player = sprites.Player(self.game, (self.maze.start))
    self.maze.all_sprites.add(self.player)
    # set what the camera should follow
    self.camera.set_target(self.player)
    self.camera.pos = pg.Vector2(5,5)
```

The main loop is a bit different as there was a lot of debug features implemented in Sprites Host, most of which have been removed. It was also differently structured as it had to run the main loop, but now that is done in the main file, and the level has its tick method called every frame. The level's floor had also yet to be implemented, and now has been added to the tick method:

New Tick method:

```
def tick(self, events, dt):
    super().tick(events, dt)
    self.game.screen.fill((255,0,255))

    for event in events:
        if event.type == pg.KEYUP:
            if event.key == pg.K_ESCAPE:
                self.game.game_state_stack.append(
                    self.game.pause_screen.tick)

        if event.type == pg.MOUSEBUTTONDOWN:
            if event.button == 4:
                self.camera.zoom = min(self.camera.zoom+1, 20)
            if event.button == 5:
                self.camera.zoom = max(self.camera.zoom-1 , 1)
            self.rescale()

    # update all sprites
    self.maze.all_sprites.update(dt)
    self.camera.update(dt)
    self.timer.update(dt)

    # call all sprites render method
    for sprite in self.maze.all_sprites:
        sprite.render(dt)

    self.game.screen.fill((32,32,32))
    # draw floor in correct position
    for y in range(0, self.maze.bsize[1]-1, 5):
        for x in range(0, self.maze.bsize[0]-1, 5):
            self.floor_rect.topleft = self.camera.wrld_2_scrn_coord((x,y))
            if self.floor_rect.colliderect(self.screen_rect):
                self.screen.blit(self.floor, self.floor_rect)

    self.maze.all_sprites.draw(self.game.screen)
```

Old loop method:

```
def loop(self):

    clock = pg.time.Clock()
    while True:
        dt = clock.tick(75)

        for event in pg.event.get():
            if event.type == pg.QUIT:
                return
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_t:
                    print(self.timer.total_time)
                if event.key == pg.K_r:
                    self.timer.reset()

            if event.type == pg.MOUSEBUTTONDOWN:
                if event.button == 4:
                    self.camera.zoom = min(self.camera.zoom+1, 20)
                if event.button == 5:
                    self.camera.zoom = max(self.camera.zoom-1 , 1)

    # update all sprites
    self.maze.all_sprites.update(dt)
    self.camera.update(dt)
    self.timer.update(dt)

    # call all sprites render method
    for sprite in self.maze.all_sprites:
        sprite.render(dt)

    self.game.screen.fill((32,32,32))
    self.maze.all_sprites.draw(self.game.screen)

    for sprite in self.maze.all_sprites:
        pg.draw.rect(self.game.screen, (255,255,255), sprite.hit_rect, 1)

    pg.display.flip()
```

TASK 3: IMPLEMENTING THE WIN CONDITION

When the player reaches the end of the level, the win condition is met, so the End screen should show with all the correct level statistics. To achieve this , there have been multiple alterations to multiple modules:

The Mai: Game:

The game object now has an end level method, which creates an end screen and displays it:

```
def end_level(self, size, seed, time):
    self.end_screen = MSYS.End(self, round(time), *size, seed)
    self.game_state_stack.append(self.end_screen.tick)
```

Sprites.py : Exit sprite:

The exit sprite now has some additional functionality to perform the final opening animation and indicate that it is open:

```
def update(self, dt):
    keys = self.game.level.player.keys
    player_pos = (self.game.level.player.pos+vec2(0.25,0.75))//1

    if self.opened:
        self.imgs = [self.state_imgs[-1]]
        self.frame_index = 0
    else:
        if (player_pos - self.pos).length() < 2:
            self.imgs = self.state_imgs[-2:] * 6
            if self.frame_index == len(self.imgs)-1:
                self.opened = True
        else:
            self.imgs = [self.state_imgs[keys]]
```

Menu System: Level:

Now the main loop also performs a check to see if the player is at the same location as the exit, and if they are, it plays the win sound and calls the game's end level method.:

```
# detect win condition:
if (self.player.pos+vec2(0.25,0.75))//1 == self.maze.exit.pos:
    self.win_snd.play()
    self.game.end_level(self.size, self.seed, self.timer.total_time)
```

With these features implemented, the end screen is integrated into finishing the level, meaning the player can finish a level with actual scores as expected.

TASK 4: INTEGRATING SOUND CONTROL

While the music slider currently affects the music's volume, the slider for the rest of the game sounds doesn't, because it hasn't been integrated into the sound system for the gameplay. This has been implemented as a method of the sound loader, which sets the volume of all sounds the sound loader provides:

```
def set_all_vol(self):
    # set default volume
    self.load_vol = self.game.config.game_vol
    # set volume for all loaded sounds
    for snd in self.assets.values():
        snd.set_volume(self.game.config.game_vol)
```

The load_snd_vol is then set up to call sound loader's set_all_vol method:

```
def load_snd_vol(self):
    # change game volume
    self.snd_loader.set_all_vol()
    # change music volume
    self.music.set_volume(self.config.music_vol)
```

This implements sound control for the rest of the game sounds, and as the music sound is set after all other sounds, even though it was loaded by the sound loader as well, its sound is controlled individually

TASK 5: POLISHING

Now the game is very close to done, but there are a few niceties that I would like to add to it: first, the window name:

 Colour Between The Lines  pygame window

This has been implemented using a pygame function:

```
pg.display.set_caption("Colour Between The Lines")
```

The window icon is implemented in a similar way:

