

NEA

Toby Mok

<b>1 Analysis</b>	<b>2</b>
1.1 Background . . . . .	2
1.1.1 Game Description . . . . .	2
1.1.2 Current Solutions . . . . .	3
1.1.3 Client Interview . . . . .	4
1.2 Objectives . . . . .	5
1.2.1 Client Objectives . . . . .	5
1.2.2 Other User Considerations . . . . .	6
1.3 Research . . . . .	7
1.3.1 Board Representation . . . . .	7
1.3.2 CPU techniques . . . . .	8
1.3.3 GUI framework . . . . .	9
1.4 Proposed Solution . . . . .	9
1.4.1 Language . . . . .	9
1.4.2 Development Environment . . . . .	10
1.4.3 Source Control . . . . .	11
1.4.4 Techniques . . . . .	11
1.5 Limitations . . . . .	12
1.6 Critical Path Design . . . . .	12
<b>2 Design</b>	<b>14</b>
2.1 System Architecture . . . . .	14
2.1.1 Main Menu . . . . .	15
2.1.2 Settings . . . . .	15
2.1.3 Past Games Browser . . . . .	17
2.1.4 Config . . . . .	18
2.1.5 Game . . . . .	19
2.1.6 Board Editor . . . . .	20
2.2 Algorithms and Techniques . . . . .	21
2.2.1 Minimax . . . . .	21
2.2.2 Minimax improvements . . . . .	22
2.2.3 Board Representation . . . . .	26
2.2.4 Evaluation Function . . . . .	30
2.2.5 Shadow Mapping . . . . .	33
2.2.6 Multithreading . . . . .	37
2.3 Data Structures . . . . .	37
2.3.1 Database . . . . .	37
2.3.2 Linked Lists . . . . .	39
2.3.3 Stack . . . . .	40
2.4 Classes . . . . .	41
2.4.1 Class Diagram . . . . .	45
<b>3 Technical Solution</b>	<b>46</b>
3.1 File Tree Diagram . . . . .	47
3.2 Summary of Complexity . . . . .	48
3.3 Overview . . . . .	48
3.3.1 Main . . . . .	48
3.3.2 Loading Screen . . . . .	49
3.3.3 Helper functions . . . . .	51

3.3.4	Theme . . . . .	59
3.4	GUI . . . . .	60
3.4.1	Laser . . . . .	60
3.4.2	Particles . . . . .	63
3.4.3	Widget Bases . . . . .	66
3.4.4	Widgets . . . . .	75
3.5	Game . . . . .	87
3.5.1	Model . . . . .	87
3.5.2	View . . . . .	92
3.5.3	Controller . . . . .	98
3.5.4	Board . . . . .	103
3.5.5	Bitboards . . . . .	108
3.6	CPU . . . . .	114
3.6.1	Minimax . . . . .	114
3.6.2	Alpha-beta Pruning . . . . .	116
3.6.3	Transposition Table . . . . .	118
3.6.4	Iterative Deepening . . . . .	119
3.6.5	Evaluator . . . . .	120
3.6.6	Multithreading . . . . .	123
3.6.7	Zobrist Hashing . . . . .	124
3.6.8	Cache . . . . .	125
3.7	States . . . . .	127
3.7.1	Review . . . . .	127
3.8	Database . . . . .	133
3.8.1	DDL . . . . .	133
3.8.2	DML . . . . .	134
3.9	Shaders . . . . .	137
3.9.1	Shader Manager . . . . .	137
3.9.2	Bloom . . . . .	142
3.9.3	Rays . . . . .	145
<b>4</b>	<b>Testing</b> . . . . .	<b>150</b>
4.1	Iterative Testing . . . . .	150
4.1.1	Minimax . . . . .	150
4.1.2	Migrations . . . . .	151
4.2	Unit Tests . . . . .	151
4.2.1	Board Evaluator . . . . .	151
4.2.2	CPU . . . . .	151
4.2.3	Shadow Mapping . . . . .	152
4.3	Final Tests . . . . .	152
4.3.1	Objective 1 . . . . .	152
4.3.2	Objective 2 . . . . .	152
4.3.3	Objective 3 . . . . .	153
4.3.4	Objective 4 . . . . .	153
4.3.5	Objective 5 . . . . .	153
4.4	Videos . . . . .	154

