OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name : Alexander Mills

Candidate Number : <INSERT CANDIDATE NUMBER>

<Institution Name> : <INSERT CENTRE NUMBER>

Title of Project : <INSERT PROJECT TITLE>

H446-03 – Project CONTENTS

Table of Contents

[A. Analysis 6](#_Toc94684633)

[Outline 6](#_Toc94684634)

[Stakeholders 6](#_Toc94684635)

[Game Research: Tetris 7](#_Toc94684636)

[Game research: Hue 10](#_Toc94684637)

[Game Research: World’s Hardest Game 13](#_Toc94684638)

[Survey 14](#_Toc94684639)

[Planning 14](#_Toc94684640)

[Survey Response review 15](#_Toc94684641)

[Graphics: 15](#_Toc94684642)

[Sound: 16](#_Toc94684643)

[Level Design: 17](#_Toc94684644)

[Other Feedback: 18](#_Toc94684645)

[Proposed Feature List 18](#_Toc94684646)

[Limitations and Scope 19](#_Toc94684647)

[Why this Solution is Suited to a Computation Solution 20](#_Toc94684648)

[Abstraction 20](#_Toc94684649)

[Thinking Ahead 21](#_Toc94684650)

[Thinking Procedurally 21](#_Toc94684651)

[Thinking Logically 21](#_Toc94684652)

[Thinking Concurrently 22](#_Toc94684653)

[Hardware and Software Requirements 23](#_Toc94684654)

[Success Criteria – designed for usability 23](#_Toc94684655)

[Graphics - 1 23](#_Toc94684656)

[Player design - a 23](#_Toc94684657)

[Environment design - b 24](#_Toc94684658)

[ENemy design - c 24](#_Toc94684659)

[Objective design - d 25](#_Toc94684660)

[User interface - 2 25](#_Toc94684661)

[Main menu - a 25](#_Toc94684662)

[Pause menu - b 26](#_Toc94684663)

[Options menus - c 26](#_Toc94684664)

[GUI design - d 27](#_Toc94684665)

[Sound - 3 27](#_Toc94684666)

[Sprite sounds - a 27](#_Toc94684667)

[Level sounds - b 27](#_Toc94684668)

[Background sounds - c 28](#_Toc94684669)

[Level design - 4 28](#_Toc94684670)

[Maze layout - a 28](#_Toc94684671)

[Maze population - b 29](#_Toc94684672)

[Enemies - c 29](#_Toc94684673)

[Checkpoints – d 29](#_Toc94684674)

[Win criteria - e 30](#_Toc94684675)

[Game mechanics - 4 30](#_Toc94684676)

[Player controller - a 30](#_Toc94684677)

[Enemy controller - b 31](#_Toc94684678)

[B. Design 32](#_Toc94684679)

[User interface Design 32](#_Toc94684680)

[User Interface Type 32](#_Toc94684681)

[Layout 32](#_Toc94684682)

[Navigation 36](#_Toc94684683)

[Asset Selection 36](#_Toc94684684)

[Theming: 37](#_Toc94684685)

[Font: 37](#_Toc94684686)

[Tiles: 38](#_Toc94684687)

[Character sprites: 39](#_Toc94684688)

[Systems diagram 40](#_Toc94684689)

[Overall program layout 41](#_Toc94684690)

[Module Declarations 43](#_Toc94684691)

[Module: Config 43](#_Toc94684692)

[Config – Class: game\_config 43](#_Toc94684693)

[Config – testing 45](#_Toc94684694)

[Module: Asset\_loader 46](#_Toc94684695)

[Asset\_loader – Class: Img\_loader 46](#_Toc94684696)

[Asset\_Loader – Class: Snd\_Loader 47](#_Toc94684697)

[Asset\_loader – Class: Sprite\_Sheet 47](#_Toc94684698)

[Asset\_Loader – Testing and Verification 48](#_Toc94684699)

[Module: Main 49](#_Toc94684700)

[Main - Class: Game 49](#_Toc94684701)

[Main – testing 51](#_Toc94684702)

[Module: maze\_gen 52](#_Toc94684703)

[Maze\_GEN - Class: Maze 52](#_Toc94684704)

[Maze\_GEN - Algorithms 55](#_Toc94684705)

[Maze\_Gen – Testing: 58](#_Toc94684706)

[Module: Sprites 60](#_Toc94684707)

[Sprites – Class: Renderable\_Sprite 60](#_Toc94684708)

[sprites – Class: Player 60](#_Toc94684709)

[Sprites – Class: Wall 63](#_Toc94684710)

[Sprites – Class: Gateway 64](#_Toc94684711)

[Sprites – Class: Block 64](#_Toc94684712)

[Sprites – Class: Checkpoint 65](#_Toc94684713)

[Sprites – Class: Key 66](#_Toc94684714)

[Sprites – Class: Exit 66](#_Toc94684715)

[Sprites – Class: Camera 67](#_Toc94684716)

[Sprites – Class: Timer 67](#_Toc94684717)

[Sprites – Testing 67](#_Toc94684718)

[Module: Menu\_Sprites 71](#_Toc94684719)

[Menu\_Sprites – Class: Text 71](#_Toc94684720)

[Menu\_Sprites – Class: Input\_box 71](#_Toc94684721)

[Menu\_Sprites – Class: Toggle 72](#_Toc94684722)

[Menu\_Sprites – Class: Slider 73](#_Toc94684723)

[Menu\_Sprites – Class: Spinner 73](#_Toc94684724)

[Menu\_Sprites – Class: Button 74](#_Toc94684725)

[Menu\_Sprites – Testing 75](#_Toc94684726)

[Module: Menu\_System 77](#_Toc94684727)

[Menu\_System – Class: UI\_Screen 77](#_Toc94684728)

[Menu\_System – Class: Main 78](#_Toc94684729)

[Menu\_System – Class: Start 79](#_Toc94684730)

[Menu\_System – Class: End 80](#_Toc94684731)

[Menu\_System – Class: Scoreboard 80](#_Toc94684732)

[Menu\_System – Class: Pause 81](#_Toc94684733)

[Menu\_System – Class: Options 82](#_Toc94684734)

[Menu\_System – Class: GFX\_Options 83](#_Toc94684735)

[Menu\_System – Class: SND\_Options 84](#_Toc94684736)

[Menu\_System – Class: Level 84](#_Toc94684737)

[Menu\_system - Testing 85](#_Toc94684738)

[C. Developing the coded solution (“The development story”) 87](#_Toc94684739)

[D. Evaluation 87](#_Toc94684740)

[Project Appendixes 88](#_Toc94684741)

# 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 unstainable 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: Graphical user interface

Description automatically generated

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 an 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-crowed, so I will in my game I won’t be re-using UI elements to reduce clutter.