```
pg.display.set_icon(self.img_loader.get("icon"))
```

 Colour Between The Lines

Now that everything has been implemented, tested, and polished, the game is now finished! This means that it can be presented as a complete program to the stakeholders for alpha testing, and from that the final review and evaluation can be written

D. EVALUATION

Now that the game is fully programmed, it must be evaluated and reviewed to verify that it amply satisfies the success criteria. This will allow me to see how well my solution matches up to the user needs, and what areas still have room for improvement which could be changed in further development. To fully evaluate how successful the game's development has been, these actions will be performed:

- Post development testing:
 - Each line of the success criteria is individually compared to the final game, and how well it fits to that line will be considered; where the game fails to meet lines, the reason for the failure will be provided and possible solutions considered
- Usability testing:

- The game will be holistically assessed to verify that it is simple, intuitive, and enjoyable to use in all cases. This will ensure that the program effectively meets the needs of all users. Any features or functionality that isn't sufficiently usable will be highlighted, the original reasoning behind them explained, and alternative, clearer options will be discussed
- Stakeholder review:
 - The final build of the game will be handed to the stakeholders to allow them to experience the whole, completed game. This will allow them to feedback on all aspects of the game and how well they interact with the game. This gives a clear picture of how well the game meets the user needs and any needs that the game fails to meet.
- Evaluation:
 - The design and development stages will be reflected on, reviewing how well the design met the success criteria, what had to be changed in the final program to improve it and how this affected the game's alignment with the user needs. The final game will also be holistically evaluated and reflected upon, considering how the individual features interact to create an enjoyable user experience. As well as success criteria that have been met, any limitations of the solution that don't meet the success criteria or only partially meet the criteria will be reviewed, the reasons why they could not be met will be explained and reflected on.
- Maintainability:
 - How the game is structured to ensure it will be easy to understand and navigate for future developers will be assessed. The file hierarchy will be discussed, explaining why it is structured and how this makes it more understandable. The programming conventions used will be explained, showing how they work together to ensure all code is predictable, understandable and easy to work on. How the modular architecture improved maintainability will be discussed, and any issues with code maintainability will be reviewed and improvements will be proposed.

POST DEVELOPMENT TESTING – SUCCESS CRITERIA

GRAPHICS - 1

PLAYER DESIGN - A

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Bright outstanding player colour scheme	Makes the player stand out from the rest of the game background	met	

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	met	
3	While walking, the player's feet animate	Makes the game much more immersive than the player sliding across the ground	met	
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	Not met	no visual indication

ENVIRONMENT DESIGN - B

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Wall sprites are square	Makes wall sprites easy to procedurally tile, which the maze population engine needs	met	

2	All types of wall sprites have the same texture	Indicates to the user that they cannot pass through this sprite	met	
3	Wall sprites are of sufficient resolution to fit the theme	Ensures that the game has enjoyable, cohesive aesthetics	met	
4	wall sprites have a dark colour	Makes the game more relaxing to look at	met	
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	met	
6	Background environment colours are dark	Makes the game more relaxing to look at	met	

ENEMY DESIGN - C

Index	Requirement	Function	Met / Partially met / not met	Evidence

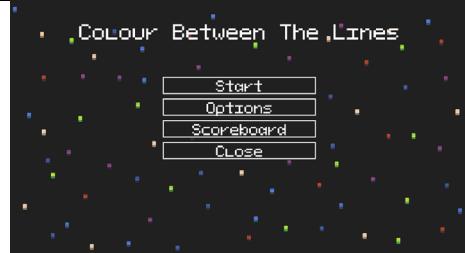
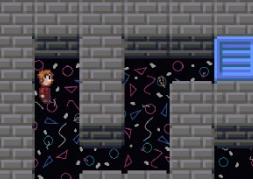
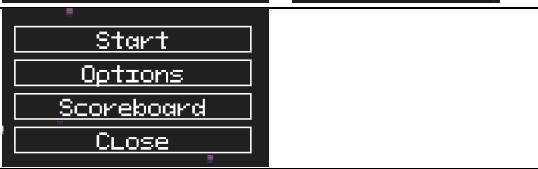
1	Dangerous colour scheme: accents and highlights are red	Intuitively indicates this sprite is dangerous	Partially met		
2	Sprite is threatening: pointy angles, sharp shading	Intuitively indicates this sprite is dangerous	met		
3	Sprite clearly indicates what state it is in	Shows user if the enemy is attacking them or not	Not met		

OBJECTIVE DESIGN - D

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Enticing colour scheme: accents and highlights in gold	Draws the player towards them, so their importance is easy to understand	Met	
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	Met	
3	Blocks and the corresponding Gateways they open are colour coded	Allows user to pair together objectives while planning how to solve the level	Met	

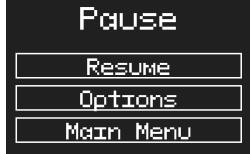
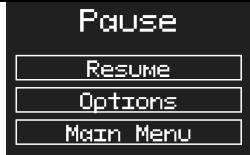
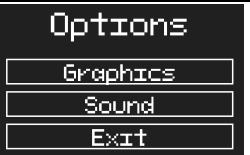
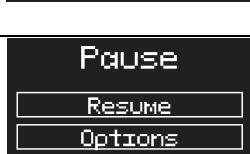
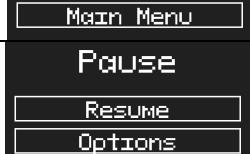
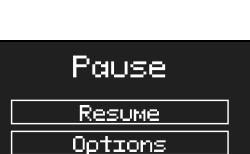
USER INTERFACE - 2

MAIN MENU - A

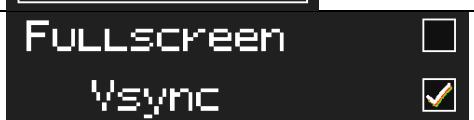
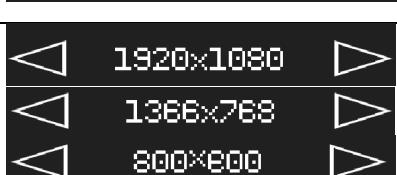
Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Background represents the game with an image of gameplay	Show the user what they are about to play, fits with graphical theme	Not met	
2	Start menu that opens a level	Allows the user to start playing a level	Met	 
3	When start button is pressed user is prompted to enter seed or allow a random seed	Allows a user to play the same level multiple times	Met	
4	Options button to open options menu	Allows user to configure game	Met	 
5	Scoreboard button that opens the locally stored scoreboard	Allows user to view previous high scores for each seed	Met	 
6	Exit button that closes the game	Allows user to exit the game	Met	

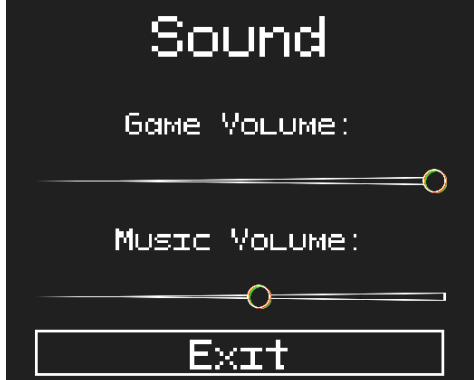
PAUSE MENU - B

Index	Requirement	Function	Met / Partially met / not met	Evidence

1	Can be opened by pressing escape	Minimises on screen UI	Met		
2	Gameplay can be resumed by pressing resume button or ESC	Allows user to return to playing the game	Met		
3	Button to access option menu	Allows user to change settings mid game	Met		
4	Button to restart level	Allows user to restart a level if they have made a mistake	Met		
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	Met		

OPTIONS MENUS - C

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Buttons to open either graphics menu or sound menu	Separates different options to make menus easier to navigate	Met	
2	Graphics menu has buttons to toggle fullscreen and vsync	Allows user to tick which settings they want enabled	Met	
3	Graphics menu has buttons to switch between available resolutions	Allows user to have the game at a good resolution for their screen	Met	
4	Graphics menu has Apply button	User can change settings without the ui rescaling	Met	

5	Sound menu has sliders for game sound and background music volume	Allows user to change the volumes of the game	Met	
6	Changes in sound menu take effect instantly	Allows user to gauge how loud it should be	Met	

GUI DESIGN - D

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Buttons highlighted in bright colours	Allows user to clearly see and distinguish the menu functionalities	Partially met	
2	Buttons provide visual feedback when hovered over by changing texture	Shows user which button they are about to press	Not Met	
3	Buttons provide visual feedback when pressed by darkening texture and moving	The user can see which buttons they are pressing	Partially met	

4	Buttons provide audible feedback when pressed	The user can hear which buttons they are pressing	met	Resume click sound heard
---	---	---	-----	------------------------------------

SOUND - 3

SPRITE SOUNDS - A

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Walking sound	Indicates when the player is walking, making game more immersive	Met	Walking sound heard
2	Injury sound	Indicates when the player takes damage	Met	Hurt sound heard
3	Respawn sound	Indicates when the player has respawned	Not met	No Respawn sound heard

LEVEL SOUNDS - B

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Block collection sound	Indicates to the user they have collected a block, so must be a positive sound	Met	Block collection sound heard when picking up blocks
2	Block placing sound	Indicates to the user they have placed a block	Met	Block collection sound heard when placing blocks
3	Exit sound	Indicates to the user that the puzzle exit has been used, and they have finished the puzzle	Met	Level complete sound heard when the player exits the level

BACKGROUND SOUNDS - C

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Relaxing Background music	Allows the user to relax while playing the game	Met	Background music is ambient and tranquil.

LEVEL DESIGN - 4

MAZE LAYOUT - A

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Maze has an entrance located on the edge	Acts as a starting place for the player to start from	Met	
2	Maze has an exit located on the edge	Acts as a final objective for the player to navigate towards	Met	
3	Maze is surrounded by walls on all sides	Stops the player from walking out of the maze, where the world isn't defined	Met	
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	Met	 No walls outside of the maze

5	There is a path from the entrance to the exit	Ensures the puzzle is solvable, otherwise the player will be frustrated	Met	
6	All parts of the maze are connected	Makes sure enemies can navigate to the player, otherwise some enemies will be useless, and will make the game slower unnecessarily	Met	
7	Maze is well populated with walls	Ensures each level is challenging and not a strait forward corridor	Met	

MAZE POPULATION - B

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Maze is still solvable	Users need to have completable puzzles or they will get frustrated	Met	

2	Blocks can be found before they must be used	Allows maze to be solvable	Met	
3	Blocks are evenly distributed throughout the maze	Ensures the maze isn't too easy to solve	Met	
4	Enemies are evenly distributed throughout the maze	Stops the user from being overwhelmed by a group of enemies	Met	

ENEMIES - C

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	Hurts player on contact, dealing damage	Ensures the enemies are dangerous, making the player avoid them	Met	 Hurt sound plays

2	Pushes player back on contact	Makes the enemy attack more realistic	Not met		Player stays still
3	Has attack cooldown	Stops them from draining player health by attacking every frame	met		Play hurt sound plays regularly not constantly

CHECKPOINTS – D

Index	Requirement	Function	Met / Partially met / not met	Evidence	
1	When the player reaches a checkpoint, it is activated	Allows the checkpoint to detect when the player has reached it	Met		
2	When the user activates a checkpoint, other checkpoints are deactivated	Ensures only one checkpoint can be enabled at a time	Met		
3	When the player dies, they respawn at the nearest checkpoint	Allows user to restart the level from the last checkpoint when they die	Met		Player respawns at the most recent check point

WIN CRITERIA - E

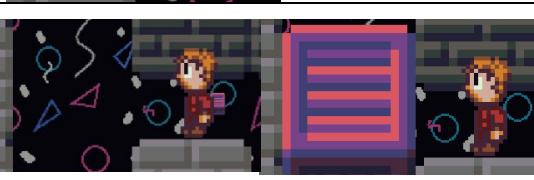
Index	Requirement	Function	Met / Partially met / not met	Evidence	

1	When player reaches a key, they pick it up	Allows the player to achieve secondary objective to enable completing the level	Met	 
2	When a player reaches an exit without all the keys, nothing happens	Ensures that the player must collect keys before trying to exit	Partially met	 exit open animation plays, but the exit can't be used
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	met	 exit opens and can be used

GAME MECHANICS - 4

PLAYER CONTROLLER - A

Index	Requirement	Function	Met / Partially met / not met	Evidence
1	When user presses arrow or wasd keys the player moves in that direction	Allows player to move around	Met	 Player moves in the correct directions
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	Met	Player accelerates up to maximum speed as expected

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	Met		all colours selectable
4	When the player hits a wall, they stop moving in the axis of collision	Stops players moving through walls	Met		Player stops moving
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	Met		Block placed in front of the player
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	Met		no block placed

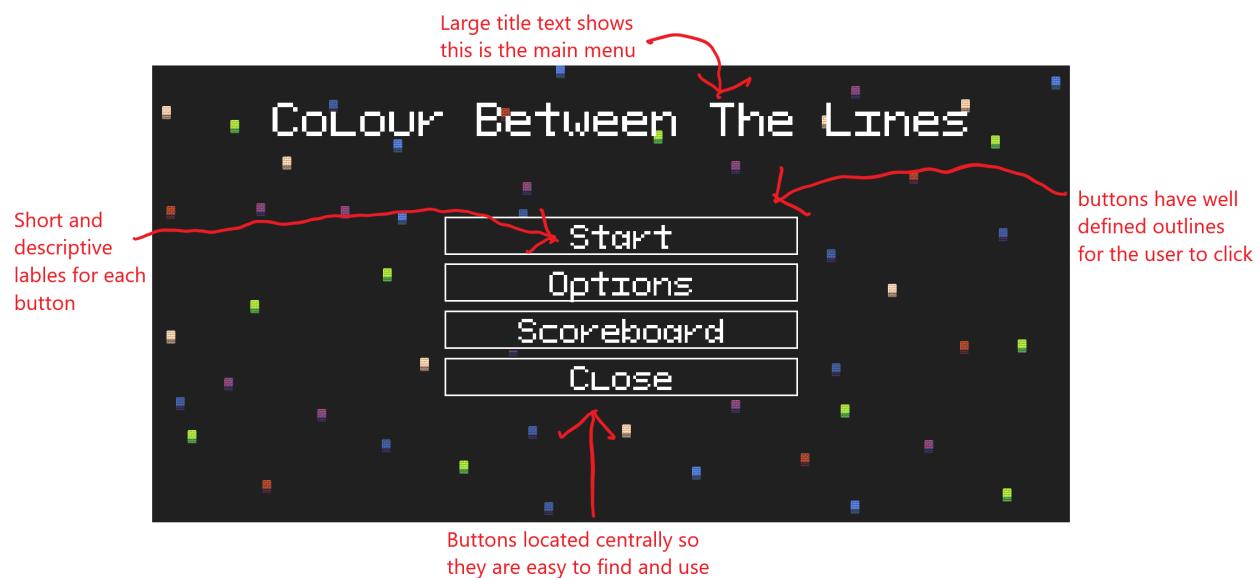
ENEMY CONTROLLER - B

Index	Requirement	Function	Met / Partially met / not met	Evidence	
1	Defines a point in the maze to go towards	Gives the Enemy a place to go to	Met		enemy goes to a specified point
2	Path finds to get to the defined point.	Acts as a simple AI for the Enemy to follow	Met		enemy goes to a specified, accessible point

3	When the player places a block, re-evaluates path	If a placed block obscures the path, the Enemy continues with a new path	Met		
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	Partially met		enemy moves with constant velocity, not accelerating

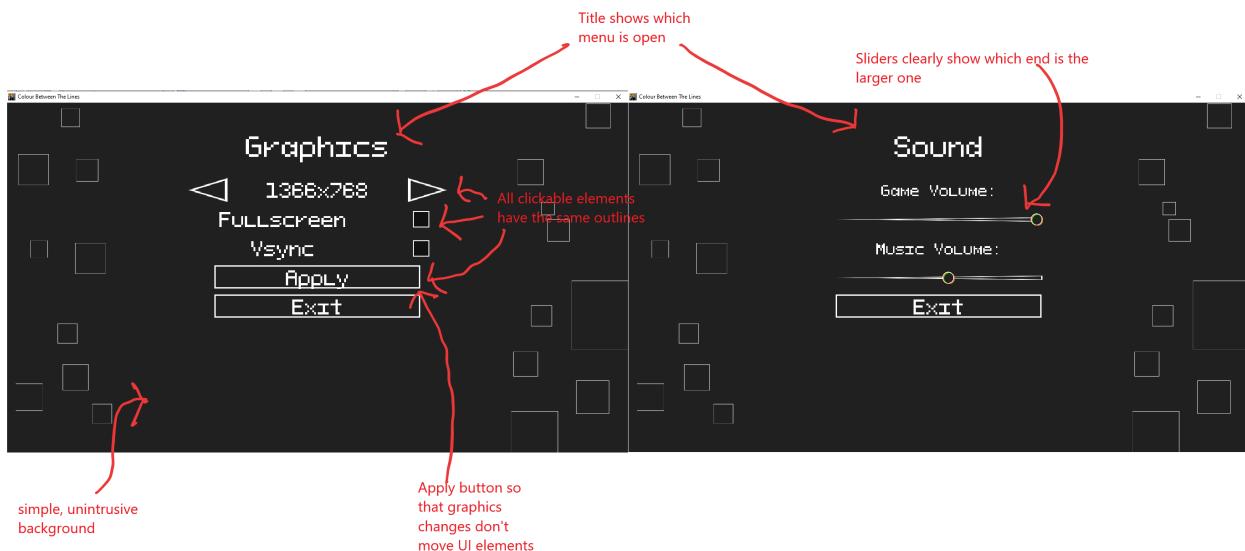
USABILITY FEATURES

Main menu:



The main menu has a simple and clear layout, with the buttons located centrally in a stacked arrangement. This is a typical ui layout found in many games; this familiarity will make it easier for the user to find the buttons. Each button is clearly labelled with what it does, making it much easier to use as opposed to symbolic buttons. The buttons can be clicked if the mouse is within the box shaped outline, and don't register a release if the mouse is released outside the box; these make the button behave simply, and robustly, stopping multiple from being clicked at once.

Options menus:

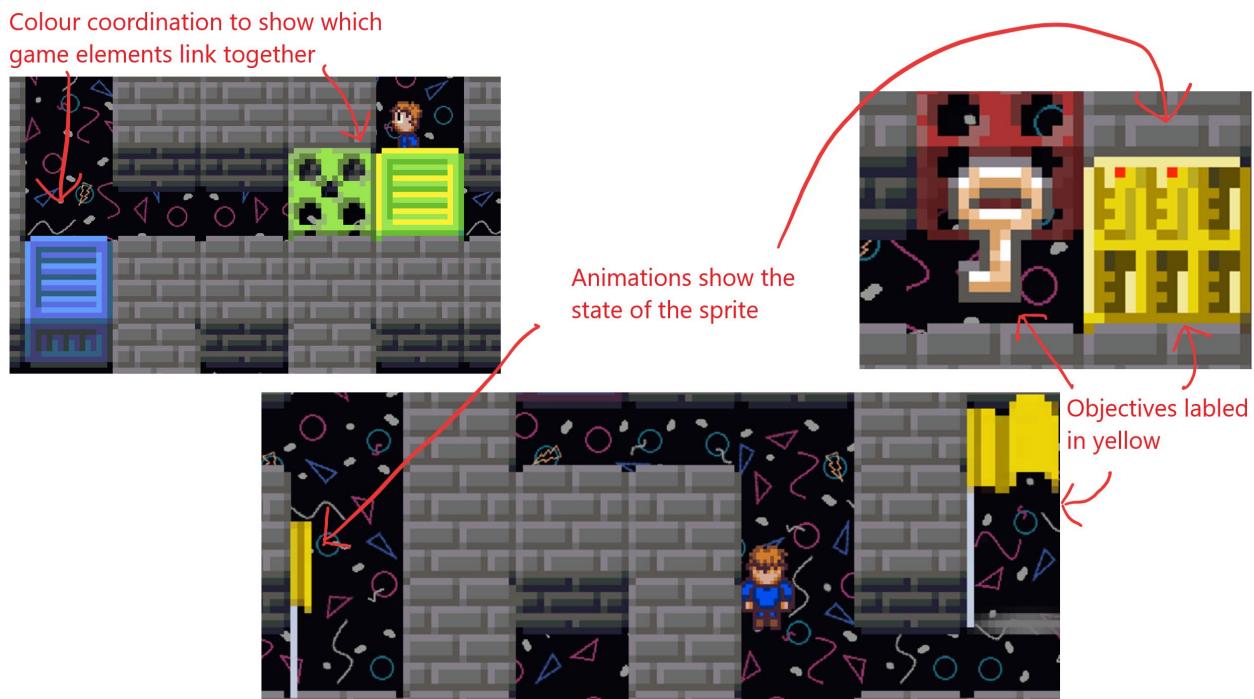


The Options menus have a similar centralised layout that places the options in the user's focus, listing them by category; this makes it easier to discern between them. Where appropriate, they are also clearly and simply named. Each of the interactive elements has a similar white outline, shows what can be clicked. The background is simple and has no features in the middle of the screen; this stops it from overlapping with the UI, which would make it hard to read.

To improve the usability of the options menus, each option could also have text bubbles that pop up when the cursor hovers over them, offering information and a description of what those options change; this would make them more understandable to new players, allowing them to get better use out of the game.

Another feature missing from the menus is a simple tutorial / controls list. This screen would be accessed from the main menu and explain to the user what the controls are, then they wouldn't have to figure them out during gameplay. It would also explain some core game concepts, showing how they could be used. Then there would be less dependence on the graphics and symbols showing the user how to use the game, and them figuring it out as they go could then be less annoying and more fun as they would know the basics to start from.

Level visuals:



The game makes strong use of colours to thematically link together related game elements such as the blocks and gateways. This shows the user that they must be used together to progress through the level. The keys and exit are also linked together through the use of the colour gold, which indicates to the user that these are sprites are important to finishing the level. While the colour scheme works well for distinguishing gameplay elements and mechanics, it doesn't take into consideration how it would be viewed with different kinds of colour blindness, and this would make the game much harder for some users. This could be resolved with a colour blindness setting in the graphics options which would alter the colour pallet to better distinguish them for users with colour blindness.

The game also makes use of a minimalistic UI with an intention to remove clutter from the screen and make the game more relaxing. This works well then fully implemented, for example the checkpoints, which have different animations depending on if they have been reached or not. However, this is a rarely used system for videogames, and that means that it is less familiar to many users, meaning they are less likely to understand the info embedded in the level than they would if it was laid out around the screen. It has also been harder to implement, meaning some info, such as player health, isn't shown due to development time constraints. Giving the user some way of knowing this would be useful for them while playing the level; this could be resolved by spending more time on animations and visuals to show this info to the user.

STAKEHOLDER REVIEW

To more directly assess how well the game meets the user needs, a compiled copy of the game has been provided to each of the stakeholders. They can then go through the full game, navigating all the menus and exploring all aspects of the gameplay, judging how each element is implemented. This means that they can comment on how well certain features are implemented and feedback about anything they find unintuitive, hard to use or any other problems they discover in the final solution.

Stakeholder	Comment	Relevant game feature	Possible improvements
Ben	Wall collisions behave unpredictably	Collisions	The collision rects for the player needs to be positioned at their feet, not at the centre
Ben	The inventory is hard to use	Inventory indication	The blocks the player picks up should be rendered on an on screen ui with labels to show what key should be pressed to play the game
Ben	The user should be able to use the scroll wheel to move through the colours	Colour selection	Detect scroll wheel events and use those to increase or decrease the colour index
Josh	It is hard to find the correct colour	Colour selection	Which key corresponds to which colour should be shown on screen
Ben	Player needs a visual indicator of when they take damage	Player controller	The player could flash red
Máté	The player movement is difficult to use as it only allows 4-way movement	Player controller	Implement 8-way movement
Máté	Player should interact with the enemies, and not be able to move past them so easily	Enemy interaction	Have the player slow down when moving past an enemy
Máté	No indication of current player health	Ui	There could either be a health bar, or a red outline on the screen when the player is low on health
Josh	The player must walk in the direction of the wall to pick up block	Player controller	If the player is colliding with the wall, their velocity is zeroed, so they face forwards; the player should instead

			remember the direction they are facing
--	--	--	--

This has highlighted 3 main issues with the game:

- The enemy – player interaction isn't well thought out
 - The enemies and the player were designed separately. This resulted in little thought being put into how they should interact with each other, so the way the player responds to an enemy is simply just a hurt sound and a loss of health, this has multiple drawbacks. Firstly, the player can just quickly sprint past the enemy and only take 1 damage, making the enemies not much of an inconvenience. Secondly, if the game audio is turned down, there is no way of telling if the player is taking damage or not. Thirdly, there is no way of knowing how many more hits the player can take before they must respawn.
 - To resolve these issues, the player should be slowed down when interacting with an enemy or perhaps knocked to the side; this would make passing the enemies much more difficult and inconvenient, encouraging avoiding them. The player should also flash red when hit; this is a common affect in games for when a sprite takes damage, so will be familiar to the users. To indicate absolute player health, there could be a small health bar in one corner of the screen; this would allow the player to see the fraction of their health they have lost. Another alternative is to have a red haze appear around the edge of the screen when the player's health is low, but this only shows low health not how much health the player starts with.
- The collisions system needs fine tuning
 - The collisions of the player use a separate rect to the actual sprite. This rect is centred on the centre of the sprite. This means it doesn't align well with the feet of the player, which are on the ground, which is used for all coordinate alignment. The collisions would therefore work much better if the collision rect was moved down a bit. Changing from 4-way motion to 8-way motion could be achieved by removing some else statements in the key registering code.
- The user interface can be hard to interpret
 - One of the core design goals of the game was a truly minimalistic UI, where all information needed to play the game is embedded in the level, not needing to be rendered atop the screen as a HUD. This has the benefit of making the screen much less cluttered, however, it comes with a variety of drawbacks.
 - It is much harder for the user to read the embedded UI by comparison to a HUD as there are common methods of implementing HUDs, which all users know about and understand, but there is no such system for embedded UIs.
 - The embedded UI requires specialised animations and graphics to show the information. As these must fit the theme of the level and be sufficiently detailed to convey what they must concisely and clearly. This makes such assets and animations hard to develop, and

as such I have neglected to develop some assets, resulting in some information, such as player health, not being represented on screen.

EVALUATION

The developed game in its current state meets a sufficient proportion of the success criteria that it is playable and is in most cases enjoyable and relaxing to work through. In order for it to meet all of its success criteria and thus become a polished, releasable game, however, there must be some changes made. Some of these are fairly minor, such as small bug fixes or adjustments and balancing of pre-existing systems, and some changes require slightly more significant design and design philosophy changes.

SUCCESES

- Menu system
 - The menus have been designed to behave intuitively and predictably, even under unusual circumstances. As it makes use of the same logic and behaviour as the windows operating system it runs on, all users are already familiar with how to use the menus.
 - Modular system has allowed easy development of many distinct menu screens, enabling a richer set of features to be easily tied into the game.
 - The options menus enable the game to be configured to the user's system and use case, affording a more comfortable user experience
- Maze generation
 - The mazes in the game are reliably generated at the size requested and contain a good distribution of keys, blocks, gateways, and enemies. This makes them enjoyable to play through as the player has to explore the whole maze before they can finish the level.
 - All mazes encountered have been solvable but not strait forward, allowing the levels to be challenging but not frustrating.
- Reliable and predictable physics
 - The physics system has been designed such that if the player moves naturally around the maze, gliding around smoothly due to the acceleration up to walking pace
 - The camera follows the player smoothly, gliding to the player's position and smoothly stopping at the edge of the maze to prevent seeing outside the maze

LIMITATIONS

- Minimal UI
 - It is hard to label the functionality with the controls they are bound to, meaning it is hard to know what the controls should be used in different situations of gameplay.
 - There is a Strong dependence on symbols to show what different visuals indicate, so the player must be familiar with what the symbols mean to understand the level

- Information is missing due to there not being enough time to develop animations to indicate information such as player health
- The game requires new users to figure out too much
 - Due to the type of game it is, there are some complexities to playing it, so the user should have the option to be briefed on how to play before starting a level. This could be solved with either a simple description of the control mapping or a small play through with a reduced level that slowly adds features on one by one.
 - A tutorial would also support the UI, explaining to the user what each feature in the game shows, and how this piece of information is important to gameplay; this would give the minimal UI a much shallower learning curve, enabling the user to start enjoying the levels sooner.
- Lack of multiple level types
 - Due to time constraints, only one level variety has been implemented
 - More levels varieties would make the game more varied as each level could have different colour schemes and different sprites to give it a distinct visual. It could also have altered maze generation.
 - The game could have a progression tree, where the user beats levels and unlocks new level types to explore; this would require a menu with a level type select that shows the level progression tree and which levels are yet to be unlocked. This would make the game a lot more rewarding to play as there would be much more of a sense of progress and accomplishment
- Lack of character types
 - Multiple character types could allow much more customisability in the levels, as the user could choose how they want their player to look like, configuring it how they want
 - The configurable characters could then be rendered in the leader board to more clearly show who is who.
- Lack of a global scoreboard
 - The scoreboard is only stored locally on the user's computer, so the scores won't be shared between 2 computers playing the game. This could be solved by having a scoreboard database stored on a cloud-based server which is read whenever the scoreboard is rendered. Then scores could be committed to this database, and they would be visible for every instance of the game.

MAINTAINABILITY

Throughout the entire development process, the game's maintainability has been a significant consideration. Maintainability is important to any coded solution as it enables the code to be changed and improved upon by future developers who have no prior understanding of how the code should work, yet the code can be quickly and easily understood on all levels, from the implementation of each function all the way up to the architecture of the whole game.

The design section of this project has presented the entire construction of the game in great detail. This has enabled the entire game to be planned well, with every module's layout and each function's order of execution already fully figured out before the first lines of code were written. This means that the design had to change very little during the development process, where most issues arose due to inconsistencies between the newly implemented code and the existing design. This document's design section therefore accurately reflects the game's source code and can thus be used as a reference for it, with the detailed explanations of all variables and subroutines for future developers to use. In the few cases where there had to be variations from the original design, such as the enemy's AI, those are clearly documented in the implementation section, which is structured in a way such that it is easy to find the implementation of any particular module.

The game is entirely modular, meaning any one module can be separated from the others and can operate in isolation with minimal supporting code. This means that future developers only ever must consider one module at once, reducing how much of the overall program they need to understand to make changes to one specific feature. It also means that large sections of the game's functionality can be reused to build other programs with similar features. For example, the menu system and UI works very well so can be reused to develop other programs with GUIs much more quickly, reusing all the functionality and replacing the assets to fit any application.

PROJECT APPENDICES

CODE LISTING

All the code for the game, along with a compiled single executable file is provided at:
<https://github.com/Scaniox/H446-programming-project>

CONFIG.PY