<b>5 Evaluation</b>	<b>155</b>
5.1 Objectives . . . . .	155
5.1.1 Objective 1 . . . . .	155
5.1.2 Objective 2 . . . . .	155
5.1.3 Objective 3 . . . . .	155
5.1.4 Objective 4 . . . . .	156
5.1.5 Objective 5 . . . . .	156
5.1.6 Objective 6 . . . . .	156
5.2 Client Feedback . . . . .	156
5.3 Conclusion . . . . .	157
<b>A Screenshots</b>	<b>159</b>
<b>B Source Code</b>	<b>163</b>

# Chapter 1

## Analysis

### 1.1 Background

Mr Myslov is a teacher at Tonbridge School, and currently runs the school chess club. Seldomly, a field day event will be held, in which the club convenes together, playing a chess, or another variant, tournament. This year, Mr Myslov has decided to instead, hold a tournament around another board game, namely laser chess, providing a deviation yet retaining the same familiarity of chess. However, multiple physical sets of laser chess have to be purchased for the entire club to play simultaneously, which is difficult due to it no longer being manufactured. Thus, I have proposed a solution by creating a digital version of the game.

#### 1.1.1 Game Description

Laser Chess is an abstract strategy game played between two opponents. The game differs from regular chess, involving a 10x8 playing board arranged in a predefined condition. The aim of the game is to position your pieces such that your laser beam strikes the opponents Pharoah (the equivalent of a king). Pieces include:

1. Pharoah
  - Equivalent to the king in chess
2. Scarab
  - 2 for each colour
  - Contains dual-sided mirrors, capable of reflecting a laser from any direction
  - Can move into an occupied adjacent square, by swapping positions with the piece on it (even with an opponent's piece)
3. Pyramid
  - 7 for each colour
  - Contains a diagonal mirror used to direct the laser
  - The other 3 out of 4 sides are vulnerable from being hit
4. Anubis

- 2 for each colour
- Large pillar with one mirrored side, vulnerable from the other sides

### 5. Sphinx

- 1 for each colour
- Piece from which the laser is shot from
- Cannot be moved

On each turn, a player may move a piece one square in any direction (similar to the king in regular chess), or rotate a piece clockwise or anticlockwise by 90 degrees. After their move, the laser will automatically be fired. It should be noted that a player's own pieces can also be hit by their own laser. As in chess, a three-fold repetition results in a draw. Players may also choose to forfeit or offer a draw.

#### 1.1.2 Current Solutions

Current free implementations of laser chess that are playable online are limited, seemingly only available on <https://laser-chess.com/>.

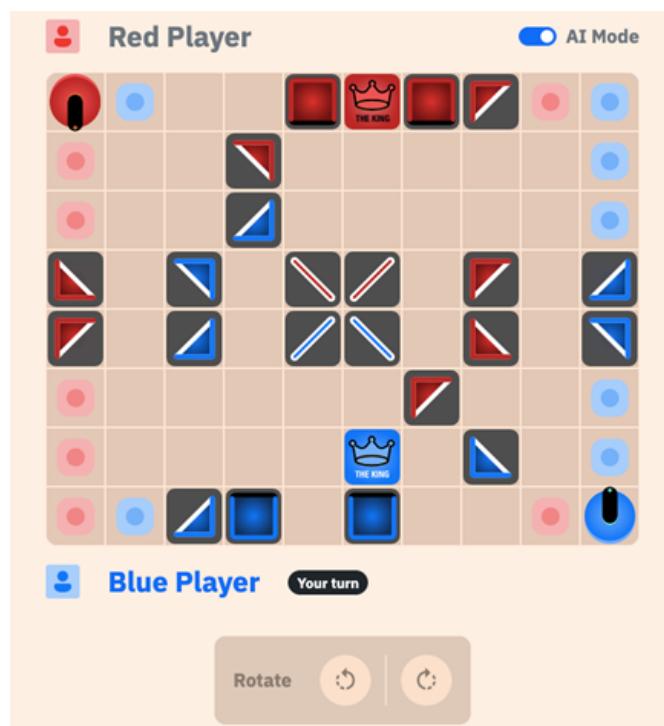


Figure 1.1: Online implementation on laser-chess.com

The game is hosted online and is responsive and visually appealing, with pieces easy to differentiate and displaying their functionality clearly. It also contains a two-player mode for playing between friends, or an option to play against a functional CPU bot. However, the game lacks the following basic functionalities that makes it unsuitable for my client's requests:

- No replay options (going through past moves)
  - A feature to look through previous moves is common in digital board game implementations
  - My client requires this feature as it is an essential tool for learning from past games and to aid in analysing past games
- No option to save and load previous games
  - This QOL feature allows games to be continued on if they cannot be finished in one sitting, and to keep an archive of past games
- Internet connection required
  - My client has specifically requested an offline version as the game will predominantly be played in settings where a connection might not be available (i.e. on a plane or the maths dungeons)
- Unable to change board configuration
  - Most versions of laser chess (i.e. Khet) contain different starting board configurations, each offering a different style of play

Our design will aim to append the missing feature from this website while learning from their commendable UI design.

### 1.1.3 Client Interview

**Q:** Why have you chosen Laser Chess as your request?

**A:** Everyone is familiar with chess, so choosing a game that feels similar, and requires the same thinking process and calculations was important to me. Laser chess fit the requirements, but also provides a different experience in that the new way pieces behave have to be learnt and adapted to. It hopefully will be more fun and a better fit for the boys than other variants such as Othello, as the laser aspects and visuals will keep it stimulating.

*Objectives 1 & 7*

Implementing laser chess in a style similar to normal chess will be important. The client also requests for it to be stimulating, requiring both proper gameplay and custom visuals.

---

**Q:** Have you explored any alternatives?

**A:** I remember Laser Chess was pretty popular years ago, but now it's harder to find a good implementation I can use, since I don't plan on buying multiple physical copies or paid online copies for every student. I have seen a few free websites offering a decent option, but I'm worried that with the terrible connection in the basement will prove unreliable if everybody tries to connect at once. However, I did find the ease-of-use and simple visuals of some websites pleasing, and something that I wish for in the final product as well.

*Objective 6*

The client's limitations call for a digital implementation that plays offline. Taking inspiration from alternatives, a user-friendly GUI is also expected.

---

**Q:** What features are you looking for in the final product?

**A:** I'm looking for most features chess websites like Chess.com or Lichess offer, a smooth playing experience with no noticeable bugs. I'm also expecting other features such as having a functional timer, being able to draw and resign, as these are important considerations in our everyday chess games too. Since this will be a digital game, I think having handy features such as indicators for moves and audio cues will also make it more user-friendly and enjoyable. If not for myself, having the option to play against a computer bot will be appreciated as well, since I'll be able to play during lesson time, or in the case of odd numbers in the tournament. All in all, I'd be happy with a final product that plays Laser Chess, but emulates the playing experience of any chess website well.

*Objectives 1 & 3 & 5*

Gameplay similar to that of popular chess websites is important to our client, introducing the requirement of subtle features such as move highlighting. A CPU bot is also important to our client, who enjoys thinking deeply and analysing chess games, and so will prove important both as a learning tool and as an opponent.

---

**Q:** Are there any additional features that might be helpful for your tournament use-cases?

**A:** Being able to configure the gameplay will be useful for setting custom time-controls for everybody. I also would like to archive games and share everybody's matches with the team, so having the functionality to save games, and to go through previous ones, will be highly requested too. Being able to quickly setup board starting positions or share them will also be useful, as this will allow more variety into the tournament and give the stronger players some more interesting options.

*Objectives 2 & 4*

Saving games and customising them is a big logistical priority for a tournament, as this will provide the means to record games and for opponents to all agree on the starting conditions, depending on the circumstances of the tournament.

## 1.2 Objectives

### 1.2.1 Client Objectives

The following objectives should be met to satisfy my clients' requirements:

1. All laser chess game logic should be properly implemented
  - All pieces should be display correct behaviour (i.e reflect the laser correctly)
  - Option to rotate laser chess pieces should be implemented
  - Option to move laser chess pieces should be implemented
  - Game should allocate alternating player's turns
  - Travel path of laser should be correctly implemented
  - Pieces should be automatically detected and eliminated when hit by the laser
  - Game should automatically detect when a player has lost or won
  - Three-fold repetition should be automatically detected
2. Save or load game options should be implemented

- Games should be encoded into FEN string format
- Games can be saved locally into external files
- User should be able to scroll through previous games
- User should be able to scroll through previous moves
- Game should display information relevant to each game, and display moves correctly