Gameplay:

A picture containing text, parking

Description automatically generated

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

Pause Menu:

Graphical user interface

Description automatically generated

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:

Graphical user interface, diagram, schematic

Description automatically generated

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:

A picture containing timeline

Description automatically generated

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:

Diagram, schematic

Description automatically generated

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 gamer 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:   * 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:   * no options * a slider for game volume, and a slider for music volume * all game sounds have individual siders | 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:   * 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:

Chart, bar chart

Description automatically generated

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.

Chart, bar chart

Description automatically generated

Chart, pie chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated

#### Sound:

Chart, bar chart

Description automatically generatedChart, bar chart

Description automatically generated

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

Chart, pie chart

Description automatically generatedGraphical user interface, text, application

Description automatically generated

#### Level Design:

Graphical user interface, text, application, email

Description automatically generated

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 that 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.

Chart, bar chart

Description automatically generated

Chart, pie chart

Description automatically generated

#### Other Feedback:

Chart, pie chart

Description automatically generated

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.

A picture containing background pattern

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Proposed Feature List

|  |  |
| --- | --- |
| Feature | Justification |
| Main menu which points to   * 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 3rd dimension would allow the puzzles to be much more intricate, with many more hidden features and more alternate solutions | Im not familiar enough with 3d alternatives to pygame such as Ursina engine, which I don’t have time to learn |
| There will be no narrative to the game | Narratives make games more enjoyable by telling an engaging, emotional story. | A well written and enjoyable story takes more time to come up with than I have for this project. |
| There won’t be multiple level themes | More level themes would give the game more character, making it mor 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 a way 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 |
| Peripherials:   * 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 player sliding across the ground | User base survey |
| 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 | * 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 some enemies will be useless, and will make the game slower unnecessarily | Proposed features: maze generation |
| 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.

Diagram

Description automatically generated with low confidenceThis 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.

**Graphical user interface

Description automatically generated**

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

**A picture containing logo

Description automatically generated**

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

**Graphical user interface, application

Description automatically generated**

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.

Table

Description automatically generated

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

**Graphical user interface

Description automatically generated**

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.

**Graphical user interface, text

Description automatically generated**

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.

**Graphical user interface, text, application

Description automatically generated**

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

Chart

Description automatically generatedThe 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 bi 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.

A picture containing graphical user interface

Description automatically generated

### Font:

Graphical user interface, text, application

Description automatically generatedThe 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](https://www.fontspace.com/pixeloid-font-f69232) by GGBotNet as it meets these criteria and has a free license so I can use it in the game.

### A screenshot of a computer Description automatically generated with low confidenceTiles:

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](https://opengameart.org/content/dawnblocker-ortho) 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.

Qr code

Description automatically generated

A picture containing text, scoreboard

Description automatically generatedChart, histogram

Description automatically generated

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](https://opengameart.org/content/arcade-carpet-textures) from Luckius meets these requirements perfectly and aligns with the 8-bit aesthetic.

### Character sprites:

Graphical user interface, application

Description automatically generatedA close up of a rug

Description automatically generated with low confidenceThe 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](https://opengameart.org/content/classic-hero-and-baddies-pack) and devurandom’s [RPG Character Sprites](https://opengameart.org/content/db16-rpg-character-sprites-v2). 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.

A screenshot of a computer

Description automatically generated with medium confidence

## Module Declarations

Module: Config

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

Config – Class: game\_config

* Purpose: stores all configuration data for the game, such as user controllable settings and game balancing data. As all gameplay critical values are stored here, it is easy to make large changes to the game’s dynamics and behavior in a single place
* Attributes:
  + img\_path – String
    - path of image assets folder
  + snd\_path – String
    - path of sound assets folder
  + music\_path – String
    - path of background music folder
  + scoreboard\_path – String
    - path of scoreboard csv file
  + resolution – array(int width, int height)
    - screen resolution
  + fullscreen – Boolean
  + vsync – Boolean
  + game\_vol – float
    - domain: 0 to 1 continuous
    - sets game volume
  + music\_vol – float
    - domain: 0 to 1 continuous
    - sets music volume
  + text\_colour – Tuple (int r int g in b)
    - each of r g and b are represent colour values
    - domain of r, g and b: 0 to 255
  + text\_font\_name – String
    - name of the font used for ui text
  + player\_hurt\_cooldown – int
    - domain: 0+
    - time in ms until between when the player can be hurt
  + player\_max\_health – int
    - domain: 0+
    - maximum number of heath points the player has
  + player\_max\_speed – int
    - domain: 0+
  + player\_acc – int
    - domain: 0+
    - how fast the player accelerates
  + enemy\_speed – int
    - domain: 0+
    - how fast the enemy moves
  + maze\_blocks\_start\_proportion – float
    - domain: 0 to 1 continuous
    - where along the path gateway population starts
  + maze\_blocks\_distance\_proportion – float
    - domain: 0 to 1 continuous
    - how far along the path each gateway is from the next
  + maze\_gateway\_jitter – int
    - domain: any integer
    - how much gateways randomly move back and forth from their planned position on the path
  + maze\_gateway\_skip\_threshold – float
    - domain: 0 to 1
    - how often a gateway isn’t placed immediately after it’s block, but is saved for further in the maze
  + maze\_branch\_stop\_threshold – float
    - domain: 0 to 1
    - probability that a branch keeps branching deeper into the maze
  + maze\_key\_count – int
    - domain: 0+
    - number of keys to be generated in the maze
  + maze\_checkpoint\_count – int
    - domain: 0+
    - number of checkpoints to be generated in the maze
  + maze\_enemy\_count – int
    - domain: 0+
    - number of enemies to be generated in the maze
* Methods
  + VOID \_\_init\_\_ ()
    - initialise
  + VOID save ()
    - Self-modifying code: this method replaces it’s class’s attribute definitions with ones which have the values of it’s attribute’s current values; this implements a persistent python file where a class’s attributes can be store values across multiple load cycles
    - Reads config.py from secondary storage
    - Replace all variable definitions with new ones in accordance with the current values stored in the attributes
    - Save config.py to secondary storage