```
import re
from pathlib import Path
import sys

class Config():
    def __init__(self):

        # get file paths
        if getattr(sys, 'frozen', False):
            # is running in exe
            if hasattr(sys, '_MEIPASS'):
                app_path = Path(sys._MEIPASS)
                exe_parent_path = Path(sys.executable).parent
            else:
```

```
        print("can't find local path")
        input()
else:
    # is running in python interpreter
    app_path = Path(__file__).parent
    exe_parent_path = app_path

print(f"local path: {app_path}, exe parent path: {exe_parent_path}")

# file paths: they are in reference to the game root foolder
self.img_pathX = (app_path / 'img').as_posix()
self.snd_pathX = (app_path / 'snd').as_posix()
self.scoreboard_pathX = (exe_parent_path / 'scoreboard.csv').as_posix()
self.settings_save_pathX = (exe_parent_path / 'settings.set')

# graphics config
self.resolution = [1366, 768]
self.rescaleable = False
self.fullscreen = False
self.vsync = True
self.coloursX = [(0xAC, 0x32, 0x32),
                  (0xDF, 0x71, 0x26),
                  (0x99, 0XE5, 0X50),
                  (0X00, 0X50, 0xEF),
                  (0X76, 0X42, 0X8A),
                  (0X00, 0XCC, 0XCC)]
self.camera_zoom = 10
self.key_frame_count = 10
self.key_displacement = 4

# sound
self.game_vol = 1
self.music_vol = 0.25
self.player_step_snd_delay = 300

# fonts
self.text_colour = (255, 255, 255)
self.text_font_name = 'PixeloidMono-1G8ae.ttf'

# walking sprites
self.player_hurt_cooldown = 500
self.player_max_health = 5
self.player_max_speed = 0.05
```

```
self.player_acc = 0.3
self.enemy_speed = 0.02

# maze generation
self.maze_blocks_start_proportion = 0.0833333333333333
self.maze_blocks_distance_proportion = 0.16666666666666666
self.maze_gateway_jitter = 0
self.maze_gateway_skip_threshold = 0.2
self.maze_branch_stop_threshold = 0.05
self.maze_key_count = 6
self.maze_checkpoint_count = 5
self.maze_enemy_count = 6
self.walls_width_px = 16
self.walls_height_px = 24

self.load()

def save(self):
    """saves settings"""
    save_file_str = ""
    for identifier, val in self.__dict__.items():
        if identifier[-1] != "X":
            save_file_str += f"{identifier}|{repr(val)}\n"

    save_file = open(self.settings_save_pathX, "w")
    save_file.write(save_file_str)
    save_file.close()

def load(self):
    """loads settings"""
    try:
        save_file = open(self.settings_save_pathX, "r")
    except:
        return

    for line in save_file.readlines():
        if "|" in line:
            id, val = line.strip().split("|")
            exec(f"self.{id} = {val}")

    save_file.close()
```

```
import pygame as pg
from pathlib import Path
import xml.etree.ElementTree as ET

class Img_Loader():
    def __init__(self, game):
        self.game = game
        self.assets = {}
        self.sprite_sheets = []

    # load sprite sheets
    img_path = Path(self.game.config.img_pathX)
    for file_path in img_path.glob("*"):
        file_name = file_path.as_posix()
        if file_name.endswith(".xml"):
            # create a sprite sheet object for each xml file in img_path
            self.sprite_sheets.append(Sprite_Sheet(game, file_name[:-4]))

    def get(self, img_name):
        image = False
        # check already loaded assets for the image
        if img_name in self.assets.keys():
            return self.assets[img_name]

        # check for the image in the image folder
        elif (loaded_img := self.load(img_name)):
            image = loaded_img

        # try to find sprite in spritesheets
        else:
            for sheet in self.sprite_sheets:
                if loaded_image := sheet.get(img_name):
                    # when we find an image, stop looking
                    image = loaded_image
                    break

        # sprite cant be found
        if not(image):
            image = pg.surface.Surface((10, 10)).convert_alpha()
            image.fill((255, 0, 255))

        # cache and return image
        self.assets[img_name] = image
```

```
return image

def load(self, img_name):
    # search img_path for images
    img_path = Path(self.game.config.img_pathX)
    # iterate through files in img_path
    for file_path in img_path.glob("*"):
        if file_path.name.startswith(img_name):
            # if the names match, load image and return it
            image = pg.image.load(file_path.as_posix()).convert_alpha()
            return image

    # no image was found
    return False

class Snd_Loader():
    def __init__(self, game):
        self.game = game
        self.assets = {}
        self.load_vol = 1

    def get(self, snd_name):
        # check in assets
        if snd_name in self.assets.keys():
            return self.assets[snd_name]

        # try to load sound
        if loaded_sound := self.load(snd_name):
            self.assets[snd_name] = loaded_sound
            return loaded_sound

        # failed to load sound, return generic sound
        else:
            no_sound_path = Path(__file__).parent / "snd" / "no_sound.wav"
            try:
                return pg.mixer.Sound(no_sound_path.as_posix())
            except:
                print("failed to find no sound.wav in sound folder")
                input()

    def load(self, snd_name):
        snd_path = Path(self.game.config.snd_pathX)
        # iterate through all files in snd_path
```

```
for file_path in snd_path.glob("*"):
    if file_path.name.startswith(snd_name):
        # return the sound if it has the correct name
        snd = pg.mixer.Sound(file_path.as_posix())
        snd.set_volume(self.load_vol)
        return snd

def set_all_vol(self):
    # set default volume
    self.load_vol = self.game.config.game_vol
    # set volume for all loaded sounds
    for snd in self.assets.values():
        snd.set_volume(self.game.config.game_vol)

class Sprite_Sheet():
    def __init__(self, game, sheet_path):
        self.game = game
        self.sheet_path = sheet_path
        self.sprite_coords = {}

        # ensure img and xml are loadable
        img_path = Path(sheet_path + ".png")
        xml_path = Path(sheet_path + ".xml")
        if img_path.is_file() and xml_path.is_file():
            # load image
            self.img = pg.image.load(img_path.as_posix()).convert_alpha()

            # load xml
            self.load_xml(xml_path)

    def load_xml(self, xml_path):
        # load xml tree
        xml_tree_root = ET.parse(xml_path.as_posix()).getroot()
        # iterate through entries in the xml file
        for entry in xml_tree_root:
            # extract name and rect
            attributes = entry.attrib
            name = attributes["name"]
            rect = [int(attributes[i]) for i in ["x", "y", "width", "height"]]
            # store to sprite_coords
            self.sprite_coords[name] = rect
```

```
def get(self, sprite_name):
    # check the this sheet has this sprite
    if sprite_name in self.sprite_coords.keys():
        # find the rect of the requested sprite
        rect = self.sprite_coords[sprite_name]
        # gain the sprite surface
        image = self.img.subsurface(rect)
        image_copy = pg.surface.Surface(image.get_size())
        image_copy.blit(image, (0,0))
        return image

    # no sprite found
    return False
```

MAIN.PY

```
import pygame as pg
import config as cfg
import Asset_Loader as AL
import Menu_System as MSYS

class Game():
    def __init__(self):
        self.game_state_stack = []

        # init pygame env
        pg.init()
        pg.mixer.init()

        # init config
        self.config = cfg.Config()

        # init video
        self.set_screen()

        # init loaders
        self.img_loader = AL.Img_Loader(self)
        self.snd_loader = AL.Snd_Loader(self)

        # add screen's window name and icon
        pg.display.set_caption("Colour Between The Lines")
        pg.display.set_icon(self.img_loader.get("icon"))
```

```
# init screens other than level
self.main_screen = MSYS.Main(self)
self.pause_screen = MSYS.Pause(self)
self.options_screen = MSYS.Options(self)
self.gfx_options_screen = MSYS.GFX_Options(self)
self.snd_options_screen = MSYS.SND_Options(self)
self.scoreboard_screen = MSYS.Scoreboard(self)
self.start_screen = MSYS.Start(self)
self.end_screen = False
self.level = False

# load and play background music
self.music = self.snd_loader.get("Vexento - Lotus.mp3")
self.load_snd_vol()
self.music.play(loops = -1)

# push main menu onto game state stack
self.game_state_stack.append(self.main_screen.tick)

# call run
self.run()

# after running terminates, close
pg.quit()

def run(self):
    # main loop
    clock = pg.time.Clock()
    while len(self.game_state_stack) > 0:
        # calculate dt
        if self.config.vsync:
            # delay to achieve correct frame rate
            dt = clock.tick(60)
        else:
            dt = clock.tick()

        # event collect events
        event_list = list(pg.event.get())

        # check events
        for event in event_list:
            # close event: close the game
            if event.type == pg.QUIT:
                return
```

```
# rescale events: change size of the screen
elif event.type == pg.VIDEORESIZE:
    if self.config.rescaleable:
        self.config.resolution = event.size
        self.rescale()

# call correct tick function
self.game_state_stack[-1](event_list, dt)

pg.display.flip()

def start_level(self, size, seed):
    # validate
    if not 0 < size[0] <= 50 or not 0 < size[1] <= 50:
        print(f"invalid level size: {size}")
        return

    # initialise new level
    self.level = MSYS.Level(self, size, seed)
    self.level.setup()

    # push tick function to game state stack
    self.game_state_stack.append(self.level.tick)

def end_level(self, size, seed, time):
    self.end_screen = MSYS.End(self, round(time), *size, seed)
    self.game_state_stack.append(self.end_screen.tick)

def load_snd_vol(self):
    # change game volume
    self.snd_loader.set_all_vol()
    # change music volume
    self.music.set_volume(self.config.music_vol)

def set_screen(self):
    """sets the display mode based on parameters in config"""
    # fullscreen
    if self.config.fullscreen:
        desired_res = pg.display.get_desktop_sizes()[0]
        self.screen = pg.display.set_mode(desired_res,
flags=pg.FULLSCREEN)

    # rescale screen
```

```
        elif self.config.rescaleable:
            self.screen = pg.display.set_mode(self.config.resolution,
                                              pg.RESIZABLE)
        else:
            self.screen = pg.display.set_mode(self.config.resolution)

    def rescale(self):
        self.set_screen()

        pg.display.set_caption("Colour Between The Lines")
        pg.display.set_icon(self.img_loader.get("icon"))

        # call rescale method of all screens
        self.main_screen.rescale()
        self.pause_screen.rescale()
        self.options_screen.rescale()
        self.gfx_options_screen.rescale()
        self.snd_options_screen.rescale()
        self.scoreboard_screen.rescale()
        self.start_screen.rescale()
        if self.end_screen:
            self.end_screen.rescale()
        if self.level:
            self.level.rescale()