3. Other board game requirements should be implemented

- Draws should be made an option
- Resigning should be made an option
- Timer displaying time left for each player should be displayed
- Time logic should be implemented, pausing when it is the opponent's turn, forfeiting players who run out of time

4. Game settings and config should be customisable

- User should be able to select board colours
- User should be able to play against another player or CPU
- User should be able to customise timer and duration
- User should be able to customise starting board layout and colour

5. Game UI should improve player experience

- Drag and drop should be implemented
- Selected pieces should be clearly marked with an indicator
- Indicator showing available squares to move to when clicking on a piece
- Destroying a piece should display a visual and audio cue
- Captured pieces should be displayed for each player
- Status message should display current status of the game (whose turn it is, move a piece, game won etc.)

6. GUI design should be functional and display concise information

- GUI should always remain responsive throughout the running of the program
- Application should be divided into separate sections with their own menus and functionality (e.g. title page, settings)
- Navigation buttons (e.g. return to menu) should concisely display their functionality
- UI should contain exit and help buttons
- UI should be designed for clarity in mind and visually pleasing
- Application should be responsive, draggable and resizable

### 1.2.2 Other User Considerations

Although my current primary client is Mr Myslov, I aim to make my program shareable and accessible, so other parties who would like to try laser chess can access a comprehensive implementation of the game, which currently is not readily available online. Additionally, the code should be concise and well commented, complemented by proper documentation, so other parties can edit and implement additional features such as multiplayer to their own liking.

## 1.3 Research

Before proceeding with the actual implementation of the game, I will have to conduct research to plan out the fundamental architecture of the game. Reading on available information online, prior research will prevent me from committing unnecessary time to potentially inadequate ideas or code. I will consider the following areas: board representation, CPU techniques and GUI framework.

### 1.3.1 Board Representation

Board representation is the use of a data structure to represent the state of all pieces on the chessboard, and the state of the game itself, at any moment. It is the foundation on which other aspects such as move generation and the evaluation function are built upon, with different methods of implementation having their own advantages and disadvantages on simplicity, execution efficiency and memory footprint. Every board representation can be classified into two categories: piece-centric or square-centric. Piece-centric representations involve keeping track of all pieces on the board and their associated position. Conversely, square-centric representations track every available square, and if it is empty or occupied by a piece. The following are descriptions of various board representations with their respective pros and cons.

#### Square list

Square list, a square-centric representation, involves the encoding of each square residing in a separately addressable memory element, usually in the form of an 8x8 two-dimensional array. Each array element would identify which piece, if any, occupies the given square. A common piece encoding could involve using the integers 1 for a pawn, 2 for knight, 3 for bishop, and + and - for white and black respectively (e.g. a white knight would be +2). This representation is easy to understand and implement, and has easy support for multiple chess variants with different board sizes. However, it is computationally inefficient as nested loop commands must be used in frequently called functions, such as finding a piece location. Move generation is also problematic, as each move must be checked to ensure that it does not wrap around the edge of the board.

#### 0x88

0x88, another square-centric representation, is an 128-byte one-dimensional array, equal to the size of two adjacent chessboards. Each square is represented by an integer, with two nibbles used to represent the rank and file of the respective square. For example, the 8-integer 0x42 (0100 0010) would represent the square (4, 2) in zero-based numbering. The advantage of 0x88 is that faster bitwise operations are used for computing piece transformations. For example, add 16 to the current square number to move to the square on the row above, or add 1 to move to the next column. Moreover, 0x88 allows for efficient off-the-board detection. Every valid square number is under 0x88 in hex (0111 0111), and by performing a bitwise AND operation between the square number and 0x88 (1000 1000), the destination square can be shown to be invalid if the result is non-zero (i.e. contains 1 on 4th or 8th bit).

#### Bitboards

Bitboards, a piece-centric representation, are finite sets of 64 elements, one bit per square. To represent the game, one bitboard is needed for each piece-type and colour, stored as an array of bitboards as part of a position object. For example, a player could have a bitboard

for white pawns, where a positive bit indicates the presence of the pawn. Bitboards are fast to incrementally update, such as flipping bits at the source and destination positions for a moved piece. Moreover, bitmaps representing static information, such as spaces attacked by each piece type, can be pre-calculated, and retrieved with a single memory fetch at a later time. Additionally, bitboards can operate on all squares in parallel using bitwise operations, notably, a 64-bit CPU can perform all operations on a 64-bit bitboard in one cycle. Bitboards are therefore far more execution efficient than other board representations. However, bitboards are memory-intensive and may be sparse, sometimes only containing a single bit in 64. They require more source code, and are problematic for devices with a limited number of process registers or processor instruction cache.