Config – testing

**Test environment**:

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

**Test table**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Tested Functionality | Test conditions/ input | Test type | Expected behavior |
| 1 | Test loading string attributes | Host loads img\_path and prints it | Valid | Outputs the img path saved there |
| 2 | Test loading int attributes | Host loads player\_max\_health  And print it | Valid | Outputs the player\_max\_health value |
| 3 | Test loading bool attributes | Host loads fullscreen and prints it | Valid | Outputs the fullscreen value |
| 4 | Test loading float attributes | Host loads game\_vol and prints it | Valid | Outputs the game\_vol value |
| 5 | Test loading array attributes | Host loads resolution and prints it | Valid | Outputs the resolution stored there |
| 6 | Test storing string attributes | Host sets snd\_path to a new value and calls save() | Valid | Config.py file now contains new value at snd\_path definition |
| 7 | Test storing int attributes | Host sets player\_max\_speed to a new value and calls save() | Valid | Config.py file now contains new value at player\_max\_speed definition |
| 8 | Test storing bool attributes | Host sets vsync to a new value and calls save() | Valid | Config.py file now contains new value at vsync definition |
| 9 | Test storing float attributes | Host sets music\_vol to a new value and calls save() | Valid | Config.py file now contains new value at music\_vol definition |
| 10 | Test storing array attributes | Host sets text\_colour to a new value and calls save() | Valid | Config.py file now contains new value at text\_colour definition |
| 11 | Test reloading string attributes | Host reloads img\_path and prints it | Valid | Outputs the img path saved there |
| 12 | Test reloading int attributes | Host reloads player\_max\_health  And print it | Valid | Outputs the player\_max\_health value |
| 13 | Test reloading bool attributes | Host reloads fullscreen and prints it | Valid | Outputs the fullscreen value |
| 14 | Test reloading float attributes | Host reloads game\_vol and prints it | Valid | Outputs the game\_vol value |
| 15 | Test reloading array attributes | Host reloads resolution and prints it | Valid | Outputs the resolution stored there |

Module: Asset\_loader

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

Asset\_loader – Class: Img\_loader

* Purpose: loads, handles, and caches image assets for sprites to access.
* Attributes:
  + assets – dictionary (string img1\_name : Surface s1,string img2\_name : Surface s2)[img\_name]
    - Stores all images that have already been loaded in RAM so that they don’t need to be loaded from secondary storage every time that they are needed.
    - A dictionary is used as it is a form of hash table, meaning all images very quick to access, having equal retrieval times. This will reduce delays in the game as it waits for assets to be accessed.
  + Sprite\_sheets – list (Sprite\_sheet sheet\_1,Sprite\_sheet sheet\_2)
* Methods:
  + VOID \_\_init\_\_ ()
    - Initalises assets to an empty dictionary
    - Initalise 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\_\_ ()
    - Initalises 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) again and blits it to a different location on the screen | Valid | Img1.png appears on the screen again |
| 4 | Recovering from failing to load sprites | Host calls get(img2.png) and blits it to the screen | Invalid | There is a purple square placed on the screen and the game hasn’t crashed |
| 5 | music loading | Host calls get(sound1.wav) and plays it | Valid | The sound can be heard playing |
| 6 | Recovering from failing to load music | Host calls get(sound2.wav)  And plays it | invalid | No sound is heard and game doesn’t crash |

Module: Main

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

Main - Class: Game

* Purpose: hosts the main game loop and acts as a single data structure from which all data relating to the game is stored
* Attributes:
  + screen – display surface
  + game\_state\_stack (GSS) – stack (implemented as list)
    - stores which menu / screen the player has navigated to
    - top value is the current screen / menu the player sees
    - enables a single menu function to be used for accessing a menu from multiple places eg the options menu can be opened from the pause screen and the main game screen, and the stack will be used to keep track of which screen to return to
  + img\_loader – loaders.Img\_loader
    - is used to load all visual game assets
    - stores the visual assets so that they can be rendered multiple times while loading them only once
  + snd\_loader – loaders.Snd\_loader
    - is used to load all sound game assets
    - stores the sound assets so that they can be rendered multiple times while loading them only once
  + game\_config – config.Game\_Config
    - stores data about how the game is configured
    - persistently stores settings
    - stores all balancing variables in one place, so balancing the game will be easier
  + main\_screen – Menu\_System.Main
    - The screen that the user first sees when they start up the game
  + pause\_screen – Menu\_System.Pause
    - the screen that appears when the user pauses the game
  + options\_screen – Menu\_System.Options
    - the screen used for accessing different options screens
  + gfx\_options\_screen - Menu\_System.GFX\_Options
    - the screen used for configuring graphical options
  + snd\_options\_screen - Menu\_System.SND\_Options
    - the screen used for configuring sound options
  + scoreboard\_screen – Menu\_System.Scoreboard
    - the screen that shows all previous level scores
  + start\_screen – Menu\_System.Start
    - the screen that allows the user to configure the level before starting it
  + end\_screen – Menu\_System.End
    - the screen that shows at the end of a level
  + level – Menu\_System.Level
    - the screen that plays the game
* Methods:
  + VOID \_\_init\_\_ ()
    - starts pygame execution environment
    - initializes video output screen
    - initializes asset loaders
    - initializes all screens except level
    - get music folder path from config.music\_path
    - start music playing
    - Pushes main\_menu to game state stack
  + VOID run ()
    - Runs main game loop.
      * Calculate dt (time since last loop)
      * Collect events from event queue
      * Checks for video rescale events if resolution is set to re-scalable
        + Stores new resolution to config
        + Call rescale ()
      * Calls the correct tick function for the current screen (screen atop game state stack), passing in the event list and dt
      * If game state stack is empty, the main loop exits
      * If config.vsync is true, stall such that the loop takes 16 ms to complete
  + VOID start\_level (array size, int seed)
    - Initialize new level screen with size and seed
    - Push that level to the top of the game state stack
  + VOID load\_snd\_vol ()
    - get volumes for game and music from config
    - call pygame function to set game and music sound volumes
  + VOID rescale ()
    - Re initialize screen with config.resolution
    - Call rescale methods of all screens

Main – testing

**Test environment**:

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

**Test table**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Tested Functionality | Test conditions/ input | Test type | Expected behavior |
| 1 | main screen being pushed to GSS on startup | Start main file | normal | Console output saying it is on the main screen and prompting for input |
| 2 | start screen execution | Input: start | normal | Console output saying it is on start screen |
| 3 | level screen execution | Input: level | normal | Console output saying it is on level screen |
| 4 | end screen execution | Input: end | normal | Console output saying it is on end screen |
| 5 | scoreboard screen execution | Input: scoreboard | normal | Console output saying it is on scoreboard screen |
| 6 | pause screen execution | Input: pause | normal | Console output saying it is on pause screen |
| 7 | options screen execution | Input: options | normal | Console output saying it is on options screen |
| 8 | graphics options screen execution | Input: gfx | normal | Console output saying it is on graphics options screen |
| 9 | sound options screen execution | Input: snd | normal | Console output saying it is on sound options screen |
| 10 | Level screen initialisation | Input:level init 30 50 12314515 | normal | Console output saying level has been created with width 30, heigh 50 and seed 12314515  Console output saying it is on level screen |
| 11 | Level screen initialisation | Input:level\_init 30 -50 12314515 | invalid | Console output saying failed to create level: invalid input |
| 12 | Music player | Start main file | normal | Music starts playing when file is loaded |
| 13 | Music volume control | Music volume in config set to 0 then  Input:snd\_vol | Normal | Sound volume is set to mute |
| 14 | Music volume control | Music volume in config set to 1 then  Input:snd\_vol | Normal | Sound volume is set to full |
| 15 | Music volume control | Music volume in config set to -1 then  Input:snd\_vol | invalid | Sound volume is not set |
| 16 | Rescaling | Input: rescale  config.resolution  = 640 \* 480 | Normal | Display surface changes size  Console output from all screens saying they rescaled |