Game()
```

MAZE_GEN.PY

```
import pygame as pg
import random as rng
import Sprites as sprites

def check_collidable(sprite):
    return type(sprite).__name__ in ["Wall", "Block", "Gateway", "Exit"]

class Maze():
    def __init__(self, game, msize, seed):
        # store maze size and seed
        self.game = game
        self.msize = msize
        self.seed = seed
```

```
# initialise sprite groups
self.all_sprites = pg.sprite.LayeredUpdates()
self.maze_walls = pg.sprite.Group()
self.gateway = pg.sprite.Group()
self.blocks = pg.sprite.Group()
self.enemies = pg.sprite.Group()
self.checkpoints = pg.sprite.Group()
self.keys = pg.sprite.Group()

def setup(self):
    """Initialises the self dependent aspects of the maze:
       sprite generation is dependent on this wall object already having
       been constructed"""
    # initialise RNG
    rng.seed(self.seed)

    # generate maze layout
    self.generate_layout()

    # convert layout to wall sprites
    def wall_gen(pos):
        return sprites.Wall(self.game, pos)
    self.layout_to_board(wall_gen)

    # find start to end path
    path_end = [self.end[0]-1, self.end[1]]
    self.start_to_end_path = self.get_shortest_path(self.start, path_end)

    # populate
    self.populate()

def generate_layout(self):
    """Generates a maze layout"""
    # uses Kruskal's algorithm to generate maze layout
    # init layout array
    layout = [list([True, True, x + y * self.msize[0]])
              for x in range(0, self.msize[0]))
              for y in range(0, self.msize[1])]

    # generate list of all unchecked walls below
    unchecked_walls = []
    # - 1 stops it from checking bottom most walls
    for y in range(0, self.msize[1]-1):
        for x in range(0, self.msize[0]):
```

```
unchecked_walls.append([x,y,0])

# generate a list of unchecked walls on right
for y in range(0, self.msize[1]):
    # - 1 stops it from checking right most walls
    for x in range(0, self.msize[0]-1):
        unchecked_walls.append([x,y,1])

# iterate over all walls randomly, removing them if possible
while len(unchecked_walls) > 0:
    # select random wall
    wall = rng.choice(unchecked_walls)
    x = wall[0]
    y = wall[1]
    if wall[2]: # is left right wall
        zone1 = layout[y][x][2]
        zone2 = layout[y][x+1][2]

        # check if this wall merges zones
        if zone1 != zone2:
            # delete this wall
            layout[y][x][1] = False

            # merge zones
            for row in layout:
                for node in row:
                    if node[2] == zone2:
                        node[2] = zone1

    else: # is up down wall
        zone1 = layout[y][x][2]
        zone2 = layout[y+1][x][2]

        # check if this wall merges zones
        if zone1 != zone2:
            # delete this wall
            layout[y][x][0] = False

            # merge zones
            for row in layout:
                for node in row:
                    if node[2] == zone2:
                        node[2] = zone1
```

```
# remove wall from unchecked walls
unchecked_walls.remove(wall)

# store layout to attribute
self.layout = layout

def layout_to_board(self, wall_gen):
    """converts the maze layout to a board and sprites"""
    # adjust wall gen to append created walls to the correct groups
    def wall_gen_group(start_pos):
        wall = wall_gen(start_pos)
        self.all_sprites.add(wall)
        self.maze_walls.add(wall)
        return wall

    # generate board array
    # initialise blank array
    bsize = [2 * self.msize[i] + 1 for i in (0,1)]
    board = [list(False)
              for x in range(0, bsize[0]))
              for y in range(0, bsize[1])]

    # place perimeter sprites on board
    for x in range(0, bsize[0]): # top and bottom edges
        board[0][x] = wall_gen_group((x, 0))
        board[bsize[1]-1][x] = wall_gen_group((x, bsize[1]-1))

    for y in range(1, bsize[1]-2): # side edges
        board[y][0] = wall_gen_group((0, y))
        board[y][bsize[0]-1] = wall_gen_group((bsize[0]-1, y))
        board[bsize[1]-2][0] = wall_gen_group((0, bsize[1]-2))

    #place corner sprites on board
    for y in range(2, bsize[1], 2):
        for x in range(2, bsize[0], 2):
            board[y][x] = wall_gen_group((x,y))

    # place edge horizontal edge sprites on board
    for ly in range(self.msize[1]-1):
        for lx in range(self.msize[0]):
            by = 2*ly + 1
            bx = 2*lx + 1

            if self.layout[ly][lx][0]: # wall below
```

```
        board[by+1][bx] = wall_gen_group((bx, by+1))

# place vertical edge sprites on board
for ly in range(self.msize[1]):
    for lx in range(self.msize[0]-1):
        by = 2*ly + 1
        bx = 2*lx + 1

        if self.layout[ly][lx][1]: # wall right
            board[by][bx+1] = wall_gen_group((bx+1, by))

self.board = board
self.bsize = bsize

# generate start
self.start = [1, 1]

# generate end
self.end = [bsize[0]-1, bsize[1]-2]
self.exit = sprites.Exit(self.game, self.end)
self.all_sprites.add(self.exit)
# place exit in board
self.board[-2][-1] = self.exit

def populate(self):
    """populates the maze with sprites"""
    # populate gateways and blocks

    path_len = len(self.start_to_end_path)
    remaining_colours = [i for i in range(6)] # colours not used so far
    allowed_colours = [] # colours of blocks with no gateway

    node_index = path_len * self.game.config.maze_blocks_start_proportion

    while node_index + 1 < path_len and len(remaining_colours) > 0:
        node_index = round(node_index)

        # select current node from path
        current_node = self.start_to_end_path[node_index]
        next_node = self.start_to_end_path[node_index + 1]

        # branch
        branch_node = self.branch(current_node, [next_node])
```

```
# place block at branch_node
block_colour = rng.choice(remaining_colours)
remaining_colours.remove(block_colour)
allowed_colours.append(block_colour)

block = sprites.Block(self.game, branch_node, block_colour)
self.board[branch_node[1]][branch_node[0]] = block
self.all_sprites.add(block)
self.blocks.add(block)

# conditionally set gateway to next node along path
if rng.random() > self.game.config.maze_gateway_skip_threshold:
    gateway_colour = rng.choice(allowed_colours)
    allowed_colours.remove(gateway_colour)

    gateway = sprites.Gateway(self.game, next_node, gateway_colour)
    self.board[next_node[1]][next_node[0]] = gateway
    self.all_sprites.add(gateway)
    self.gateway.add(gateway)

# increase node_index
node_index += path_len * \
              self.game.config.maze_blocks_distance_proportion

# increment node index by a small random value
node_index += rng.random() * self.game.config.maze_gateway_jitter

# populate keys
for _ in range(self.game.config.maze_key_count):
    pos = self.random_board_spot()
    while self.board[pos[1]][pos[0]] != False:
        pos = self.random_board_spot()

    key = sprites.Key(self.game, pos)
    self.board[pos[1]][pos[0]] = key
    self.all_sprites.add(key)
    self.keys.add(key)

# populate checkpoints
for _ in range(self.game.config.maze_checkpoint_count):
    pos = self.random_board_spot()
    while self.board[pos[1]][pos[0]] != False:
        pos = self.random_board_spot()
```

```
        checkpoint = sprites.Checkpoint(self.game, pos)
        self.board[pos[1]][pos[0]] = checkpoint
        self.all_sprites.add(checkpoint)
        self.checkpoints.add(checkpoint)

        # populate enemies
        for _ in range(self.game.config.maze_enemy_count):
            pos = self.random_board_spot()
            while self.board[pos[1]][pos[0]] != False:
                pos = self.random_board_spot()

            enemy = sprites.Enemy(self.game, pos)
            self.board[pos[1]][pos[0]] = enemy
            self.all_sprites.add(enemy)
            self.enemies.add(enemy)

    def get_shortest_path(self, start, end):
        """returns (list) path from start to end"""
        # dijkstra's algorithm

        # trivial path
        if start == end:
            return [start]

        # place start node in nodes to search
        nodes_to_search = [tuple(start)]
        known_nodes = {tuple(start): False}

        while len(nodes_to_search) > 0:
            current_node_pos = nodes_to_search.pop(0)

            for offset in [(0,-1), (0,1), (1,0), (-1,0)]:
                neighbour = tuple([current_node_pos[i] + offset[i] for i in
(0,1)])
                # check neighbour is a wall
                if check_collidable(self.board[neighbour[1]][neighbour[0]]):
                    continue

                # check neighbour has already been searched
                if neighbour in known_nodes.keys():
                    continue

                # new node; add to known nodes and append to nodes_to_search
                known_nodes[tuple(neighbour)] = current_node_pos
```

```
        nodes_to_search.append(neighbour)

# use known nodes to construct a path from end to start
end_to_start = []
current_node = tuple(end)
if tuple(current_node) in known_nodes.keys():
    while known_nodes[tuple(current_node)] != False:
        end_to_start.append(current_node)
        current_node = known_nodes[current_node]

end_to_start.append(start)

# reverse end_to_start to get start_to_end
return end_to_start[::-1]
else:
    return([end])

def branch(self, start_node, known_nodes):
    """branches out from a start node to another node in the maze"""

    nodes_to_search = [start_node]

    while len(nodes_to_search) > 0:

        current_node_pos = nodes_to_search.pop(0)

        for offset in [(0,-1), (0,1), (1,0), (-1,0)]:
            neighbour = [current_node_pos[i] + offset[i] for i in (0,1)]
            neighbour = tuple(neighbour)
            # check neighbour is a wall
            if check_collidable(self.board[neighbour[1]][neighbour[0]]):
                continue

            # check neighbour has already been searched
            if neighbour in known_nodes:
                continue

            # new node; add to known nodes and append to nodes_to_search
            known_nodes.append(neighbour)
            nodes_to_search.append(neighbour)

            if rng.random() < self.game.config.maze_branch_stop_threshold:
                break
```

```
        return current_node_pos

def random_board_spot(self):
    """returns (tuple) random point on the board"""
    return [rng.randint(0, self.bsize[i]-1) for i in [0,1]]
```

SPRITES.PY

```
import math
import random as rng
import time
from numpy import size
import pygame as pg

vec2 = pg.math.Vector2
colour_names = ["red", "orange", "green", "blue", "purple", "cyan"]

def collide_hit_rect(one, two):
    return one.hit_rect.colliderect(two.hit_rect)

class Renderable_Sprite(pg.sprite.Sprite):
    def __init__(self, game, start_pos=(0,0), start_rot=0):
        super().__init__()

        # initialise
        self.game = game
        self.camera = self.game.level.camera
        self.pos = vec2(start_pos)
        self.rot = start_rot
        self.layer = int(self.pos.y)

        # animations:
        self.imgs = []
        self.frame_index = 0
        self.frame_time = 100
        self.frame_countdown = 0
        self.culling = True

    def update(self, dt):
        pass
```

```
def render(self, dt):
    # decrease frame_countdown
    self.frame_countdown -= dt
    # advance to next frame if less than 0
    if self.frame_countdown < 0:
        self.frame_countdown = self.frame_time
        self.frame_index = (self.frame_index + 1) % len(self.imgs)

    screen_pos = self.camera.wrld_2_scrn_coord(self.pos)

    # retrieve correct img from imgs
    self.image = self.imgs[self.frame_index]

    # rotating and scaling images is expensive; only do it if the sprite
    # is visible on screen; this makes the game faster at higher zooms
    if self.culling == False or \
        screen_pos.x-300 < (ssize := self.game.screen.get_size())[0] and \
        screen_pos.x+300 > 0 and \
        screen_pos.y-300 < ssize[1] and \
        screen_pos.y+300 > 0:
        # rotate and scale image
        self.image = pg.transform.rotate(self.image, self.rot)
        self.image = pg.transform.scale(self.image, [oord*self.camera.zoom
                                                    for oord in self.image.get_size()])

    # set rect position correctly
    self.rect = self.image.get_rect()
    self.rect.bottomleft = screen_pos

    # place hit_rect position correctly
    opp_corner = self.camera.wrld_2_scrn_coord(self.pos + vec2(1, -1))
    self.hit_rect = pg.Rect(0, 0, self.rect[2], self.rect[2])
    self.hit_rect.bottomleft = screen_pos

def get_facing_offset(self):
    """returns the direction the sprite is facing"""
    if 45 <= self.rot % 360 < 135:
        return vec2(1, 0)
    elif 135 <= self.rot % 360 < 225:
        return vec2(0, -1)
    elif 225 <= self.rot % 360 < 315:
        return vec2(-1, 0)
    else:
```

```
    return vec2(0,1)

class Player(Renderable_Sprite):
    def __init__(self, game, start_pos):
        # call parent constructor
        super().__init__(game, start_pos, 0)
        self.maze = self.game.level.maze

        self.colour = 0
        # set health
        self.health = game.config.player_max_health
        # set hurt cooldown
        self.hurt_cooldown = game.config.player_hurt_cooldown

        # kinematics
        self.vel = vec2(0,0)
        # set max speed
        self.max_speed = game.config.player_max_speed
        # set acc
        self.acc = game.config.player_acc

        # animations
        self.animation_state = "standing"
        # load animation frames

        right_imgs = [[self.game.img_loader.get(f"player_{colour}3")
                      for colour in colour_names], \
                      [self.game.img_loader.get(f"player_{colour}2")
                      for colour in colour_names]]
        right_imgs += right_imgs.copy()

        left_imgs = [[pg.transform.flip(img, True, False)
                     for img in frame]
                     for frame in right_imgs]

        down_imgs = [[self.game.img_loader.get(f"player_{colour}5")
                      for colour in colour_names], \
                      [self.game.img_loader.get(f"player_{colour}4")
                      for colour in colour_names]]
        down_imgs += [[pg.transform.flip(img, True, False)
                     for img in frame]
                     for frame in down_imgs]
```

```
up_imgs = [[self.game.img_loader.get(f"player_{colour}1")  
           for colour in colour_names], \  
           [self.game.img_loader.get(f"player_{colour}0")  
           for colour in colour_names]]  
up_imgs += [[pg.transform.flip(img, True, False)  
            for img in frame]  
            for frame in up_imgs]  
  
self.standing_imgs = { (1,0) : [right_imgs[1]],  
                      (-1,0) : [left_imgs[1]],  
                      (0,-1) : [down_imgs[1]],  
                      (0,1) : [up_imgs[1]]  
                    }  
self.walking_imgs = { (1,0) : right_imgs,  
                      (-1,0) : left_imgs,  
                      (0,-1) : down_imgs,  
                      (0,1) : up_imgs  
                    }  
  
# sounds  
self.walk_sounds = []  
for i in range(1,9):  
    self.walk_sounds.append(self.game.snd_loader.get( \  
                                              f"stepdirt_{i}.wav"))  
self.key_collect_snd = self.game.snd_loader.get("key_pickup.wav")  
self.block_collect_snd = self.game.snd_loader.get("stepstone_4.wav")  
self.block_place_snd = self.game.snd_loader.get("stepstone_1.wav")  
self.gateway_open_snd = self.game.snd_loader.get("DoorOpen07.ogg")  
self.respawn_snd = self.game.snd_loader.get("Hit_Hurt6.wav")  
self.hurt_snd = self.game.snd_loader.get("Hit_Hurt6.wav")  
  
self.step_sound_timer = self.game.config.player_step_snd_delay  
  
# set up other mechanics  
self.inventory = [False, False]  
# stores the previous key state for edge detection  
self.prev_slot_keys = [False, False]  
# stores colour keys state:  
self.colour_key_pressed = False  
  
self.keys = 0  
self.last_checkpoint = False  
  
# render once to set up rect for collisions  
self.walking = False
```

```
self.render(0)

def update(self, dt):
    # acceleration due to wasd
    keys = pg.key.get_pressed()
    walking_x = False
    walking_y = False
    if keys[pg.K_LEFT] or keys[pg.K_a]:
        # acc left
        self.vel.x -= self.acc * dt/1000
        walking_x = True
    elif keys[pg.K_RIGHT] or keys[pg.K_d]:
        # acc right
        self.vel.x += self.acc * dt/1000
        walking_x = True

    elif keys[pg.K_UP] or keys[pg.K_w]:
        # acc up
        self.vel.y -= self.acc * dt/1000
        walking_y = True
    elif keys[pg.K_DOWN] or keys[pg.K_s]:
        # acc down
        self.vel.y += self.acc * dt/1000
        walking_y = True

