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 4](#_Toc92707136)

[Outline 4](#_Toc92707137)

[Stakeholders 4](#_Toc92707138)

[Game Research: Tetris 5](#_Toc92707139)

[Game research: Hue 7](#_Toc92707140)

[Game Research: World’s Hardest Game 10](#_Toc92707141)

[Survey 11](#_Toc92707142)

[Planning 11](#_Toc92707143)

[Survey Response review 12](#_Toc92707144)

[Interview 15](#_Toc92707145)

[Proposed Feature List 15](#_Toc92707146)

[Limitations and Scope 16](#_Toc92707147)

[Why this Solution is Suited to a Computation Solution 17](#_Toc92707148)

[Abstraction 17](#_Toc92707149)

[Thinking Ahead 18](#_Toc92707150)

[Thinking Procedurally 18](#_Toc92707151)

[Thinking Logically 18](#_Toc92707152)

[Thinking Concurrently 19](#_Toc92707153)

[Hardware and Software Requirements 20](#_Toc92707154)

[Success Criteria 20](#_Toc92707155)

[Graphics - 1 20](#_Toc92707156)

[User interface - 2 22](#_Toc92707157)

[Sound - 3 24](#_Toc92707158)

[Level design - 4 25](#_Toc92707159)

[Game mechanics - 4 27](#_Toc92707160)

[B. Design 29](#_Toc92707161)

[Systems diagram 29](#_Toc92707162)

[Overall program layout 29](#_Toc92707163)

[Module Declarations 30](#_Toc92707164)

[Module: Main.py 30](#_Toc92707165)

[Module: maze\_gen.py 32](#_Toc92707166)

[Module: Sprites.py 37](#_Toc92707167)

[Module: Asset\_loader 42](#_Toc92707168)

[C. Developing the coded solution (“The development story”) 43](#_Toc92707169)

[D. Evaluation 43](#_Toc92707170)

[Project Appendixes 44](#_Toc92707171)

# 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 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:

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.

Graphical user interface, text, application, email

Description automatically generated

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

## Interview

## 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 |
| Hazards 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 |
| 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

### 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 the player hits a wall, they stop moving in the axis of collision | Stops players moving through walls | All games researched |
| 4 | 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 |
| 5 | 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 |

#### hazard controller - b

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Requirement | Function | Source |
| 1 | Defines a point in the maze to go towards | Gives the hazard a place to go to | Game research: Hue  Proposed features: hazards |
| 2 | Path finds to get to the defined point. | Acts as a simple AI for the hazards to follow | Proposed features:  Hazards |
| 3 | When the player places a block, re evaluates path | If a placed block obscures the path, the hazard 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 hazard’s movement more fluid and predictable | Proposed features: hazards |

# B. Design

## Systems diagram

The systems diagram shows that I have chosen a top-down approach to designing 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.

## Module Declarations

### Module: Main.py

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 – 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
  + all\_sprites – spriteGroup
    - stores all sprites that are currently updated and rendered in the game
    - provides an iterable to iterate over to update all sprites and then to render all sprites
  + 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
  + game\_timer – Sprites.timer
* Methods:
  + VOID \_\_init\_\_ ()
    - starts pygame execution environment
    - initializes video output screen
    - initializes asset loaders
    - Pushes tick\_main\_menu to game state stack
  + VOID run ()
    - Runs main game loop.
      * Collect events from event queue
      * Checks for video rescale events if resolution is set to re-scalable
        + Rescales display surface
      * Calls the correct tick function for the current menu / screen, passing in the event list
      * If game state stack is empty, the main loop exits
  + VOID tick\_main\_menu (array event\_list)
    - Renders main menu screen
    - Checks to see if any on screen buttons have been pressed
      * If so, pushes relevant menu onto the game state stack
    - Pops all items from game state stack if quit is pressed
  + VOID tick\_pause (array event\_list)
    - Renders pause screen
    - If options button pressed, pushes tick\_options function onto the game state stack
    - If resume button pressed the top is popped from the game state stack
    - Pops all items from game state stack and pushes tick\_main\_menu if quit is pressed
  + VOID tick\_options (array event\_list)
    - Renders options screen
    - Checks to see if buttons have been pressed
      * If close button pressed the top is popped from the game state stack
      * If so, pushes relevant tick function onto the game state stack
    - If esc is pressed, the top is popped from the game state stack
  + VOID tick\_GFX\_options (array event\_list)
    - Renders graphics options screen
    - If close button pressed the top is popped from the game state stack
    - Checks to see if any tick boxes have been pressed
      * toggles appropriate variables
    - checks to see if any buttons have been pressed
      * changes corresponding variables
    - checks to see if apply button has been pressed
      * if so, applies graphics changes.
    - If esc is pressed, the top is popped from the game state stack
  + VOID tick\_SND\_options (array event\_list)
    - Renders Sound options screen
    - If close button pressed the top is popped from the game state stack
    - Checks to see if any sliders have been clicked on
      * If so, moves slider to mouse position
    - Immediately apply changes to sound settings
    - If esc is pressed, the top is popped from the game state stack
  + VOID tick\_scoreboard(array event\_list)
    - Renders scoreboard screen
    - Shows the name, time, map size and seed for each entry
  + VOID tick\_start ()
    - Renders level config menu
    - allows a player to set up the level to play
    - Generates a random seed and places that in seed input box
    - If user clicks on seed input box and types, their seed is typed in the box
    - Generates default maze size
    - If user clicks on maze size input box and types, the new size is typed in the box
    - Checks for start button to be clicked
      * Initializes a new maze object with the required parameters and stores as attribute
      * Pop tick\_start from game state stack and push tick\_game
  + VOID tick\_game (array event\_list)
    - 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 all sprites
    - Renders background
    - Renders all sprites
  + VOID tick\_end (array, event\_list)
    - Renders end screen
    - Shows seed and time taken to complete
    - Allows user to enter their name to be put onto the scoreboard
    - Renders scoreboard for this seed
    - If esc key is pressed, clear game state stack and push tick\_main\_menu

### Module: maze\_gen.py

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
  + 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
  + player – sprites.Player
    - the player sprite that the user controls to navigate the level
  + 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

#### Diagram Description automatically generatedMaze\_GEN - Algorithms

Kruskal’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 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 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.