Module: maze\_gen

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

Maze\_GEN - Class: Maze

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (bool wall\_below,  bool wall\_right) | (bool wall\_below,  bool wall\_right) | (bool wall\_below,  bool wall\_right) | ⋯ | (bool wall\_below,  bool wall\_right) |
| (bool wall\_below,  bool wall\_right) | (bool wall\_below,  bool wall\_right) | (bool wall\_below,  bool wall\_right) | ⋯ | (bool wall\_below,  bool wall\_right) |
| ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| (bool wall\_below,  bool wall\_right) | (bool wall\_below,  bool wall\_right) | (bool wall\_below,  bool wall\_right) | ⋯ | (bool wall\_below,  bool wall\_right) |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wall(corner) | wall(edge) | wall(corner) | ⋯ | wall(corner) |
| wall(edge) | checkpoint |  | ⋯ | wall(edge) |
| wall(corner) | gateway | wall(corner) | ⋯ | wall(corner) |
| ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| wall(corner) | wall(edge) | wall(corner) | ⋯ | wall(corner) |

* + - 2d array structure to store a grid representation of the maze, where each cell is a tile in the maze
    - 2d Array structure provides random access, so all nodes are equally quick to work with
  + start – array (int x, int y)[axis\_index]
    - Start coordinate for the maze
  + end – array (int x, int y)[axis\_index]
    - End coordinate for the maze
  + start\_to\_end\_path – array ((node\_y, node\_x), (node\_y, node\_x), ⋯ (node\_y, node\_x)) [dist\_from\_start]
    - Provides a linear abstraction of the maze
    - Used to place blocks and gateways in the maze
    - Allows the maze to remain solveable
  + all\_sprites – SpriteGroup
    - store all sprites rendered during playing the level
    - The pygame.SpriteGroup data structure provides functionality for storing, updating and rendering sprites
  + maze\_walls – SpriteGroup
    - Stores all sprites that will be used to create the walls
    - The pygame.SpriteGroup data structure provides functionality for storing sprites
  + gateways – SpriteGroup
    - Stores all gateway sprites
    - The pygame.SpriteGroup data structure provides functionality for storing sprites
  + blocks – SpriteGroup
    - Stores all block sprites
    - The pygame.SpriteGroup data structure provides functionality for storing sprites
  + enemies – SpriteGroup
    - stores enemies in the maze
    - The pygame.SpriteGroup data structure provides functionality for storing sprites
  + checkpoints – SpriteGroup
    - stores checkpoints in the maze
    - The pygame.SpriteGroup data structure provides functionality for storing sprites
  + Keys – SpriteGroup
    - Stores keys in the maze
  + Exit – sprites.Exit
    - Is the final objective for the level
* Methods:
  + VOID \_\_init\_\_ (array maze\_size, int seed)
    - Stores maze size and maze seed to attributes
    - Initializes generation RNG with seed
    - Calls for maze layout to be generated
    - Calls to convert layout to wall sprites
    - Sets Start\_to\_end\_path using get\_shortest\_path
    - Calls populate method to place blocks, gateways, and enemies in the maze
  + VOID generate\_layout ()
    - uses kruskal’s algorithm to generate the maze layout
    - stores viable maze layout into layout attribute
  + VOID layout\_to\_board (funct wall\_generator)
    - Generates a 2d array of size msize\*2 + 1 by msize\*2 + 1
    - Initalise walls in set locations where both y index and x index are even to create corners
    - Initalise 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

Diagram

Description automatically generatedKruskal’s Algorithm:

A picture containing diagram

Description automatically generatedThe 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 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, 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 2: https://en.wikipedia.org/wiki/Kruskal%27s\_algorithm

Dijkstra’s Algorithm:

Diagram

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

Qr code

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

Figure 4: Flowchart of Dijkstra’s path finding algorithm

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

A picture containing diagram

Description automatically generated Maze Population algorithms:

These algorithms are responcible 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 solveable while making it harder to solve so that the game is more challenging and fun to play.

A picture containing polygon

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

The second algorithm, the Branch algorithm is not critical to level playability but makes the game much more enjoyable. It decides where to put the blocks in relation to the path to the exit, moving them away and of into the maze, so the player has to go out into the maze and explore for them. It does this by repeatedly choosing a random neighbour and advancing to that neigbour 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 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.

Figure 6: Flowchart for the Branch algorithm

Figure 5: Flowchart for the block and gateway placement algorithm

Diagram

Description automatically generated The third algorithm is responcible 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 an 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 the enemies

Figure 7: Flowchart for other maze population tasks

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 initiallised 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)
    - Initalise pos to start\_pos, or default to (0,0)
    - Initalise 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 heath 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 equals that of the player | Valid | Player passes through block |
| 11 | Player – collisions with blocks of different colours | Player is move towards a block who’s colour equals that of the player | Valid | Player stops at the edge of the block and doesn’t move through it |
| 12 | Enemies – navigation | Enemy spawned in the maze | Valid | Enemy moves towards a open space in the maze |
| 13 | Enemies – smooth movement | Enemy spawned in the maze | Valid | As enemy goes round turns, it follows a smooth path, not jittering around |
| 14 | Enemies – navigation around maze | Enemy spawned im 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 heath 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 inright 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 it true or false
  + Image – Surface
    - Stores the image to be rendered to the screen
  + rect – Rect
    - stores where on this toggle is located on screen
* Methods:
  + VOID \_\_init\_\_()
    - Initialize ticked to false
    - Initialize rect
    - Call rescale to render image
  + VOID update (dt)
    - If cursor is within rect and left mouse is clicked, negate ticked
  + VOID rescale ()
    - If ticked, set image to toggle\_ticked from img\_loader
    - If not ticked, set image to toggle\_unticked from img\_loader
    - Scale image to width and height of rect

Menu\_Sprites – Class: Slider

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

Menu\_Sprites – Class: Spinner

* Purpose: allows the user to select from a list of ordered predefined options
* Attributes:
  + options – array (string option1, string, option2 …) [option\_index]
    - Stores the possible options for this spinner
  + index – int
    - stores the index of which option is currently selected
  + font – Font
    - stores the loaded font used in this spinner
  + image – Surface
    - stores the image to be rendered to the screen
  + rect – Rect
    - stores the spinner’s location on screen
* Methods:
  + VOID \_\_init\_\_ (list options)
    - Store options to attribute
    - Initialize index to 0
    - Get font name from config
    - Initialize font
  + VOID update (dt)
    - If cursor is over the portion of the rect that represents the left button and left mouse button is pressed, decrement index and call rescale. If index is at zero play spinner\_end sound from snd\_loader
    - If cursor is over the portion of the rect that represents the right button and left mouse button is pressed, increment index and call rescale. If index is at length of options - 1, play spinner\_end sound from snd\_loader
  + VOID rescale ()
    - Set image to spinner\_buttons from img\_loader
    - Use font to render options[index]
    - Blit into center of image
    - Rescale image to width and height of rect