### 1.3.2 CPU techniques

#### Minimax

Minimax is a backtracking algorithm that evaluates the best move given a certain depth, assuming optimal play by both players. A game tree of possible moves is formulated, until the leaf node reaches a specified depth. Using a heuristic evaluation function, minimax recursively assigns each node an evaluation based on the following rules:

- If the node represents a white move, the node's evaluation is the *maximum* of the evaluation of its children
- If the node represents a black move, the node's evaluation is the *minimum* of the evaluation of its children

Thus, the algorithm *minimizes* the loss involved when the opponent chooses the move that gives *maximum* loss.

Several additional techniques can be implemented to improve upon minimax. For example, transposition tables are large hash tables storing information about previously reached positions and their evaluation. If the same position is reached via a different sequence of moves, the cached evaluation can be retrieved from the table instead of evaluating each child node, greatly reducing the search space of the game tree. Another, such as alpha-beta pruning can be stacked and applied, which eliminates the need to search large portions of the game tree, thereby significantly reducing the computational time.

#### Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) involves playouts, where games are played to its end by selecting random moves. The result of each playout is then backpropagated up the game tree, updating the weight of nodes visited during the playout, meaning the algorithm successively improves at accurately estimating the values of the most promising moves. MCTS periodically evaluates alternatives to the currently perceived optimal move, and could thereby discover a better, otherwise overlooked, path. Another benefit is that it does not require an explicit evaluation function, as it relies on statistical sampling as opposed to developed theory on the game state. Additionally, MCTS is scalable and may be parallelized, making it suitable for distributed computing or multi-core architectures. However, the rate of tree growth is exponential, requiring huge amounts of memory. In addition, MCTS requires many iterations to be able to reliably decide the most efficient path.

### 1.3.3 GUI framework

#### Pygame

Pygame is an open-source Python module geared for game development. It offers abundant yet simple APIs for drawing sprites and game objects on a screen-canvas, managing user input, audio et cetera. It also has good documentation, an extensive community, and receives regular updates through its community edition. Although it has greater customizability in drawing custom bitmap graphics and control over the mainloop, it lacks built-in support for UI elements such as buttons and sliders, requiring custom implementation. Moreover, it is less efficient, using 2D pixel arrays and the RAM instead of utilising the GPU for batch rendering, being single-threaded, and running on an interpreted language.

#### PyQt

PyQt is the Python binding for Qt, a cross-platform C++ GUI framework. PyQt contains an extensive set of documentation online, complemented by the documentation and forums for its C++ counterpart. Unlike Pygame, PyQt contains many widgets for common UI elements, and support for concurrency within the framework. Another advantage in using PyQt is its development ecosystem, with peripheral applications such as Qt Designer for layouts, QML for user interfaces, and QSS for styling. Although it is not open-source, containing a commercial licensing plan, I have no plans to commercialize the program, and can therefore utilise the open-source license.

#### OpenGL

Python contains multiple bindings for OpenGL, such as PyOpenGL and ModernGL. Being a widely used standard, OpenGL has the best documentation and support. It also boasts the highest efficiency, designed to be implemented using hardware acceleration through the GPU. However, its main disadvantage is the required complexity compared to the previous frameworks, being primarily a graphical API and not for developing full programs.

## 1.4 Proposed Solution

### 1.4.1 Language

The two main options regarding programming language choice, and their pros and cons, are as listed:

---

		Python
Pros	Cons	

---

- Versatile and intuitive, uses simple syntax and dynamic typing
  - Supports both object-oriented and procedural programming
  - Rich ecosystem of third-party modules and libraries
  - Interpreted language, good for portability and easy debugging
  - Slow at runtime
  - High memory consumption
- 

		Javascript
Pros	Cons	
<ul style="list-style-type: none"> <li>• Generally faster runtime than Python</li> <li>• Simple, dynamically typed and automatic memory management</li> <li>• Versatile, easy integration with both server-side and front-end</li> <li>• Extensive third-party modules</li> <li>• Also supports object-oriented programming</li> </ul>	<ul style="list-style-type: none"> <li>• Mainly focused for web development</li> <li>• Comprehensive knowledge of external frameworks (i.e. Electron) needed for developing desktop applications</li> </ul>	

I have chosen Python as the programming language for this project. This is mainly due to its extensive third-party modules and libraries available. Python also provides many different GUI frameworks for desktop applications, whereas options are limited for JavaScript due to its focus on web applications. Moreover, the amount of resources and documentation online will prove invaluable for the development process.