Maze Population algorithms:

Text

Description automatically generated with medium confidence 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 is the most difficult to make meet it’s success criteria. 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 will 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 he 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.

Diagram, shape

Description automatically generatedThe 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

### Module: Sprites.py

* stores 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 function to ensure this sprite can be updated without the game crashing because it can’t find an update function for a sprite
  + VOID render (dt)
    - Decrease frame countdown by dt
    - If dt < 0, advance frame\_index to index of next frame
    - Retrieve correct image from imgs
    - Rotates image according to rot
    - Set rect to position to render on the screen using camera position

#### sprites – Class: Player

* Purpose: the sprite the user controls to move around the maze
* Inherits from sprites.renderable\_sprite
* Attributes:
  + pos – inherited from renderable\_sprite
  + rot – inherited from renderable\_sprite
  + imgs – inherited from renderable\_sprite
  + frame\_index – inherited from renderable\_sprite
  + frame\_countdown – inherited from renderable\_sprite
  + frame\_time – inherited from renderable\_sprite
  + colour – tuple (int R, int G, int B)
    - stores the current colour of the player
  + health – int
    - stores how much heath the player has, going from 100 to 0
  + hurt\_cooldown – int
    - ms until the player can be attacked again. Acts as a timer for the player to flash red when hurt
  + 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
    - Load all animation frames into imgs using img\_loader
  + VOID update (dt)
    - if arrows or wasd is pressed, set animation\_state to “walking” and moves player in the corresponding direction
    - if arrows or wasd are not pressed, set animation state to “standing”
    - 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
    - checks for collisions with walls of all colours other than the current one and prevents player walking through them
    - If health = 0, call respawn
  + VOID pick\_up (int slot\_index)
    - Determine distance to each block
    - if the board coordinate in front of the player contains a block and inventory[slot\_index] isn’t full:
    - store this block to inventory[slot\_index
    - Play block collection sound
    - Remove block from maze.blocks and game.all\_sprites
    - Store block in inventory[slot\_index]
  + VOID place (int slot\_index)
    - Find board coordinate of the tile in front of player
    - if the board coordinate is empty in maze board store the wall in inventory[slot\_index] there,
    - Remove wall from inventory[slot\_index]
    - Add wall back to all\_sprites and maze.blocks
    - Update block’s pos to new position
    - Else If the board coordinate where it was going to place in maze board is an gateway, delete both the block and the gateway.
  + VOID respawn ()
    - Play respawn sound
    - If last\_checkpoint isn’t Null Set pos to last\_checkpoint’s position
    - Else set pos to maze.start
  + 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
  + 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
    - checks for collisions with walls and prevents them from happening.
    - 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 tick\_end 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 centered on screen
    - array allows for fast and mutable access to the position
* Methods:
  + VOID \_\_init\_\_ ()
    - Initalises 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
  + tuple wrld\_2\_scrn\_coord(tuple wrld\_coord)
    - transform the world coordinate to a screen coordinate using the camera position and screen size
    - return screen coordinate

#### Sprites – Testing

### 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 img\_folder, load it from a file to a surface
    - set the colorkey of that surface so that it is transparent
    - return the surface

#### Asset\_Loader – Class: Snd\_Loader

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

#### Asset\_loader – Class: Sprite\_Sheet

* Purpose: stores a spritesheet image and provides functionality for interacting with it, allowing individual sprites to be extracted and retrieved
* Attributes:
  + name – String: contains name of spritesheet for retrieval
  + path – String: the path of the sprite sheet on the disk
  + img – Surface: stores the image of the spritesheet after it is loaded, providing quick access to It as it is stored in memory
  + sprite\_coords – dictionary[sprite\_name]: stores the rect of each sprite within the spritesheet, and as it is a hash table style data structure, all data is equally fast to access.
* Methods:
  + VOID \_\_init\_\_ (string sheet\_name, string directory)
    - Stores sheet name as attribute
    - Define sheet\_path: directory / sheet\_name
    - Load image from sheet\_path + .jpg and store to attribute
    - Call load\_xml
  + VOID load\_xml ():
    - Define sheet\_xml\_path: sheet\_path + .xml
    - Load xml sheet from path
    - Convert to from xml to sprite\_coords dictionary
    - Store sprite\_coords dictionary as attribute
  + Surface get\_sprite (string sprite\_name)
    - retrieve sprite\_rect from sprite\_coords
    - Store sprite\_rect subsurface of img to sprite\_img Surface
    - Return sprite\_img Surface

### 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:

All sprite animation frame names

Sprite sheet names

Sprite movement speeds

Graphics settings

sound settings

* + Player\_health – int
  + Player\_hurt\_cooldown - int

# C. Developing the coded solution (“The development story”)

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

# D. Evaluation

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

# Project Appendixes

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

These might include, but are not limited to:

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