Menu\_Sprites – Class: Button

* Purpose: the user can press buttons to interact with the ui
* Attributes:
  + text – string
    - stores the text that the button says on it
  + font – Font
    - stores the loaded font used in this button
  + pressed – list (Bool left\_button, Bool middle\_button, Bool right\_button)[button\_index]
    - stores which buttons are pressed on this button on this frame
  + rising\_edges – list (Bool left\_button, Bool middle\_button, Bool right\_button)[button\_index]
    - stores which buttons have had a rising edge this frame
  + falling\_edges – list (Bool left\_button, Bool middle\_button, Bool right\_button)[button\_index]
    - stores which buttons have had a falling edge this frame
  + image – Surface
    - stores the image to be rendered to the screen
  + rect – Rect
    - stores the position of this button on screen
* Methods:
  + VOID \_\_init\_\_ (string text)
    - Store text as attribute
    - Get font name from config
    - Initialize font
  + VOID update (dt)
    - Set rising\_edges and falling\_edges to all falses
    - Get pressed mouse buttons and store to new\_pressed
    - For button\_index from 0 to 2
      * if new\_pressed[button\_index] is true and pressed[button\_index] is false, set rising\_edges[button\_index] to true
      * if new\_pressed[button\_index] is false and pressed[button\_index] is true, set falling\_edges[button\_index] to true
    - set pressed to new\_pressed
  + VOID rescale ()
    - Set image to button\_img from img\_loader
    - Use font to render text and blit this to image
    - Rescale image to width and height of rect

Menu\_Sprites – Testing

**Test environment**:

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

**Test table**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Tested Functionality** | **Test conditions/ input** | **Test type** | **Expected behavior** |
| 1 | Text – initialization | A text element is instantiated with “text1”and rendered | Valid | Text 1 appears on screen, using the font from font .config |
| 2 | Text – rescaling | Screen is rescaled | Valid | Text 1 still appears on the screen in the same position, now scaled down as to maintain the proportions on screen |
| 3 | Input\_box - initialization | Input box is instantiated with no default text | Valid | Input box appears on screen |
| 4 | Input\_box – default text | Input box is instantiated with default text “input box 1” | Valid | Default text renders within input box |
| 5 | Input\_box - selection | cursor is hovered over input\_box and left clicked | Valid | Input\_box is highlighted, indicating it will accept text input |
| 6 | Input\_box – registering allowed keystrokes | Input\_box is selected and allowed keys are pressed | Valid | The characters corresponding to the keystrokes appear in the input box |
| 7 | Input\_box – not registering disallowed keystrokes | input box is selected and disallowed keystrokes are pressed | Invalid | The characters of the corresponding keystrokes aren’t added to the input box |
| 8 | Input\_box – deselection | cursor is moved away from input box and left button is pressed | Valid | Input box is deselected: highlight goes away and keystrokes are no longer registered |
| 9 | Input\_box – rescaling | Screen is rescaled | Valid | Input box rescales as to maintain it’s proportions on screen |
| 10 | Toggle – initialization | New toggle box is initialised | Valid | New toggle object is created and rendered to screen with no crashing |
| 11 | Toggle – rendering | New toggle box is initialised | Valid | Toggle box appears on screen unticked |
| 12 | Toggle – toggling | cursor is moved over toggle and left button is pressed | Valid | Toggle changes state and renders that it is now in the other state |
| 13 | Toggle – rescaling | Screen is rescaled | Valid | Toggle rescales such that it maintains the same proportions on screen |
| 14 | Slider – initialization | New slider is initialized | Valid | New slider object appears on screen and causes no crashing |
| 15 | Slider - grabbing | cursor hovered over slider and left button is pressed | Valid | Slider thumb moves around so that it follows the mouse cursor |
| 16 | Slider – thumb stays on slider | Slider grabbed and cursor is moved around | Valid | If the cursor goes of the end of the slider, the thumb stops at the corresponding end until cursor returns |
| 17 | Slider – releasing | Left mouse button is released | Valid | Thumb stays in the same place it was in before left button was released |
| 18 | Slider – rescaling | Screen is rescaled | Valid | Slider rescales such that it’s proportions on screen are maintained |
| 19 | Spinner - initialization | A new spinner object is initialized with options: o1, o2, o3 | Valid | New spinner object is created and appears on screen |
| 20 | Spinner – right button | Cursor hovers over right button and right click is pressed | Valid | Spinner moves to next option |
| 21 | Spinner – left button | Cursor hovers over left button and left click is pressed | Valid | Spinner moves to previous option |
| 22 | Spinner - ends | Left and right buttons are pressed until spinner gets to either end | Valid | Spinner moves to the final option and then doesn’t go any further and doesn’t wrap around |
| 23 | Spinner – rescaling | Screen is rescaled | Valid | Spinner rescaled as to maintain the same proportions on screen |
| 24 | Button – initialization | A new button object is initialized with text b1 | Valid | Button object appears on screen with correct text and font |
| 25 | Button – press registration | Cursor hovers over button and mouse buttons are pressed | Valid | Console output saying which buttons are pressed equating to the mouse buttons that are currently pressed |
| 26 | Button – rising edge registration | Cursor hovers over button and mouse buttons are pressed | Valid | Console output saying when there is a rising edge of each button as they are depressed |
| 27 | Button – falling edge detection | Cursor hovers over button and mouse buttons are pressed | Valid | Console output saying when there is a falling edge of each button as they are released |
| 28 | Button – clicking elsewhere on the screen | Cursor is moved elsewhere on screen and mouse buttons are pressed | Valid |  |
| 29 | Button - rendering | Cursor hovers over button and mouse buttons are pressed | Valid | Button visually indicates that has been pressed |
| 30 | Button – rescaling | Screen is rescaled | Valid | Button rescales to maintain same proportions on screen |