    # decelerate back to vel = 0 if not walking
    if not walking_x:
        self.vel.x -= min(self.acc * self.vel.x, self.vel.x*0.9,
                          key = lambda x: abs(x))
    if not walking_y:
        self.vel.y -= min(self.acc * self.vel.y, self.vel.y*0.9,
                          key = lambda x: abs(x))
    self.walking = walking_x or walking_y

    # enforce max speed
    self.vel = vec2([min(max(-self.max_speed, self.vel[i]), self.max_speed) \
                    for i in (0,1)]) 

    # block picking and placing
    slot_keys = [keys[pg.K_q], keys[pg.K_e]]
    for slot_index in (0,1):
        # only trigger on the rising edge of the key press
        if slot_keys[slot_index] and not self.prev_slot_keys[slot_index]:
            # if the slot is empty, pick up
```

```
        if self.inventory[slot_index] == False:
            self.pick_up(slot_index)
            # if slot is full, place
        else:
            self.place(slot_index)
        self.prev_slot_keys = slot_keys

    # get which keys are pressed for colour changing
    colour_keys = [keys[k] for k in (pg.K_1, pg.K_2, pg.K_3,
                                      pg.K_4, pg.K_5, pg.K_6)]
    # only change colour on rising edge a a key being pressed
    if not self.colour_key_pressed:
        for key_index in range(0,6):
            if colour_keys[key_index]:
                self.colour = key_index

    self.colour_key_pressed = max(colour_keys)

    # collisions with keys
    hits = pg.sprite.spritecollide(self,
                                    self.maze.keys,
                                    True,
                                    collide_hit_rect)
    self.keys += len(hits)
    if len(hits):
        self.key_collect_snd.play()

    # collisions with enemies
    hits = pg.sprite.spritecollide(self,
                                    self.maze.enemies,
                                    False,
                                    collide_hit_rect)
    if [h for h in hits if h.colour != self.colour]:
        # only take damage if hurt cooldown has elapsed
        if self.hurt_cooldown <= 0:
            # reset hurt cooldown
            self.hurt_cooldown = self.game.config.player_hurt_cooldown
            # take damage
            self.health -= 1
            # play hurt sound
            self.hurt_snd.play()

    # decrease hurt cooldown
    self.hurt_cooldown = max(0, self.hurt_cooldown - dt)
```

```
# play step sound if step sound timer has elapsed
if self.walking:
    if self.step_sound_timer <= 0:
        rng.choice(self.walk_sounds).play()
        self.step_sound_timer = self.game.config.player_step_snd_delay
    else:
        self.step_sound_timer -= dt
else:
    self.step_sound_timer = 0

# update rot based on movement
if self.vel.length != 0:
    self.rot = self.vel.angle_to(vec2(0,1))

# collide with walls
self.collide()

# change position by velocity
self.pos += self.vel

# respawn
if self.health == 0:
    self.respawn()

# change layer
self.maze.all_sprites.change_layer(self, int(self.pos.y))

def pick_up(self, slot_index):
    """picks up block in front of the player and stores in inventory"""
    # slot is already full
    if self.inventory[slot_index]:
        return

    # find block in front of the player
    facing_pos = (self.pos+vec2(0.25,0.75))/1 + self.get_facing_offset()
    facing_sprite = self.maze.board[int(facing_pos.y)][int(facing_pos.x)]

    # check if it is a block
    if type(facing_sprite).__name__ == "Block":
        # store this block to inventory
        self.inventory[slot_index] = facing_sprite

    # remove this block from board
```

```
        self.maze.board[int(facing_pos.y)][int(facing_pos.x)] = False

        # remove block from all sprites and blocks
        self.maze.blocks.remove(facing_sprite)
        self.maze.all_sprites.remove(facing_sprite)

        # play block collection sound
        self.block_collect_snd.play()

    def place(self, slot_index):
        """places a block in front of the player from inventory slot"""
        # nothing to place
        if self.inventory[slot_index] == False:
            return

        # find block in front of the player
        facing_pos = (self.pos+vec2(0.25,0.75))//1 + self.get_facing_offset()
        facing_sprite = self.maze.board[int(facing_pos.y)][int(facing_pos.x)]

        # ensure the space is free before placing in a free space
        if facing_sprite == False:
            # remove block from inventory
            block = self.inventory[slot_index]
            self.inventory[slot_index] = False

            # set block's new position
            block.pos = facing_pos//1

            # store block in inventory there
            self.maze.board[int(facing_pos.y)][int(facing_pos.x)] = block

            # add block to all sprites and blocks
            self.maze.all_sprites.add(block)
            self.maze.blocks.add(block)

            # change block's layer so it renders correctly
            self.maze.all_sprites.change_layer(block, int(block.pos.y))

            # play placing sound
            self.block_place_snd.play()

        # if space isn't free, check if it is a gateway
        if type(facing_sprite).__name__ == "Gateway" and \
            facing_sprite.colour == self.inventory[slot_index].colour:
```

```
# remove both the block and gateway
self.inventory[slot_index].kill()
self.inventory[slot_index] = False

facing_sprite.kill()
self.maze.board[int(facing_pos.y)][int(facing_pos.x)] = False
self.gateway_open_snd.play()

def respawn(self):
    """respawns the player at the correct location when they die"""
    # play respawn sound
    self.respawn_snd.play()

    # set health to player max health
    self.health = self.game.config.player_max_health

    if self.last_checkpoint != False:
        self.pos = vec2(self.last_checkpoint.pos)
    else:
        self.pos = vec2(self.maze.start)

def collide(self):
    # overcomplicated collisions to remove weird snapping
    # - no dependency on velocities, which can cause problems

def check_collidable(sprite):
    return type(sprite).__name__ == "Wall" or \
           (type(sprite).__name__ == "Block" and
            sprite.colour != self.colour) or \
           type(sprite).__name__ == "Gateway" or \
           type(sprite).__name__ == "Exit" and self.keys != 6

board = self.maze.board

for sprite in self.maze.all_sprites:
    if collide_hit_rect(sprite, self) and check_collidable(sprite):

        if self.hit_rect.x+10 > sprite.hit_rect.right:
            # wall on left
            spot = sprite.pos//1 + vec2(1,0)

            # collide if outside the bounds or isn't a collidable
            if not(0 <= spot.x < self.maze.bsize[0]) or \
               not(check_collidable(board[int(spot.y)][int(spot.x)])):
```

```
        self.vel.x = max(0, self.vel.x)

    elif self.hit_rect.bottom-10 < sprite.hit_rect.y:
        # wall below
        spot = sprite.pos//1 + vec2(0,-1)

        # collide if outside the bounds or isnt a collideable
        if not(0 <= spot.y < self.maze.bsize[0]) or \
            not(check_collidable(board[int(spot.y)][int(spot.x)])):
            self.vel.y = min(0, self.vel.y)

    elif self.hit_rect.right-10 < sprite.hit_rect.x:
        # wall on right
        spot = sprite.pos//1 + vec2(-1,0)

        # collide if outside the bounds or isnt a collideable
        if not(0 <= spot.x < self.maze.bsize[0]) or \
            not(check_collidable(board[int(spot.y)][int(spot.x)])):
            self.vel.x = min(0, self.vel.x)

    elif self.hit_rect.y+10 > sprite.hit_rect.bottom:
        # wall above
        spot = sprite.pos//1 + vec2(0,1)

        # collide if outside the bounds or isnt a collideable
        if not(0 <= spot.y < self.maze.bsize[0]) or \
            not(check_collidable(board[int(spot.y)][int(spot.x)])):
            self.vel.y = max(0, self.vel.y)

def render(self, dt):
    """render the player sprite"""
    facing_dir = self.get_facing_offset()

    # retrieve correct set of imgs for player's current rotation
    if self.walking:
        self.imgs = self.walking_imgs[tuple(facing_dir)]
    else:
        self.imgs = self.standing_imgs[tuple(facing_dir)]
        self.frame_index = 0

    # decrease frame_countdown
    self.frame_countdown -= dt
    # advance to next frame if less than 0
    if self.frame_countdown < 0:
```

```
        self.frame_countdown = self.frame_time
        self.frame_index = (self.frame_index + 1) % len(self.imgs)

        # retrieve correct img from imgs
        self.image = self.imgs[self.frame_index][self.colour]
        self.image = pg.transform.scale(self.image,
                                       [oord * self.camera.zoom // 2 for
                                        oord in self.image.get_size()])

        # data for rendering of blocks in inventory
        b1_pos = {(0,-1) : vec2(3,6),
                   (1,0)  : vec2(0,6),
                   (-1,0) : vec2(6,6)}
        b2_pos = {(0,-1) : vec2(3,4.5),
                   (1,0)  : vec2(0,4.5),
                   (-1,0) : vec2(6,4.5)}

        # render blocks in inventory so that they scale correctly
        for block in self.inventory:
            if block:
                block.render(0)

        # no image if facing down
        if facing_dir != (0,1):
            # work out the position and size of each block's image
            if block_1 := self.inventory[0]:
                block_1_image = pg.transform.scale(
                    block_1.image, vec2(block_1.image.get_size())//8)

                block_1_pos = b1_pos[tuple(facing_dir)] * self.camera.zoom

            if block_2 := self.inventory[1]:
                block_2_image = pg.transform.scale(
                    block_2.image, vec2(block_2.image.get_size())//8)

                block_2_pos = b2_pos[tuple(facing_dir)] * self.camera.zoom

            # blit on top if facing up, otherwise blit behind
            if facing_dir == (0,-1):
                if block_1:
                    self.image.blit(block_1_image, block_1_pos)
                if block_2:
                    self.image.blit(block_2_image, block_2_pos)
            else:
```

```
        image = pg.surface.Surface(self.image.get_size(),
                                    flags = pg.SRCALPHA)
        if block_1:
            image.blit(block_1_image,block_1_pos)
        if block_2:
            image.blit(block_2_image, block_2_pos)
        image.blit(self.image, (0,0))
        self.image = image

        # set rect position correctly
        self.rect = self.image.get_rect()
        screen_pos = self.camera.wrld_2_scrn_coord(self.pos)
        self.rect.bottomleft = screen_pos

        # place hit_rect position correctly
        opp_corner = self.camera.wrld_2_scrn_coord(self.pos + vec2(1,-1))
        self.hit_rect = pg.Rect(0,0,self.rect[2], self.rect[2])
        self.hit_rect.bottomleft = screen_pos

class Enemy(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)
        self.colour = rng.randint(0,5)
        self.maze = self.game.level.maze

        # load all animation frames from img loader

        left_imgs = [[self.game.img_loader.get(f"enemy_{colour}{i}")
                      for colour in colour_names]
                     for i in range(4)]

        right_imgs = [[pg.transform.flip(img, True, False)
                      for img in frame]
                     for frame in left_imgs]

        up_imgs = [[self.game.img_loader.get(f"enemy_{colour}6")
                    for colour in colour_names], \
                   [self.game.img_loader.get(f"enemy_{colour}7")]
                    for colour in colour_names]]
        up_imgs += [[pg.transform.flip(img, True, False)
                     for img in frame]
                     for frame in up_imgs]
```

```
down_imgs = [[self.game.img_loader.get(f"enemy_{colour}4")
              for colour in colour_names], \
              [self.game.img_loader.get(f"enemy_{colour}5")
              for colour in colour_names]]
down_imgs += [[pg.transform.flip(img, True, False)
               for img in frame]
               for frame in down_imgs]

self.walking_imgs = {(-1,0) : left_imgs,
                     (1,0) : right_imgs,
                     (0,1) : down_imgs,
                     (0,-1) : up_imgs
                     }

# get a path
self.get_path()

def get_path(self):
    """calculates a new path for this sprite to follow"""

    current_pos = (self.pos+vec2(0.25,0.75))//1
    current_pos = (int(current_pos.x), int(current_pos.y))
    destination = self.maze.branch(current_pos, [])

    # find path to this location
    self.target_path = self.maze.get_shortest_path(current_pos, destination)

def update(self, dt):
    # get direction to current target
    target = vec2(self.target_path[0]) + vec2(0.25, -0.25)
    target_delta = target - self.pos

    # if it has reached the target, remove it from target_path
    if target_delta.length() < 0.05:
        self.target_path.pop(0)

    # if the whole path has been followed, generate a new one
    if len(self.target_path) == 0:
        self.get_path()

    # recompute target
    target = vec2(self.target_path[0])
    target_delta = target - self.pos
```

```
# move towards target
if target_delta.length() != 0:
    self.vel = target_delta.normalize() * self.game.config.enemy_speed
    self.rot = self.vel.angle_to(vec2(0,1))
    self.pos += self.vel

# change layer
self.maze.all_sprites.change_layer(self, int(self.pos.y))

def render(self, dt):
    """render the enemy sprite"""
    facing_dir = self.get_facing_offset()
    self.imgs = self.walking_imgs[tuple(facing_dir)]

    # decrease frame_countdown
    self.frame_countdown -= dt
    # advance to next frame if less than 0
    if self.frame_countdown < 0:
        self.frame_countdown = self.frame_time
        self.frame_index = (self.frame_index + 1) % len(self.imgs)

    # retrieve correct img from imgs
    self.image = self.imgs[self.frame_index][self.colour]
    self.image = pg.transform.scale(self.image,
                                   [oord * self.camera.zoom // 1.5 for
                                    oord in self.image.get_size()])

    # set rect position correctly
    self.rect = self.image.get_rect()
    screen_pos = self.camera.wrld_2_scrn_coord(self.pos)
    self.rect.bottomleft = screen_pos

    # place hit_rect position correctly
    opp_corner = self.camera.wrld_2_scrn_coord(self.pos + vec2(1,-1))
    self.hit_rect = pg.Rect(0,0,self.rect[2], self.rect[2])
    self.hit_rect.bottomleft = screen_pos

class Wall(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

    # initialise assets
```

```
self.imgs = [game.img_loader.get("brick dark grey")]

class Gateway(Renderable_Sprite):
    def __init__(self, game, start_pos, colour):
        super().__init__(game, start_pos)
        self.colour = colour

        # initialise assets
        self.imgs = [game.img_loader.get(
            f"gateway_{colour_names[self.colour]}")]

class Block(Renderable_Sprite):
    def __init__(self, game, start_pos, colour):
        super().__init__(game, start_pos)
        self.colour = colour
        self.culling = False

        # initialise assets
        match self.colour:
            case 0:
                self.imgs = [game.img_loader.get("crate light red")]
            case 1:
                self.imgs = [game.img_loader.get("crate orange")]
            case 2:
                self.imgs = [game.img_loader.get("crate lime")]
            case 3:
                self.imgs = [game.img_loader.get("crate dark blue")]
            case 4:
                self.imgs = [game.img_loader.get("crate purple")]
            case 5:
                self.imgs = [game.img_loader.get("crate light blue")]

class Checkpoint(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        self.active = False