Although Python generally has worse performance than JavaScript, speed and memory efficiency are not primary objectives in my project, and should not affect the final program. Therefore, I have prioritised Python's simpler syntax over JavaScript's speed. Being familiar with Python will also allow me to divert more time for development instead of researching new concepts or fixing unfamiliar bugs.

#### 1.4.2 Development Environment

A good development environment improves developer experiences, with features such as auto-indentation and auto-bracket completion for quicker coding. The main development environments under consideration are: Visual Studio Code (VS Code), PyCharm and Sublime Text. I have decided to use VS Code due to its greater library of extensions over Sublime, and its more

user-friendly implementation of features such as version control and GitHub integration. Moreover, VS Code contains many handy features that will speed up the development process, such as its built-in debugging features. Although PyCharm is an extensive IDE, its default features can be supplemented by VS Code extensions. Additionally, VS Code is more lightweight and customizable, and contains vast documentation online.

### 1.4.3 Source Control

A Source Control Tool automates the process of tracking and managing changes in source code. A good source control tool will be essential for my project. It provides the benefits of: protecting the code from human errors (i.e. accidental deletion), enabling easy code experimentation on a clone created through branching from the main project, and by tracking changes through the code history, enabling easier debugging and rollbacks. For my project, I have chosen Git as my version control tool, as it is open-source and provides a more user-friendly interface and documentation over alternatives such as Azure DevOps, and contains sufficient functionality for a small project like mine.

### 1.4.4 Techniques

I have decided on employing the following techniques, based on the pros and cons outlined in the research section above.

#### Board representation

I have chosen to use a bitboard representation for my game. The main consideration was computational efficiency, as a smooth playing experience should be ensured regardless of device used. Bitboards allow for parallel bitwise operations, especially as most modern devices nowadays run on 64-bit architecture CPUs. With bitboards being the mainstream implementation, documentation should also be plentiful.

#### CPU techniques

I have chosen minimax as my searching algorithm. This is due to its relatively simplistic implementation and evaluation accuracy. Additionally, Monte-Carlo Tree Search is computationally intensive, with a high memory requirement and time needed to run with a sufficient number of simulations, which I do not have.

#### GUI framework

I have chosen Pygame as my main GUI framework. This is due to its increased flexibility, in creating custom art and widgets compared to PyQt's defined toolset, which is tailored towards building commercial office applications. Although Pygame contains more overhead and boilerplate code to create standard functionality, I believe that the increased control is worth it for a custom game such as laser chess, which requires dynamic rendering of elements such as the laser beam.

I will also integrate Pygame together with ModernGL, using the convenient APIs in for handling user input and sprite drawing, together with the speed of OpenGL to draw shaders and any other effect overlays.