Module: Menu\_System

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

Menu\_System – Class: UI\_Screen

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

Menu\_System – Class: Main

* Purpose: updates and renders the main menu screen
* Inherits from UI\_Screen
* Attributes:
  + Elements – inherited from UI\_Screen
  + Background - inherited from UI\_Screen
  + main\_title\_text – Menu\_Sprites.Text
    - the text at the top of the title screen
  + Start\_Button – Menu\_Sprite.Button
    - Allows the user to open the start menu
  + options\_button – Menu\_Sprite.Button
    - Allows the user to open the options menu
  + scoreboard\_button – Menu\_Sprite.Button
    - Allows the user to open the Scoreboard
  + close\_button – Menu\_Sprite.Button
    - Removes all items from the game\_state\_stack to close the game
* Methods:
  + VOID \_\_init\_\_ ()
    - Initializes maint\_title\_text, start\_button, options\_button, scoreboard\_button and close\_button
    - Loads background image from image loader and stores into attribute
    - Call rescale to render image
  + VOID tick (array event\_list, float dt)
    - Call parent tick method
    - If a button has been pressed, push the corresponding screen onto the game state stack
    - If the close button has been pressed, clear the game state stack
  + VOID rescale ()
    - Set rect of all elements such that they are stacked vertically in the center of the screen
    - Call parent rescale method

Menu\_System – Class: Start

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

Menu\_System – Class: End

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

Menu\_System – Class: Scoreboard

* Purpose: update and render the scoreboard screen so the player can see how well they have done
* Attributes:
  + Elements – inherited from UI\_Screen
  + Background - inherited from UI\_Screen
  + title\_text – menu\_sprites.Text
    - Title to indicate to the user which menu they are on
  + scoreboard\_data – array [entry\_index][field\_index]

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

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

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 |  |  |  |

# 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 foolder

        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 = 0.08333333333333333

        self.maze\_blocks\_distance\_proportion = 0.16666666666666666

        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 |  | Pass |
| 2 | Test loading int attributes | Host loads player\_max\_health  And print it | Valid | Outputs the player\_max\_health value |  | Pass |
| 3 | Test loading bool attributes | Host loads fullscreen and prints it | Valid | Outputs the fullscreen value |  | Pass |
| 4 | Test loading float attributes | Host loads game\_vol and prints it | Valid | Outputs the game\_vol value |  | Pass |
| 5 | Test loading array attributes | Host loads resolution and prints it | Valid | Outputs the resolution stored there |  | 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 |  | Fail:  Strings are not being formatted correctly when rewriting to the config file:  for identifier, val in self.\_\_dict\_\_.items():              self\_file\_str = re.sub(f" self.{identifier} = .+\n",                                     f" self.{identifier} = {val}\n",                                     self\_file\_str)  solution: adjust the formatting code such that it will include parentheses correctly:  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) |
| 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 | self.player\_max\_speed = 30 | 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 | self.vsync = True | 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 | self.music\_vol = 0.3 | 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 | self.text\_colour = (123, 45, 67) | Pass |
| 11 | Test reloading string attributes | Host reloads snd\_path and prints it | Valid | Outputs the snd\_path saved there |  | Pass |
| 12 | Test reloading int attributes | Host reloads player\_max\_speed  And print it | Valid | Outputs the player\_max\_speed value |  | Pass |
| 13 | Test reloading bool attributes | Host reloads vsync and prints it | Valid | Outputs the vsync value |  | Pass |
| 14 | Test reloading float attributes | Host reloads music\_vol and prints it | Valid | Outputs the music\_vol value |  | Pass |
| 15 | Test reloading array attributes | Host reloads text\_colour and prints it | Valid | Outputs the text\_colour stored there |  | 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 the 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

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

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\_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):

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

        image = False

        # check already loaded assets for the image

        if img\_name in self.assets.keys():

            return self.assets[img\_name]

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 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):

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

                    # 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)

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

        # 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

Convert\_alpha allows the image to have transparency render correctly

        # no image was found

        return False

class Snd\_Loader():

    def \_\_init\_\_(self, game):

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

        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 = {}

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

        # 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

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.

    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

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: Text

Description automatically generated

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:

Text

Description automatically generated

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          # 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())  Solution1: load no\_sound.wav from the snd folder  Solution2: don’t call the parent path          # 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()) |

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)

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

        self.scoreboard\_screen = MS.Scoreboard(self)

        self.start\_screen = MS.Start(self)

        self.end\_screen = MS.End(self)

        self.level = False

Music loops is set to -1 to enable continuous looping

        # 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:

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

                # calculate dt

                dt = clock.tick(60)

            else:

                dt = clock.tick()

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

            # event collect events

            event\_list = list(pg.event.get())

            # check events

            for event in event\_list:

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

                # 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()

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

            # 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()

The level screen is only rescaled if it already exists

        self.end\_screen.rescale()

        if self.level:

            self.level.rescale()

Game()

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 call’s it’s 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 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:

Text

Description automatically generated

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.

Text

Description automatically generated

Solution 1:

Don’t reuse rescaling code for initialising the screen:

Text

Description automatically generated

Error 2:

Text

Description automatically generated

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

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 |  | Pass |
| 2 | start screen execution | Input: start | normal | Console output saying it is on start screen |  | Pass |
| 3 | level screen execution | Input: level | normal | Console output saying it is on level screen |  | Pass |
| 4 | end screen execution | Input: end | normal | Console output saying it is on end screen |  | Pass |
| 5 | scoreboard screen execution | Input: scoreboard | normal | Console output saying it is on scoreboard screen |  | Pass |
| 6 | pause screen execution | Input: pause | normal | Console output saying it is on pause screen |  | Pass |
| 7 | options screen execution | Input: options | normal | Console output saying it is on options screen |  | Pass |
| 8 | graphics options screen execution | Input: gfx | normal | Console output saying it is on graphics options screen |  | Pass |
| 9 | sound options screen execution | Input: snd | normal | Console output saying it is on sound options 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 |  | Pass |
| 11 | Level screen initialisation | Input:level\_init 30 -50 12314515 | invalid | Console output saying failed to create level: invalid input |  | Fail: validation missing  def start\_level(self, size, seed):          # initialise new level          self.level = MS.Level(self, size, seed)  Solution: implement validation  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) |
| 12 | Music player | Start main file | normal | Music starts playing when file is loaded | no sound.wav starts playing | Fail: music.wav is missing  self.music = self.snd\_loader.get("music.wav")  Solution: add music file and change the loader request to match it:  self.music = self.snd\_loader.get("Vexento - Lotus.wav") |
| 13 | Music volume control | Music volume in config set to 0 then  Input:snd\_vol | Normal | Sound volume is set to mute |  | Pass |
| 14 | Music volume control | Music volume in config set to 1 then  Input:snd\_vol | Normal | Sound volume is set to full |  | Pass |
| 15 | Music volume control | Music volume in config set to -1 then  Input:snd\_vol | invalid | Sound volume is not set |  | Pass |
| 16 | Rescaling | Input: rescale  config.resolution  = 640 \* 480 | Normal | Display surface changes size  Console output from all screens saying they 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

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

        self.all\_sprites = pg.sprite.Group()

        self.maze\_walls = pg.sprite.Group()

        self.gateways = pg.sprite.Group()

        self.blocks = pg.sprite.Group()

        self.enemies = pg.sprite.Group()

        self.checkpoints = pg.sprite.Group()

        self.keys = pg.sprite.Group()