        # init active images
        self.active_imgs = []
        for img_index in range(0,6):
```

```
        img = self.game.img_loader.get(f"Flag{img_index}")
        self.active_imgs.append(img)
    # init deactivate images
    self.deactive_imgs = []
    for img_index in range(0,2):
        img = self.game.img_loader.get(f"Flag_down{img_index}")
        self.deactive_imgs.append(img)
    # innit
    self.imgs = self.deactive_imgs

def update(self, dt):
    if not self.active:
        if (self.pos - self.game.level.player.pos).length() < 0.5:
            # deactivate currently active checkpoint
            if last_checkpoint := self.game.level.player.last_checkpoint:
                last_checkpoint.deactivate()

            # activate this checkpoint
            self.activate()

    def activate(self):
        """activates this checkpoint"""
        self.active = True
        self.imgs = self.active_imgs
        self.game.level.player.last_checkpoint = self

    def deactivate(self):
        """deactivates this checkpoint"""
        self.active = False
        self.imgs = self.deactive_imgs
        self.frame_index = 0

class Key(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        key_frame_count = self.game.config.key_frame_count
        key_displacement = self.game.config.key_displacement

        # generate images
        key_image = self.game.img_loader.get("key_cream")
        key_image = pg.transform.scale(key_image, vec2(key_image.get_size()))
        self.imgs = []
```

```
for i in range(key_frame_count):
    displaced_img = pg.surface.Surface(
        key_image.get_size() + vec2(0, key_displacement),
        flags = pg.SRCALPHA)

    # calculate how far this image must be moved
    offset = (math.cos(2*math.pi * (i/key_frame_count) ) + 1) \
              * key_displacement // 2

    # blit key image to displaced image in correct location
    displaced_img.blit(key_image, (0, offset))

    # append to imgs
    self.imgs.append(displaced_img)

class Exit(Renderable_Sprite):
    def __init__(self, game, start_pos):
        super().__init__(game, start_pos)

        # load images
        self.state_imgs = []
        for img_index in range(0,7):
            self.state_imgs.append(
                self.game.img_loader.get(f"exit_locked{img_index}"))
        self.state_imgs.append(self.game.img_loader.get("exit_open"))
        self.imgs = [self.state_imgs[0]]
        self.opened = False

    def update(self, dt):
        keys = self.game.level.player.keys
        player_pos = (self.game.level.player.pos+vec2(0.25,0.75))//1

        if self.opened:
            self.imgs = [self.state_imgs[-1]]
            self.frame_index = 0
        else:
            if (player_pos - self.pos).length() < 2:
                self.imgs = self.state_imgs[-2:] * 6
                if self.frame_index == len(self.imgs)-1:
                    self.opened = True
            else:
                self.imgs = [self.state_imgs[keys]]
```

```
class Camera():
    def __init__(self, game, target = False):
        self.game = game
        self.target = target
        self.pos = vec2(0,0) # this location is the centre of the screen
        self.zoom = self.game.config.camera_zoom

        # invisible default image to support being part of all sprites
        self.img = pg.surface.Surface((1,1))
        self.img.fill((0,0,0))
        self.rect = self.img.get_rect()
        self.rect.topleft = (-1000,-1000)

    def set_target(self, target):
        """sets the sprite which the camera should follow"""
        self.target = target

    def update(self, dt):
        """updates the position of the camera so that it tracks the player"""

        # adjust the camera pos
        target_pos = vec2(self.target.pos)
        target_pos_delta = -(target_pos - self.pos)
        self.pos = self.pos - 0.1*target_pos_delta

        # ensure camera never goes off screen
        unscaled_scrn_size = vec2(self.game.config.resolution)/ self.zoom / 16
        wrld_size = vec2(self.game.level.maze.bsize)

        left_edge = unscaled_scrn_size.x/2
        right_edge = wrld_size.x - unscaled_scrn_size.x/2
        top_edge = unscaled_scrn_size.y/2 - 1.4
        bottom_edge = wrld_size.y - unscaled_scrn_size.y/2 - 1.3

        self.pos.x = min(max(left_edge, self.pos.x), right_edge)
        self.pos.y = min(max(top_edge, self.pos.y), bottom_edge)

    def wrld_2_scrn_coord(self, wrld_coord):
        """takes a world space coordinate and converts it to screenspace"""
        scrn_size = vec2(self.game.config.resolution)
```

```
# ensures that the cameras position ends up at the centre of the screen
scaled_wrld_coord = vec2(wrld_coord) * self.zoom * 16
scaled_pos = self.pos * self.zoom * 16
ss_coord = scaled_wrld_coord + scrn_size/2 - scaled_pos
return ss_coord

class Timer():
    def __init__(self, game):
        self.game = game
        self.reset()

        # invisible default image to support being part of all sprites
        self.img = pg.surface.Surface((1,1))
        self.img.fill((0,0,0))
        self.rect = self.img.get_rect()
        self.rect.topleft = (-1000,-1000)

    def update(self, dt):
        # dt is in ms, but total time is in s so dt is scaled correctly
        self.total_time += dt/1000

    def reset(self):
        self.total_time = 0.0
        self.start_time = time.time()
```

MENU_SPRITES.PY

```
import pygame as pg
from pathlib import Path

vec2 = pg.math.Vector2

k2c_numeric = [(f"{i}") for i in range(10)]
k2c_alpha_lower = [chr(i) for i in range(ord("a"), ord("z")+1)]
k2c_alpha_upper = [chr(i) for i in range(ord("A"), ord("Z")+1)]
k2c_all = ["all"]

class Text(pg.sprite.Sprite):
    def __init__(self, game, start_rect, text):
        super().__init__()
        # store text to attribute
        self.game = game
        self.text = text
```

```
# get font name
font_name = self.game.config.text_font_name
font_path = Path(self.game.config.img_pathX) / font_name

# load font
self.font = pg.font.Font(font_path.as_posix(), 20)
self.text_colour = self.game.config.text_colour

# rescale to generate image
self.rect = start_rect
self.rescale()

def update(self, dt, events):
    # empty function to ensure functionality
    pass

def rescale(self):
    # render text to image
    text_img = self.font.render(self.text, False, self.text_colour)
    scale_factor = self.rect.height / text_img.get_height()
    text_size = [text_img.get_width() * scale_factor, self.rect.height]
    text_img = pg.transform.scale(text_img, text_size)
    text_img_rect = text_img.get_rect()

    # scale image to width and height of rect
    self.image = pg.surface.Surface(self.rect.size, flags=pg.SRCALPHA)
    text_img_rect.center = vec2(self.rect.size) / 2
    self.image.blit(text_img, text_img_rect.topleft)

class Input_Box(pg.sprite.Sprite):
    def __init__(self, game, start_rect, default_text, allowed_keys ):
        super().__init__()
        # store attributes
        self.game = game
        self.default_text = default_text
        self.text = ""
        self.allowed_keys = allowed_keys
        self.selected = False

        # get font name
        font_name = self.game.config.text_font_name
```

```
font_path = Path(self.game.config.img_pathX) / font_name

# load font
self.font = pg.font.Font(font_path.as_posix(), 20)
self.text_colour = self.game.config.text_colour

# rescale to generate image
self.rect = pg.rect.Rect(start_rect)
self.rescale()

def update(self, dt, events):
    for event in events:
        # set selected to true if cursor in rect
        # and left mouse button clicked, otherwise false
        if event.type == pg.MOUSEBUTTONDOWN and event.button == 1:
            mouse_pos = pg.mouse.get_pos()
            if self.rect.collidepoint(mouse_pos):
                self.selected = True
            else:
                self.selected = False
            self.rescale()

        # register key presses
        if event.type == pg.KEYDOWN and self.selected:
            # if key is backspace, remove last char from text
            if event.unicode == '\x08':
                self.text = self.text[:-1]
            # if key is in keys to chars, append to keys
            elif event.unicode in self.allowed_keys or \
                  "all" in self.allowed_keys:
                self.text += event.unicode
            self.rescale()

    def rescale(self):
        # if self.text isn't empty, render text
        if len(self.text) > 0:
            render_text = self.text
            # if self.selected, append _ to the end of the text to show this
            if self.selected:
                render_text += "_"
            # if self.text is empty, render default text
            else:
                render_text = self.default_text
```

```
# render text to image
text_img = self.font.render(render_text, False, self.text_colour)
scale_factor = self.rect.height / text_img.get_height()
text_size = [text_img.get_width() * scale_factor, self.rect.height]
text_img = pg.transform.scale(text_img, text_size)
text_img_rect = text_img.get_rect()

# scale image to width and height of rect
self.image = pg.surface.Surface(self.rect.size, flags=pg.SRCALPHA)
self.image.fill((32,32,32))
if self.selected:
    pg.draw.rect(self.image, (128,128,128), [0,0,*self.rect.size], 1)
text_img_rect.center = vec2(self.rect.size) / 2
self.image.blit(text_img, text_img_rect.topleft)

class Toggle(pg.sprite.Sprite):
    def __init__(self, game, start_rect):
        super().__init__()
        self.game = game

        # initialise ticked to false
        self.ticked = False

        self.ticked_img = self.game.img_loader.get("toggle_ticked")
        self.unticked_img = self.game.img_loader.get("toggle_unticked")
        self.click_snd = self.game.snd_loader.get("click.mp3")

        # init rect and render
        self.rect = pg.rect.Rect(start_rect)
        self.rescale()

    def update(self, dt, events):
        for event in events:
            if self.rect.collidepoint(pg.mouse.get_pos()):
                if event.type == pg.MOUSEBUTTONDOWN and event.button == 1:
                    self.ticked = not self.ticked
                    self.click_snd.play()
                    self.rescale()

    def rescale(self):
        # if ticked, render the ticked image
        if self.ticked:
            self.image = pg.transform.scale(self.ticked_img, self.rect.size)
```

```
# if not ticked, render unticked image
else:
    self.image = pg.transform.scale(self.unticked_img, self.rect.size)

class Slider(pg.sprite.Sprite):
    def __init__(self, game, start_rect):
        super().__init__()
        self.game = game

        # initialise val to 0
        self.val = 1
        self.grabbed = False

        # load assets
        self.slider_img = self.game.img_loader.get("slider.png")
        self.thumb_img = self.game.img_loader.get("slider thumb.png")

        self.grab_snd = self.game.snd_loader.get("click.mp3")

        # initialise rect and render
        self.rect = pg.rect.Rect(start_rect)
        self.rescale()

    def update(self, dt, events):
        mouse_pos = vec2(pg.mouse.get_pos())

        for event in events:
            # if mouse is clicked and hovering over thumb rect, grab
            if event.type == pg.MOUSEBUTTONDOWN and event.button == 1:
                if self.scrn_thumb_rect.collidepoint(mouse_pos):
                    self.grabbed = True
            # if mouse is unclicked, release
            elif event.type == pg.MOUSEBUTTONUP and event.button == 1:
                if self.grabbed:
                    self.grab_snd.play()
                    self.grabbed = False

        # update val based of it is grabbed and mouse movement
        if self.grabbed:
            mouse_disp = pg.mouse.get_pos()[0] - self.rect.left
            self.val = (mouse_disp - (self.thumb_width/2)) / \
                      (self.rect.width - self.thumb_width)
            self.val = min(max(0, self.val), 1)
```

```
    self.rescale()

def rescale(self):
    # blit slider image
    self.image = pg.surface.Surface(self.rect.size, flags = pg.SRCALPHA)
    self.image.blit(pg.transform.scale(self.slider_img, self.rect.size),
                  (0,0))

    thumb_img = pg.transform.scale(self.thumb_img, [self.rect.height] * 2)
    self.thumb_width = thumb_img.get_width()

    # how many pixels over is the thumb displaced:
    thumb_disp = self.val * (self.rect.width - self.thumb_width)
    thumb_disp += self.thumb_width/2

    # find thumb collide rect
    self.scrn_thumb_rect = thumb_img.get_rect()
    self.scrn_thumb_rect.center = (self.rect.left + thumb_disp,
                                   self.rect.centery)

    # blit thumb to image
    thumb_rect = thumb_img.get_rect()
    thumb_rect.center = (thumb_disp, self.rect.height / 2)
    self.image.blit(thumb_img, thumb_rect)

class Spinner(pg.sprite.Sprite):
    def __init__(self, game, start_rect, options):
        super().__init__()
        self.game = game
        self.options = options
        self.index = 0

        # get font name
        font_name = self.game.config.text_font_name
        font_path = Path(self.game.config.img_pathX) / font_name

        # load font
        self.font = pg.font.Font(font_path.as_posix(), 20)
        self.text_colour = self.game.config.text_colour

        # load images and sounds
        self.arrows_img = self.game.img_loader.get("spinner arrows")
        self.end_hit_sound = self.game.snd_loader.get("spinner end.mp3")
```

```
        self.click_snd = self.game.snd_loader.get("click.mp3")

        self.rect = start_rect
        self.rescale()

def update(self, dt, events):
    mouse_pos = pg.mouse.get_pos()
    for event in events:
        if event.type == pg.MOUSEBUTTONDOWN and event.button == 1:
            # if mouse is clicked over left button, decrease index
            if self.left_button_rect.collidepoint(mouse_pos):
                if (self.index == 0):
                    self.end_hit_sound.play()
                else:
                    self.index -= 1
                    self.click_snd.play()
                    self.rescale()
            # if mouse is clicked over right button, increase index
            elif self.right_button_rect.collidepoint(mouse_pos):
                if (self.index == len(self.options)-1):
                    self.end_hit_sound.play()
                else:
                    self.index += 1
                    self.click_snd.play()
                    self.rescale()

def rescale(self):
    # blit image
    self.image = pg.transform.scale(self.arrows_img, self.rect.size)

    text_img = self.font.render(self.options[self.index],
                                False, self.text_colour)
    # rescale text to fit screen
    scale_factor = self.rect.height / text_img.get_height()
    text_size = [text_img.get_width() * scale_factor, self.rect.height]
    text_img = pg.transform.scale(text_img, text_size)
    text_img_rect = text_img.get_rect()

    # button rects
    self.left_button_rect = pg.Rect(
        self.rect.left,
        self.rect.top,
        self.rect.width * 1/6,
        self.rect.height )
```

```
        self.right_button_rect = pg.rect.Rect(
            self.rect.left+self.rect.width * 5/6,
            self.rect.top,
            self.rect.width * 1/6,
            self.rect.height     )

        # blit text
        text_img_rect.center = vec2(self.rect.size) / 2
        self.image.blit(text_img, text_img_rect)

class Button(pg.sprite.Sprite):
    def __init__(self, game, start_rect, text):
        super().__init__()
        self.game = game
        self.text = text

        # get font name
        font_name = self.game.config.text_font_name
        font_path = Path(self.game.config.img_pathX) / font_name

        # load font
        self.font = pg.font.Font(font_path.as_posix(), 20)
        self.text_colour = self.game.config.text_colour

        self.click_snd = self.game.snd_loader.get("click.mp3")

        self.pressed = [False] * 3
        self.rising_edges = [False] * 3
        self.falling_edges = [False] * 3

        self.rect = start_rect
        self.rescale()

    def update(self, dt, events):
        self.rising_edges = [False] * 3
        self.falling_edges = [False] * 3

        for event in events:
            # detect rising edges
            if self.rect.collidepoint(pg.mouse.get_pos()):
                if event.type == pg.MOUSEBUTTONDOWN and event.button <= 3:
                    self.rising_edges[event.button-1] = True
                    self.pressed[event.button-1] = True
```

```
        self.click_snd.play()

        self.rescale()