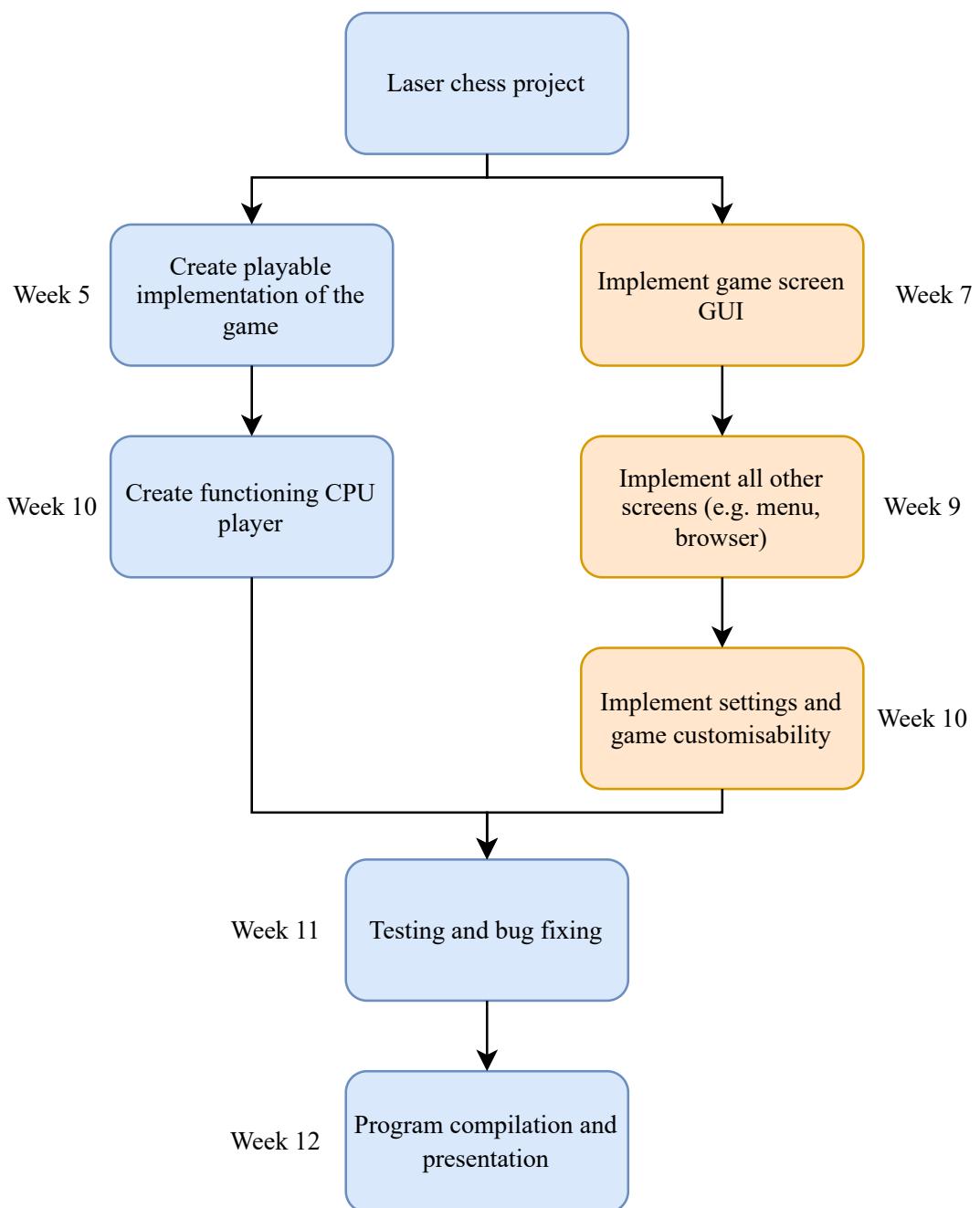
## 1.5 Limitations

I have agreed with my client that due to the multiple versions of Laser Chess that exist online, together with a lack of regulation, an implementation that adheres to the general rules of Laser Chess, and not strictly to a specific version, is acceptable.

Moreover, due to the time constraints on both my schedules for exams and for the date of the tournament, the game only has to be presented in a functional state, and not polished for release, with extra work such as porting to a wide range of OS systems.

## 1.6 Critical Path Design

In order to meet my client's requirement of releasing the game before the next field day, I have given myself a time limit of 12 weeks to develop my game, and have created the following critical path diagram to help me adhere to completing every milestone within the time limit.



# Chapter 2

## Design

### 2.1 System Architecture

In this section, I will lay out the overall logic, and an overview of the steps involved in running my program. By decomposing the program into individual abstracted stages, I can focus on the workings and functionality of each section individually, which makes documenting and coding each section easier. I have also included a flowchart to illustrate the logic of each screen of the program.

I will also create an abstracted GUI prototype in order to showcase the general functionality of the user experience, while acting as a reference for further stages of graphical development. It will consist of individually drawn screens for each stage of the program, as shown in the top-level overview. The elements and layout of each screen are also documented below.