Ensures identical random mazes can be generated

        # initialise RNG

        rng.seed(self.seed)

        # generate maze layout

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

        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

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

        unchecked\_walls = []

        # - 1 stops it from checking bottom most walls

        for y in range(0, self.msize[1] -1 ):

            # - 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]

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

            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

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

        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

        # initalise 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])]

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

        # place perimiter 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]][x] = wall\_gen\_group((x, bsize[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))

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

        # 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

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

        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 bellow

                    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))

    def populate(self):

        """populates the maze with sprites"""

        # populate gateways and blocks

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

        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() \* \

The place blocks and gateways algorithm from the design section

                          self.game.confg.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)

Gateways are placed if the value is less than the threshold

            # 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

        # populate keys

        for \_ in range(self.game.config.maze\_key\_count):

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

            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"""

Standard Dijkstra’s Implementation using the board to get adjacencies

        # 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

                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]

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

    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)

        return current\_node\_pos

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

    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]]

Stage 4: Unit Testing

**Issue Resolution:**

Error 1:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

Problem 5:

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



 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}



Error 6:

Text

Description automatically generated

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:

Text

Description automatically generated

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 = end

        while known\_nodes[current\_node)] != False:

Solution 7:

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



        current\_node = tuple(end)

        while known\_nodes[current\_node)] != False:

Error 8:

Text

Description automatically generated

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:

Text

Description automatically generated

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"""

        # 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:

Text

Description automatically generated

Problem 10:

Misspelled config:

 self.game.confg.maze\_gateway\_jitter)

Solution 10:

Spell config correctly:

 self.game.config.maze\_gateway\_jitter)

Error 11:

Text

Description automatically generated

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)



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)



Error 12:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated with medium confidence

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:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. | Tested Functionality | Test conditions/ input | Test type | Expected behavior | Resultant Behavior | Pass / Fail + Solution |
| 1 | Maze initialization | New maze object instantiated | Valid | New maze object is created with the specified parameters without causing any errors |  | Pass |
| 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 |  | Fail­: there are multiple zone values [index 2 of each list] meaning there are in accessible slots  Problem : when a zone merges with another zone, all nodes in both zones must be given the new index, not just the node that caused the zones to merge  layout[y+1][x][2] = zone1  Solution: iterate through the layout and check for any replaced all occurrences of the second zone with the first zone  # merge zones  for row in layout:      for node in row:          if node[2] == zone2:              node[2] = zone1 |
| 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 |  | Fail: layout missing vertical edge walls  Problem: code that generates vertical walls places walls bellow the current node, not to the right:  if self.layout[ly][lx][1]: # wall right      board[by][bx+1] = wall\_gen\_group((bx, by+1))  Solution: ensure wall is placed in the correct location  if self.layout[ly][lx][1]: # wall right      board[by][bx+1] = wall\_gen\_group((bx+1, by)) |
| 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 | 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 |  | 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 |  | 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  neighbour = [current\_node\_pos[i] + offset[i] for i in (0,1)]  Solution:  Convert neighbour to a tuple                  neighbour = tuple(neighbour) |
| 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 |  | 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:

A picture containing text

Description automatically generated

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:

A picture containing graphical user interface

Description automatically generated

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

Default values for variable are initialized

        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):

Advancing to the next frame of animation

        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 \

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 to expensive and complicated to implement

           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.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)

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

        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], \

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

                          [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")

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

        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

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

        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

Movement input keys: implements 4 way movement for the player

            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

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

            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

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

                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:

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

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

        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:

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

            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:

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

            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"""

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 it’s corresponding board position correctly

        # 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

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

            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 wierd snapping

        # - no dependency on velocities, which can cause problems

        def check\_collidable(sprite):

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

             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 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 = 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)

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

    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

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

        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)

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 rendered in front or behind, and in different locations, depending on what is best

                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):

        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]

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

        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))

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 (djikstra’s algorithm) to find a path to this location

        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

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

        # 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))

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

            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.Rect(0,0,self.rect[2], self.rect[2])

        self.hit\_rect.bottomleft = screen\_pos

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 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(f"crate light red")]

            case 1:

                self.imgs = [game.img\_loader.get(f"crate orange")]

            case 2:

                self.imgs = [game.img\_loader.get(f"crate lime")]

            case 3:

                self.imgs = [game.img\_loader.get(f"crate dark blue")]

            case 4:

                self.imgs = [game.img\_loader.get(f"crate purple")]

            case 5:

                self.imgs = [game.img\_loader.get(f"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 deactive 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

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.

        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))

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

            # 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]]

    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

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

        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 of 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

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

        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

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

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 scalled correctly

        self.total\_time += dt/1000

    def reset(self):

        self.total\_time = 0.0

        self.start\_time = time.time()

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()

        # initalise 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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text, timeline

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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]

Text

Description automatically generated

Solution 7:

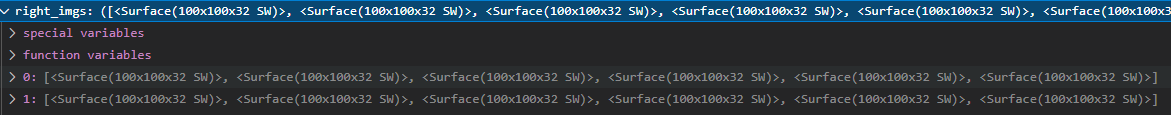
fix right\_imgs initialisation so that it is a 2x6 array

        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]



Error 8:

Text

Description automatically generated with low confidence

Problem 8:

Player was passed level object instead of game object

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



Solution 8:

Pass player the correct game object argument

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

Error 9:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated with medium confidence

Problem 12:

Imgs 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:

Graphical user interface, application

Description automatically generated

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:

A picture containing chart

Description automatically generated

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'

Text

Description automatically generated

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()

Text

Description automatically generated

Error 15:

Text

Description automatically generated

Problem 15:

Poorly formatted xml in on of the sprite sheets

    <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:

A picture containing bar chart

Description automatically generated

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

Graphical user interface

Description automatically generated

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 | player is missing from the maze | 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:              # 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)  Solution 1: update render and draw player and camera separately:              # 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)  Problem 2:    The player’s sprite is only initialized once it has been rendered at least once.  Solution 2: render the player once before updating it      # render once to set up rect for collisions          self.walking = False          self.render(0)  Problem 3:    Player’s images aren’t being stored correctly after they are loaded         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  }  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:    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:  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 collisions:          # 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  Solution 5:  Render all sprites once after they have been initialized but before they are updated by the main loop:          # initalise sprites in maze          for sprite in self.maze.all\_sprites:                  sprite.render(0) |
| 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 | Fail: Problem:  One walking variable is used to track whether both x and y are changing, but this should be done per axis  Solution: implement deceleration for each axis separately:          # 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 |
| 4 | Player – standing animation | No keys are pressed | Valid | Player displays the standing animation | Player stands there with walking animation: | Pass |
| 5 | Player – walking animation | Wasd or arrow keys are pressed | Valid | Player displays walking animation | Player goes through walking animation while walking: | Fail : Problem 1:    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              self.imgs = self.standing\_imgs[tuple(facing\_dir)]              self.frame\_index = 0 |
| 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 | player phases through the wall | Fail:  Problem: wall collisions aren’t registering because the type checking isn’t correct:              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              ):  Solution:  Correctly check the type of sprite the player is colliding with:              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              ): |
| 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: | Fail:  Problem: collisions don’t care if the side is exposed or not  # wall on left                      self.vel.x = max(0, self.vel.x)  Solution check if the side is exposed before colliding with it                      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 = max(0, self.vel.x) |
| 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 | Fail:  Problem: enemy can’t choose a target location they can navigate to:      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)                  break              except:                  # failed to find a location that can be navigated to, try again                  pass  Solution:  Select a location using the maze’s branch method: then the location will be accessible |
| 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 heath 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  if slot\_keys[slot\_index] and not slot\_keys[slot\_index]:  Solution 1: fix edge detection code  if slot\_keys[slot\_index] and not self.prev\_slot\_keys[slot\_index]:  Problem 2: it isn’t correctly checking if there is a block in front of the player:          if type(facing\_sprite) == "Block":  Solution 2; check type’s name is equal to block instead:          if type(facing\_sprite).\_\_name\_\_ == "Block": |
| 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 | Fail:  Problem 1:  Positioning code doesn’t work correctly, so the sprite position isn’t in a visible location          # 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)  Solution 1: correctly calculate the position of the sprite          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  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:                  image = pg.surface.Surface(self.image.get\_size())  Solution 2: Enable per pixel transparency for the surface:                  image = pg.surface.Surface(self.image.get\_size(),                                             flags = pg.SRCALPHA) |
| 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 | Block is placed in front of the player | Pass |
| 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 | Pass |
| 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 | Pass |
| 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 | pass |
| 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 | pass |
| 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 | Fail:  Problem 1: Placing can’t detect if it is trying to place on a gateway due to incorrect type checking:          if type(facing\_sprite) == "Gateway"  Solution 1: correctly get the name of the type it is trying to place on          if type(facing\_sprite).\_\_name\_\_ == "Gateway"  Problem 2: the block can only be placed on a gateway of the same colour as the player          if type(facing\_sprite).\_\_name\_\_ == "Gateway" and \                  facing\_sprite.colour == self.colour:  solution 2: check of the gateway is the same colour as the block, not the same colour as the player          if type(facing\_sprite).\_\_name\_\_ == "Gateway" and \                  facing\_sprite.colour == self.inventory[slot\_index].colour: |
| 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 | Fail:  Problem:    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              if last\_checkpoint := self.game.level.player.last\_checkpoint:                  last\_checkpoint.deactivate() |
| 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 | 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          if self.last\_checkpoint != False:              self.pos = self.last\_checkpoint.pos  solution: copy position before assignment to break link between them          if self.last\_checkpoint != False:              self.pos = vec2(self.last\_checkpoint.pos) |
| 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 | Fail:  Problem: the surface the key is blitted onto doesn’t support per pixel transparency:              displaced\_img = pg.surface.Surface(key\_image.get\_size() +                                                   vec2(0,key\_displacement))  solution enable per pixel transparency for the surface:              displaced\_img = pg.surface.Surface(                  key\_image.get\_size() + vec2(0,key\_displacement),                  flags = pg.SRCALPHA) |
| 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: | 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          unscaled\_scrn\_size = vec2(self.game.config.resolution)/ self.zoom  Solution:  Account for sprite dimensions:          unscaled\_scrn\_size = vec2(self.game.config.resolution)/ self.zoom / 16 |
| 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")

        # initalise 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.vale = 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.img, 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:

A picture containing text

Description automatically generatedProblem 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:

Text

Description automatically generated

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:

Text

Description automatically generated

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:

Text

Description automatically generated with low confidenceError 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:

Text

Description automatically generated

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:

Text

Description automatically generatedProblem 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 | Fail:  Problem: not correctly calculating the text’s position on this sprite’s image:          text\_img\_rect.center = self.rect.center  Solution: correctly calculate text’s position on the image:          text\_img\_rect.center = vec2(self.rect.size) / 2 |
| 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 | Crashes | Fail:  Problem: when processing mouse events, it checks which button on the mouse has be pressed, but it is trying to check event.key:  if event.type == pg.MOUSEBUTTONDOWN and event.key == 0:  Solution: Access event.button for the mouse buttons:  if event.type == pg.MOUSEBUTTONDOWN and event.button == 0: |
| 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: | Fail:  Problem: an keydown events aren’t in the expected format: expected variables like pg.K\_left, but they provide Unicode characters instead:    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 it’s 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:  self.ticked\_img = self.game.img\_loader.get("toggle\_ticked")  self.unticked\_img = self.game.img\_loader.get("toggle\_unticked")  Solution: load correct image name:  self.ticked\_img = self.game.img\_loader.get("toggle ticked")  self.unticked\_img = self.game.img\_loader.get("toggle unticked") |
| 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  for event in events:      if event.type == pg.MOUSEBUTTONDOWN and event.button == 1:          self.ticked = not self.ticked          self.rescale()  Solution : add checking mouse position to update code:  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() |
| 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 | Pass |
| 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 | Fail:  Problem: not getting just the x component of the mouse’s position  self.val = (pg.mouse.get\_pos() - self.rect.left) / self.rect.width  Solution: only use the x part of the mouse’s position  mouse\_disp = pg.mouse.get\_pos()[0] - self.rect.left  self.val = (mouse\_disp - (self.thumb\_width/2)) /        \              (self.rect.width - self.thumb\_width) |
| 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 | Pass |
| 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 | Pass |
| 18 | Slider – rescaling | Screen is rescaled | Valid | Slider rescales such that it’s proportions on screen are maintained | slider rescales to a different size | pass |
| 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 | pass |
| 20 | Spinner – right button | Cursor hovers over right button and right click is pressed | Valid | Spinner moves to next option | moves to next option | pass |
| 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 | pass |
| 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 | 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 |  | Fail:  Problem: falling and rising edge arrays not initialized correctly:  self.falling\_edges = [False \* 3]  Solution: correctly initialize arrays:  self.falling\_edges = [False] \* 3 |
| 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 …….

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

Stage 7: Unit testing

Stage 7: Stakeholder Review

# D. Evaluation

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

# Project Appendixes

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

These might include, but are not limited to:

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