# detect falling edges
if event.type == pg.MOUSEBUTTONUP and event.button <= 3:
    if self.rect.collidepoint(pg.mouse.get_pos()) and \
       self.pressed[event.button-1]:
        self.falling_edges[event.button-1] = True
    self.pressed[event.button-1] = False

    self.rescale()

def rescale(self):
    # rescale image to rect
    self.image = pg.surface.Surface(self.rect.size)
    self.image.fill((32,32,32))
    if not max(self.pressed):
        pg.draw.rect(self.image, (255,255,255), (0,0,self.rect.width-3,
                                                    self.rect.height-3), 3)
    else:
        pg.draw.rect(self.image, (255,255,255), (3,3,self.rect.width-3,
                                                    self.rect.height-3), 3)

    # render text
    text_img = self.font.render(self.text, False, self.text_colour)
    # rescale text to fit screen
    scale_factor = self.rect.height / text_img.get_height()
    text_size = [text_img.get_width() * scale_factor, self.rect.height]
    text_img = pg.transform.scale(text_img, text_size)
    text_img_rect = text_img.get_rect()

    # blit text
    text_img_rect.center = vec2(self.rect.size) / 2
    self.image.blit(text_img, text_img_rect)
```

MENU_SYSTEM.PY

```
import pygame as pg
import Menu_Sprites as MS
import Sprites as sprites
import Maze_Gen as mg
import random as rng
```

```
vec2 = pg.math.Vector2
default_rect = lambda : pg.rect.Rect(0,0,1,1)

class Ui_Screen():
    def __init__(self, game):
        self.game = game

        self.elements = pg.sprite.Group()
        self.background = False

        self.screen = self.game.screen

    def tick(self, event_list, dt):
        self.elements.update(dt, event_list)
        if self.background:
            self.game.screen.blit(self.bg_img, self.bg_rect)
        self.elements.draw(self.game.screen)

    def rescale(self):
        screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
        screen_size = vec2(screen_rect.size)

        # rescale the background if it is present
        if self.background:
            # choose scale factor to fill screen
            match_width_SF = screen_rect.width / self.background.get_width()
            match_height_SF = screen_rect.height / self.background.get_height()
            scale_factor = max(match_height_SF, match_width_SF)

            # rescale image
            background_size = vec2(self.background.get_size()) * scale_factor
            self.bg_img = pg.transform.scale(self.background, background_size)

            # position image
            self.bg_rect = self.bg_img.get_rect()
            self.bg_rect.center = screen_rect.center

        # rescale all elements
        for elements in self.elements:
            elements.rescale()

class Main(Ui_Screen):
    def __init__(self, game):
```

```
super().__init__(game)

# init elements
self.title_text = MS.Text(game, default_rect(),
                           "Colour Between The Lines")
self.elements.add(self.title_text)
self.start_b = MS.Button(game, default_rect(), "Start")
self.elements.add(self.start_b)
self.options_b = MS.Button(game, default_rect(), "Options")
self.elements.add(self.options_b)
self.scoreboard_b = MS.Button(game, default_rect(), "Scoreboard")
self.elements.add(self.scoreboard_b)
self.close_b = MS.Button(game, default_rect(), "Close")
self.elements.add(self.close_b)
# load background image
self.background = game.img_loader.get("main background")

# call rescale to render image
self.rescale()

def tick(self, event_list, dt):
    # call parent tick function
    super().tick(event_list, dt)
    # push tick functions onto GSS for button presses

    # start
    if self.start_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.start_screen.tick)
    # options
    elif self.options_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.options_screen.tick)
    # scoreboard
    elif self.scoreboard_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.scoreboard_screen.tick)
    # close
    elif self.close_b.falling_edges[0]:
        self.game.game_state_stack = []

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.title_text.rect.width = screen_size.x
```

```
        self.title_text.rect.height = screen_size.y / 10
        self.title_text.rect.centerx = screen_rect.centerx
        self.title_text.rect.centery = screen_size.y / 8

        # position buttons
        button_width = screen_size.x / 3
        button_height = screen_size.y / 14
        button_spacing = screen_size.y / 12
        button_pos_y = screen_size.y * 2 / 6

        # start button
        self.start_b.rect.width = button_width
        self.start_b.rect.height = button_height
        self.start_b.rect.centerx = screen_rect.centerx
        self.start_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # options button
        self.options_b.rect.width = button_width
        self.options_b.rect.height = button_height
        self.options_b.rect.centerx = screen_rect.centerx
        self.options_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # scoreboard button
        self.scoreboard_b.rect.width = button_width
        self.scoreboard_b.rect.height = button_height
        self.scoreboard_b.rect.centerx = screen_rect.centerx
        self.scoreboard_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # close button
        self.close_b.rect.width = button_width
        self.close_b.rect.height = button_height
        self.close_b.rect.centerx = screen_rect.centerx
        self.close_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # call parent rescale method
        super().rescale()

class Pause(Ui_Screen):
    def __init__(self, game):
```

```
super().__init__(game)

# init elements
self.pause_text = MS.Text(game, default_rect(), "Pause")
self.elements.add(self.pause_text)
self.resume_b = MS.Button(game, default_rect(), "Resume")
self.elements.add(self.resume_b)
self.options_b = MS.Button(game, default_rect(), "Options")
self.elements.add(self.options_b)
self.exit_b = MS.Button(game, default_rect(), "Main Menu")
self.elements.add(self.exit_b)

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # resume button
    if self.resume_b.falling_edges[0]:
        self.game.game_state_stack.pop(-1)
    # ESC key
    for event in events:
        if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
            self.game.game_state_stack.pop(-1)
            break

    # options button
    if self.options_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.options_screen.tick)

    # exit button
    elif self.exit_b.falling_edges[0]:
        self.game.game_state_stack = [self.game.main_screen.tick]

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
```

```
        self.pause_text.rect.width = screen_size.x
        self.pause_text.rect.height = screen_size.y / 10
        self.pause_text.rect.centerx = screen_rect.centerx
        self.pause_text.rect.centery = screen_size.y / 8

        # position buttons
        button_width = screen_size.x / 3
        button_height = screen_size.y / 14
        button_spacing = screen_size.y / 12
        button_pos_y = screen_size.y / 4

        # resume button
        self.resume_b.rect.width = button_width
        self.resume_b.rect.height = button_height
        self.resume_b.rect.centerx = screen_rect.centerx
        self.resume_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # options button
        self.options_b.rect.width = button_width
        self.options_b.rect.height = button_height
        self.options_b.rect.centerx = screen_rect.centerx
        self.options_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = button_width
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # call parent rescale method
        super().rescale()

class Options(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.options_text = MS.Text(game, default_rect(), "Options")
        self.elements.add(self.options_text)
        self.gfx_b = MS.Button(game, default_rect(), "Graphics")
        self.elements.add(self.gfx_b)
```

```
self.snd_b = MS.Button(game, default_rect(), "Sound")
self.elements.add(self.snd_b)
self.exit_b = MS.Button(game, default_rect(), "Exit")
self.elements.add(self.exit_b)

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick function
    super().tick(events, dt)

    # gfx button
    if self.gfx_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.gfx_options_screen.tick)

    # snd button
    if self.snd_b.falling_edges[0]:
        self.game.game_state_stack.append(self.game.snd_options_screen.tick)

    # exit button
    if self.exit_b.falling_edges[0]:
        self.game.game_state_stack.pop(-1)
    # ESC key
    for event in events:
        if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
            self.game.game_state_stack.pop(-1)
            break

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.options_text.rect.width = screen_size.x
    self.options_text.rect.height = screen_size.y / 10
    self.options_text.rect.centerx = screen_rect.centerx
    self.options_text.rect.centery = screen_size.y / 8

    # position buttons
    button_width = screen_size.x / 3
```

```
button_height = screen_size.y / 14
button_spacing = screen_size.y / 12
button_pos_y = screen_size.y / 4

# gfx button
self.gfx_b.rect.width = button_width
self.gfx_b.rect.height = button_height
self.gfx_b.rect.centerx = screen_rect.centerx
self.gfx_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# snd button
self.snd_b.rect.width = button_width
self.snd_b.rect.height = button_height
self.snd_b.rect.centerx = screen_rect.centerx
self.snd_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# exit button
self.exit_b.rect.width = button_width
self.exit_b.rect.height = button_height
self.exit_b.rect.centerx = screen_rect.centerx
self.exit_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# call parent rescale method
super().rescale()

class GFX_Options(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.gfx_text = MS.Text(game, default_rect(), "Graphics")
        self.elements.add(self.gfx_text)
        self.res_sp = MS.Spinner(game, default_rect(), [
            "640x480",
            "800x600",
            "1280x720",
            "1366x768",
            "1600x900",
            "1920x1080",
            "Rescalable"])
        self.elements.add(self.res_sp)
```

```
if game.config.rescaleable:
    res_n = "Rescalable"
else:
    res_n = f"{game.config.resolution[0]}x{game.config.resolution[1]}"
self.res_sp.index = self.res_sp.options.index(res_n)

self.fullscreen_text = MS.Text(game, default_rect(), "Fullscreen")
self.elements.add(self.fullscreen_text)
self.Vsync_text = MS.Text(game, default_rect(), "Vsync")
self.elements.add(self.Vsync_text)

self.fullscreen_tg = MS.Toggle(game, default_rect())
self.elements.add(self.fullscreen_tg)
self.fullscreen_tg.ticked = game.config.fullscreen

self.Vsync_tg = MS.Toggle(game, default_rect())
self.elements.add(self.Vsync_tg)
self.Vsync_tg.ticked = game.config.vsync

self.apply_b = MS.Button(game, default_rect(), "Apply")
self.elements.add(self.apply_b)
self.exit_b = MS.Button(game, default_rect(), "Exit")
self.elements.add(self.exit_b)

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # change settings if apply pressed
    if self.apply_b.falling_edges[0]:
        # resolution
        res_option = self.res_sp.options[self.res_sp.index]
        if res_option == "Rescalable":
            self.game.config.rescaleable = True
        else:
            self.game.config.rescaleable = False
            self.game.config.resolution = [int(i) for i in
                                           res_option.split("x")]
```

```
# fullscreen
self.game.config.fullscreen = self.fullscreen_tg.ticked

# vsync
self.game.config.vsync = self.Vsync_tg.ticked

# save changes
self.game.config.save()

self.game.rescale()

# exit button
if self.exit_b.falling_edges[0]:
    self.game.game_state_stack.pop(-1)
# ESC key
for event in events:
    if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
        self.game.game_state_stack.pop(-1)
        break

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.gfx_text.rect.width = screen_size.x
    self.gfx_text.rect.height = screen_size.y / 10
    self.gfx_text.rect.centerx = screen_rect.centerx
    self.gfx_text.rect.centery = screen_size.y / 8

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 14
    button_spacing = screen_size.y / 12
    button_pos_y = screen_size.y / 4

    # res spinner
    self.res_sp.rect.width = button_width * 5/4
    self.res_sp.rect.height = button_height
    self.res_sp.rect.centerx = screen_rect.centerx
    self.res_sp.rect.centery = button_pos_y
    button_pos_y += button_spacing
```

```
# fullscreen text
self.fullscreen_text.rect.width = button_width
self.fullscreen_text.rect.height = button_height
self.fullscreen_text.rect.centerx = screen_rect.centerx-button_width*1/6
self.fullscreen_text.rect.centery = button_pos_y

# fullscreen toggle
self.fullscreen_tg.rect.width = button_height * 2/3
self.fullscreen_tg.rect.height = button_height * 2/3
self.fullscreen_tg.rect.centerx = screen_rect.centerx+button_width*1/2
self.fullscreen_tg.rect.centery = button_pos_y
button_pos_y += button_spacing

# Vsync text
self.Vsync_text.rect.width = button_width
self.Vsync_text.rect.height = button_height
self.Vsync_text.rect.centerx = screen_rect.centerx-button_width*1/6
self.Vsync_text.rect.centery = button_pos_y

# Vsync toggle
self.Vsync_tg.rect.width = button_height * 2/3
self.Vsync_tg.rect.height = button_height * 2/3
self.Vsync_tg.rect.centerx = screen_rect.centerx+button_width*1/2
self.Vsync_tg.rect.centery = button_pos_y
button_pos_y += button_spacing

# apply button
self.apply_b.rect.width = button_width
self.apply_b.rect.height = button_height
self.apply_b.rect.centerx = screen_rect.centerx
self.apply_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# exit button
self.exit_b.rect.width = button_width
self.exit_b.rect.height = button_height
self.exit_b.rect.centerx = screen_rect.centery
self.exit_b.rect.centery = button_pos_y
button_pos_y += button_spacing

# call parent rescale method
super().rescale()

class SND_Options(Ui_Screen):
```

```
def __init__(self, game):
    super().__init__(game)

    # init elements
    self.snd_text = MS.Text(game, default_rect(), "Sound")
    self.elements.add(self.snd_text)
    self.game_text = MS.Text(game, default_rect(), "Game Volume:")
    self.elements.add(self.game_text)
    self.music_text = MS.Text(game, default_rect(), "Music Volume:")
    self.elements.add(self.music_text)

    self.game_slider = MS.Slider(game, default_rect())
    self.elements.add(self.game_slider)
    self.game_slider.val = game.config.game_vol

    self.music_slider = MS.Slider(game, default_rect())
    self.elements.add(self.music_slider)
    self.music_slider.val = game.config.music_vol

    self.exit_b = MS.Button(game, default_rect(), "Exit")
    self.elements.add(self.exit_b)

    # load background
    self.background = game.img_loader.get("menu background")

    # call rescale to render image
    self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # store sound vols to config
    self.game.config.game_vol = self.game_slider.val
    self.game.config.music_vol = self.music_slider.val

    # exit button
    if self.exit_b.falling_edges[0]:
        self.game.config.save()
        self.game.game_state_stack.pop(-1)
    # ESC key
    for event in events:
        if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
            self.game.config.save()
```

```
        self.game.game_state_stack.pop(-1)
        break

# call game's load sound function
self.game.load_snd_vol()

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.snd_text.rect.width = screen_size.x
    self.snd_text.rect.height = screen_size.y / 10
    self.snd_text.rect.centerx = screen_rect.centerx
    self.snd_text.rect.centery = screen_size.y / 8

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 14
    button_spacing = screen_size.y / 12
    button_pos_y = screen_size.y / 4

    # game volume text
    self.game_text.rect.width = button_width
    self.game_text.rect.height = button_height * 3/4
    self.game_text.rect.centerx = screen_rect.centerx
    self.game_text.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # game volume slider
    self.game_slider.rect.width = button_width
    self.game_slider.rect.height = button_height / 2
    self.game_slider.rect.centerx = screen_rect.centerx
    self.game_slider.rect.centery = button_pos_y
    button_pos_y += button_spacing

    # music volume text
    self.music_text.rect.width = button_width
    self.music_text.rect.height = button_height * 3/4
    self.music_text.rect.centerx = screen_rect.centerx
    self.music_text.rect.centery = button_pos_y
    button_pos_y += button_spacing

# music volume slider
```

```
        self.music_slider.rect.width = button_width
        self.music_slider.rect.height = button_height / 2
        self.music_slider.rect.centerx = screen_rect.centerx
        self.music_slider.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = button_width
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # call parent rescale method
        super().rescale()

class Scoreboard(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.sb_text = MS.Text(game, default_rect(), "Scoreboard")
        self.elements.add(self.sb_text)
        self.exit_b = MS.Button(game, default_rect(), "Exit")
        self.elements.add(self.exit_b)

        # score boxes
        self.score_boxes = []
        for _ in range(13):
            box = MS.Text(game, default_rect(), "")
            self.score_boxes.append(box)
            self.elements.add(box)

        # load background
        self.background = game.img_loader.get("menu background")

        # call rescale to render image
        self.load()
        self.rescale()

    def tick(self, events, dt):
        # call parent tick method
        super().tick(events, dt)
```

```
# exit button
if self.exit_b.falling_edges[0]:
    self.game.game_state_stack.pop(-1)
# ESC key
for event in events:
    if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
        self.game.game_state_stack.pop(-1)
        break

def rescale(self):
    screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.sb_text.rect.width = screen_size.x
    self.sb_text.rect.height = screen_size.y / 10
    self.sb_text.rect.centerx = screen_rect.centerx
    self.sb_text.rect.centery = screen_size.y / 16

    # render top 10 scores
    table_width = screen_size.x
    table_row_height = screen_size.y * 1/6
    table_entry_height = screen_size.y * 1/20

    # table heading
    box = self.score_boxes[0]
    box.rect.width = table_width
    box.rect.height = table_entry_height
    box.rect.centerx = screen_rect.centerx
    box.rect.centery = table_row_height
    table_row_height += table_entry_height
    box.text = \
f"# |{'Name':^15}|{'time':^6}|{'width':^6}|{'height':^6}|{'seed':^8}""

    # sort by shortest time
    sorted_scores = sorted(self.scoreboard_data, key = lambda x : int(x[1]))

    # set data for each box
    for i in range(1,13):
        # set row position
        box = self.score_boxes[i]
        box.rect.width = table_width
```

```
        box.rect.height = table_entry_height
        box.rect.centerx = screen_rect.centerx
        box.rect.centery = table_row_height
        table_row_height += table_entry_height

        # set row text
        if len(sorted_scores) > i-1:
            r = sorted_scores[i-1]
            box.text = \
                f"{{i:^2}}|{{r[0]:^15}}|{{r[1]:^6}}|{{r[2]:^6}}|{{r[3]:^6}}|{{r[4]:^8}}"
        else:
            box.text = f"{{i:^2}}|{{'':^15}}|{{'':^6}}|{{'':^6}}|{{'':^6}}|{{'':^8}}"

        # exit button
        self.exit_b.rect.width = screen_size.x / 3
        self.exit_b.rect.height = screen_size.y / 14
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = screen_size.y * 7/8

        # call parent rescale method
        super().rescale()

    def load(self):

        try:
            # load scoreboard file
            scores_file = open(self.game.config.scoreboard_pathX, "r")

            # split by lines
            self.scoreboard_data = []
            for line in scores_file.readlines():
                self.scoreboard_data.append(line.strip().split(","))

            # close scoreboard file
            scores_file.close()
        except:
            print(f"failed to read scoreboard file")
            self.scoreboard_data = []

    def save(self):
        # convert data to a string
        output_str = ""
        for row in self.scoreboard_data:
            for column in row:
```

```
        output_str += f"{column},"
    output_str = output_str[:-1]
    output_str += "\n"

try:
    # load scoreboard file
    scores_file = open(self.game.config.scoreboard_pathX, "w")

    # write to file
    scores_file.write(output_str)

    # close file
    scores_file.close()
except Exception as error:
    print(f"failed to write scoreboard file: {error}")

def add_score(self, name, time, width, height, seed):
    # load data before appending to it
    self.load()

    # add new data
    self.scoreboard_data.append([name, time, width, height, seed])

    # save new data and apply changes to screen
    self.save()
    self.rescale()

class Start(Ui_Screen):
    def __init__(self, game):
        super().__init__(game)

        # init elements
        self.start_text = MS.Text(game, default_rect(), "Start Level")
        self.elements.add(self.start_text)
        self.label_text = MS.Text(game, default_rect(), "Maze Size:")
        self.elements.add(self.label_text)
        self.width_text = MS.Text(game, default_rect(), "Width:")
        self.elements.add(self.width_text)
        self.height_text = MS.Text(game, default_rect(), "Height:")
        self.elements.add(self.height_text)
        self.cross_text = MS.Text(game, default_rect(), "X")
        self.elements.add(self.cross_text)
        self.seed_text = MS.Text(game, default_rect(), "Seed:")
```

```
self.elements.add(self.seed_text)
self.width_i = MS.Input_Box(game, default_rect(), "20", MS.k2c_numeric)
self.elements.add(self.width_i)
self.height_i = MS.Input_Box(game, default_rect(), "10", MS.k2c_numeric)
self.elements.add(self.height_i)
self.seed_i = MS.Input_Box(game, default_rect(), "0", MS.k2c_numeric)
self.elements.add(self.seed_i)
self.start_b = MS.Button(game, default_rect(), "Start Level")
self.elements.add(self.start_b)
self.exit_b = MS.Button(game, default_rect(), "Cancel")
self.elements.add(self.exit_b)

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # start level button
    if self.start_b.falling_edges[0]:
        # get width
        if len(self.width_i.text) > 0:
            width = int(self.width_i.text)
        else:
            width = int(self.width_i.default_text)

        # width validation
        if not 10 <= width <= 50:
            self.game.snd_loader.get("spinner end").play()
            self.width_i.text = ""
            self.width_i.rescale()
            return

        # get height
        if len(self.height_i.text) > 0:
            height = int(self.height_i.text)
        else:
            height = int(self.height_i.default_text)

        # height validation
```

```
        if not 10 <= height <= 50:
            self.game.snd_loader.get("spinner end").play()
            self.height_i.text = ""
            self.height_i.rescale()
            return

        # get seed
        if len(self.seed_i.text) > 0:
            seed = int(self.seed_i.text)
        else:
            seed = int(self.seed_i.default_text)

        # start level
        self.game.start_level((width, height), seed)

        # exit button
        if self.exit_b.falling_edges[0]:
            self.game.game_state_stack.pop(-1)
        # ESC key
        for event in events:
            if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
                self.game.game_state_stack.pop(-1)
                break

    def rescale(self):
        screen_rect = pg.rect.Rect(0, 0, *self.screen.get_size())
        screen_size = vec2(screen_rect.size)

        # positon title text
        self.start_text.rect.width = screen_size.x
        self.start_text.rect.height = screen_size.y / 10
        self.start_text.rect.centerx = screen_rect.centerx
        self.start_text.rect.centery = screen_size.y / 8

        # position buttons
        button_width = screen_size.x / 3
        button_height = screen_size.y / 14
        button_spacing = screen_size.y / 12
        button_pos_y = screen_size.y * 1 / 4

        # maze size text
        self.label_text.rect.width = button_width
        self.label_text.rect.height = button_height
        self.label_text.rect.centerx = screen_rect.centerx
```

```
        self.label_text.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # width and height text
        width_x = screen_rect.centerx - screen_size.x / 6
        height_x = screen_rect.centerx + screen_size.x / 6
        dim_width = button_width * 3/4

        self.width_text.rect.width = button_width
        self.width_text.rect.height = button_height
        self.width_text.rect.centerx = width_x
        self.width_text.rect.centery = button_pos_y

        self.height_text.rect.width = button_width
        self.height_text.rect.height = button_height
        self.height_text.rect.centerx = height_x
        self.height_text.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # width and height inputs
        self.width_i.rect.width = dim_width
        self.width_i.rect.height = button_height
        self.width_i.rect.centerx = width_x
        self.width_i.rect.centery = button_pos_y

        self.height_i.rect.width = dim_width
        self.height_i.rect.height = button_height
        self.height_i.rect.centerx = height_x
        self.height_i.rect.centery = button_pos_y

        # X
        self.cross_text.rect.width = screen_size.x / 20
        self.cross_text.rect.height = button_height
        self.cross_text.rect.centerx = screen_rect.centerx
        self.cross_text.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # seed input
        self.seed_text.rect.width = button_width * 1/2
        self.seed_text.rect.height = button_height
        self.seed_text.rect.centerx = screen_rect.centerx - button_width * 3/8
        self.seed_text.rect.centery = button_pos_y

        self.seed_i.rect.width = button_width * 3/4
```

```
        self.seed_i.rect.height = button_height
        self.seed_i.rect.centerx = screen_rect.centerx + button_width * 1/4
        self.seed_i.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # randomise seed
        rng.seed()
        self.seed_i.default_text = str(rng.randint(0,999999))

        # start button
        self.start_b.rect.width = button_width
        self.start_b.rect.height = button_height
        self.start_b.rect.centerx = screen_rect.centerx
        self.start_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = button_width
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # call parent rescale method
        super().rescale()

class End(Ui_Screen):
    def __init__(self, game, time, width, height, seed):
        super().__init__(game)
        self.score_added = False
        self.time = time
        self.width = width
        self.height = height
        self.seed = seed

        # init elements
        self.end_text = MS.Text(game, default_rect(), "Level Complete!")
        self.elements.add(self.end_text)
        self.dim_text = MS.Text(game, default_rect(), f"{width:^8}X{height:^8}")
        self.elements.add(self.dim_text)
        self.seed_text = MS.Text(game, default_rect(), f"Seed:{seed:^8}")
        self.elements.add(self.seed_text)
        self.Time_text = MS.Text(game, default_rect(), f"Time:{time:^8}")
```

```
self.elements.add(self.Time_text)

self.sb1_text = MS.Text(game, default_rect(),
                      "Put Your Name on")
self.elements.add(self.sb1_text)
self.sb2_text = MS.Text(game, default_rect(),
                      "The Scoreboard!:")
self.elements.add(self.sb2_text)

self.name_i = MS.Input_Box(game, default_rect(), "Enter Name",
                           MS.k2c_numeric +
                           MS.k2c_alpha_lower +
                           MS.k2c_alpha_upper +
                           [" "])
self.elements.add(self.name_i)

self.add_b = MS.Button(game, default_rect(), "Add To Scoreboard")
self.elements.add(self.add_b)

self.exit_b = MS.Button(game, default_rect(), "Return to Main Menu")
self.elements.add(self.exit_b)

# load background
self.background = game.img_loader.get("menu background")

# call rescale to render image
self.rescale()

def tick(self, events, dt):
    # call parent tick method
    super().tick(events, dt)

    # add to scoreboard
    if not self.score_added and self.add_b.falling_edges[0]:
        if len(self.name_i.text) > 0:
            self.score_added = True
            self.game.scoreboard_screen.add_score(self.name_i.text,
                                                   self.time,
                                                   self.width,
                                                   self.height,
                                                   self.seed)
            self.add_b.text_colour = (128,128,128)
            self.add_b.rescale()
```

```
# exit button
if self.exit_b.falling_edges[0]:
    self.game.game_state_stack = [self.game.main_screen.tick]
# ESC key
for event in events:
    if event.type == pg.KEYUP and event.key == pg.K_ESCAPE:
        self.game.game_state_stack = [self.game.main_screen.tick]
        break

def rescale(self):
    screen_rect = pg.Rect(0, 0, *self.screen.get_size())
    screen_size = vec2(screen_rect.size)

    # positon title text
    self.end_text.rect.width = screen_size.x
    self.end_text.rect.height = screen_size.y / 10
    self.end_text.rect.centerx = screen_rect.centerx
    self.end_text.rect.centery = screen_size.y / 8

    # position buttons
    button_width = screen_size.x / 3
    button_height = screen_size.y / 14
    button_spacing = screen_size.y / 6
    button_pos_y = screen_size.y * 1 / 3

    left_column = screen_size.x * 1/4
    right_column = screen_size.x * 3/4

    # maze size text
    self.dim_text.rect.width = screen_size.x * 1/2
    self.dim_text.rect.height = button_height
    self.dim_text.rect.centerx = left_column
    self.dim_text.rect.centery = button_pos_y

    # add to scoreboard text
    self.sb1_text.rect.width = screen_size.x * 1/2
    self.sb1_text.rect.height = button_height * 2/3
    self.sb1_text.rect.centerx = right_column
    self.sb1_text.rect.centery = button_pos_y

    self.sb2_text.rect.width = screen_size.x * 1/2
    self.sb2_text.rect.height = button_height * 2/3
    self.sb2_text.rect.centerx = right_column
```

```
        self.sb2_text.rect.centery = button_pos_y + button_spacing / 3
        button_pos_y += button_spacing

        # seed text
        self.seed_text.rect.width = screen_size.x * 1/2
        self.seed_text.rect.height = button_height
        self.seed_text.rect.centerx = left_column
        self.seed_text.rect.centery = button_pos_y

        # name input
        self.name_i.rect.width = screen_size.x * 1/2
        self.name_i.rect.height = button_height
        self.name_i.rect.centerx = right_column
        self.name_i.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # time text
        self.Time_text.rect.width = screen_size.x * 1/2
        self.Time_text.rect.height = button_height
        self.Time_text.rect.centerx = left_column
        self.Time_text.rect.centery = button_pos_y

        # add name button
        self.add_b.rect.width = screen_size.x * 1/2
        self.add_b.rect.height = button_height
        self.add_b.rect.centerx = right_column
        self.add_b.rect.centery = button_pos_y
        button_pos_y += button_spacing

        # exit button
        self.exit_b.rect.width = screen_size.x * 2/3
        self.exit_b.rect.height = button_height
        self.exit_b.rect.centerx = screen_rect.centerx
        self.exit_b.rect.centery = button_pos_y

        # call parent rescale method
        super().rescale()

class Level(Ui_Screen):
    def __init__(self, game, size, seed):
        super().__init__(game)
        self.size = size
        self.seed = seed
```

```
self.timer = sprites.Timer(self.game)
self.win_snd = game.snd_loader.get("winfretless.ogg")

def setup(self):
    """sets up the level"""
    # initialise camera
    self.camera = sprites.Camera(self.game)

    # start setting up the maze
    self.maze = mg.Maze(self.game, self.size, self.seed)
    # finishes generating the maze and sprites
    self.maze.setup()
    # initialise sprites in maze
    for sprite in self.maze.all_sprites:
        sprite.render(0)

    # initialise player
    self.player = sprites.Player(self.game, (self.maze.start))
    self.maze.all_sprites.add(self.player)
    # set what the camera should follow
    self.camera.set_target(self.player)
    self.camera.pos = pg.Vector2(5,5)

    self.rescale()

def tick(self, events, dt):
    super().tick(events, dt)
    self.game.screen.fill((255,0,255))

    for event in events:
        if event.type == pg.KEYUP:
            if event.key == pg.K_ESCAPE:
                self.game.game_state_stack.append(
                    self.game.pause_screen.tick)
            # if event.key == pg.K_SPACE:
            #     self.game.end_level((20,10), 12345, 678)

            # zoom debug
            # if event.type == pg.MOUSEBUTTONDOWN:
            #     if event.button == 4:
            #         self.camera.zoom = min(self.camera.zoom+1, 20)
            #     if event.button == 5:
            #         self.camera.zoom = max(self.camera.zoom-1 , 1)
            #     self.rescale()
```

```
# update all sprites
self.maze.all_sprites.update(dt)
self.camera.update(dt)
self.timer.update(dt)

# detect win condition:
if (self.player.pos+vec2(0.25,0.75))//1 == self.maze.exit.pos:
    self.win_snd.play()
    self.game.end_level(self.size, self.seed, self.timer.total_time)

# call all sprites render method
for sprite in self.maze.all_sprites:
    sprite.render(dt)

self.game.screen.fill((32,32,32))
# draw floor in correct position
for y in range(0, self.maze.bsize[1]-1, 5):
    for x in range(0, self.maze.bsize[0]-1, 5):
        self.floor_rect.topleft = self.camera.wrld_2_scrn_coord((x,y))
        if self.floor_rect.colliderect(self.screen_rect):
            self.screen.blit(self.floor, self.floor_rect)

self.maze.all_sprites.draw(self.game.screen)

def rescale(self):
    self.screen_rect = pg.Rect(0, 0, *self.screen.get_size())

    # rescale background
    floor = self.game.img_loader.get("arcade_carpet_1_512")
    floor_size = self.camera.wrld_2_scrn_coord((5,5)) - \
                self.camera.wrld_2_scrn_coord((0,0))
    self.floor = pg.transform.scale(floor, floor_size)
    self.floor_rect = self.floor.get_rect()

super().rescale()
```