The following is a top-level overview of the logic of the program:

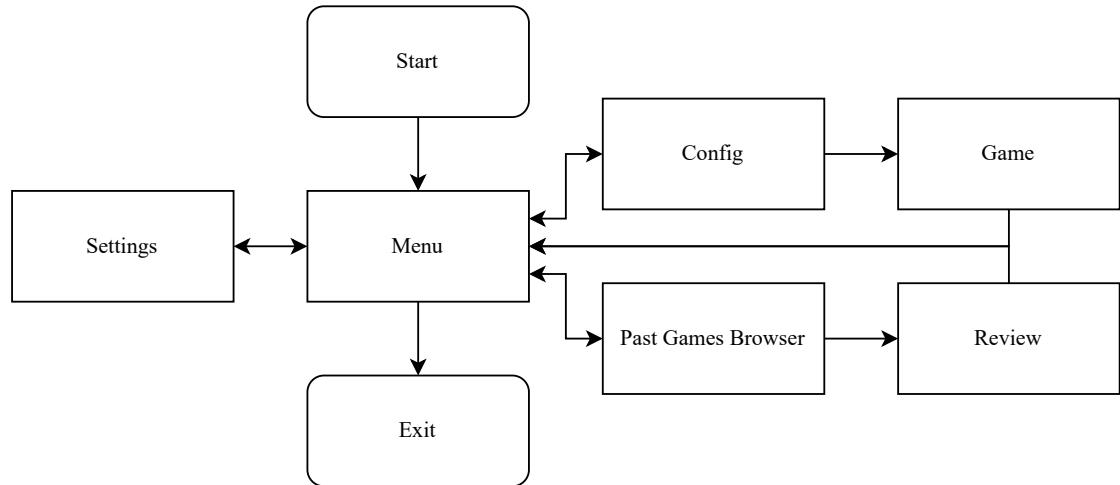


Figure 2.1: Flowchart for Program Overview

### 2.1.1 Main Menu



Figure 2.2: Main Menu screen prototype

The main menu will be the first screen to be displayed, providing access to different stages of the game. The GUI should be simple yet effective, containing clearly-labelled buttons for the user to navigate to different parts of the game.

### 2.1.2 Settings

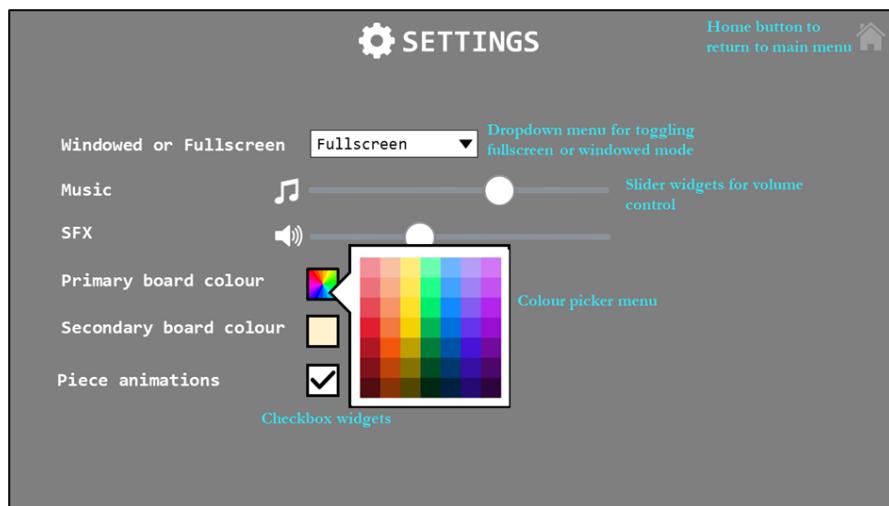


Figure 2.3: Settings screen prototype

The settings menu allows for the user to customise settings related to the program as a whole. The settings will be changed via GUI elements such as buttons and sliders, offering the ability

to customize display mode, volume, board colour etc. Changes to settings will be stored in an intermediate code class, then stored externally into a JSON file. Game settings will instead be changed in the Config screen.

The setting screen should provide a user-friendly interface for changing the program settings intuitively; I have therefore selected appropriate GUI widgets for each setting:

- Windowed or Fullscreen - Drop-down list for selecting between pre-defined options
- Music and SFX - Slider for selecting audio volume, a continuous value
- Board colour - Colour grid for the provision of multiple pre-selected colours
- Piece animation - Checkbox for toggling between on or off

Additionally, each screen is provided with a home button icon on the top right (except the main menu), as a shortcut to return to the main menu.

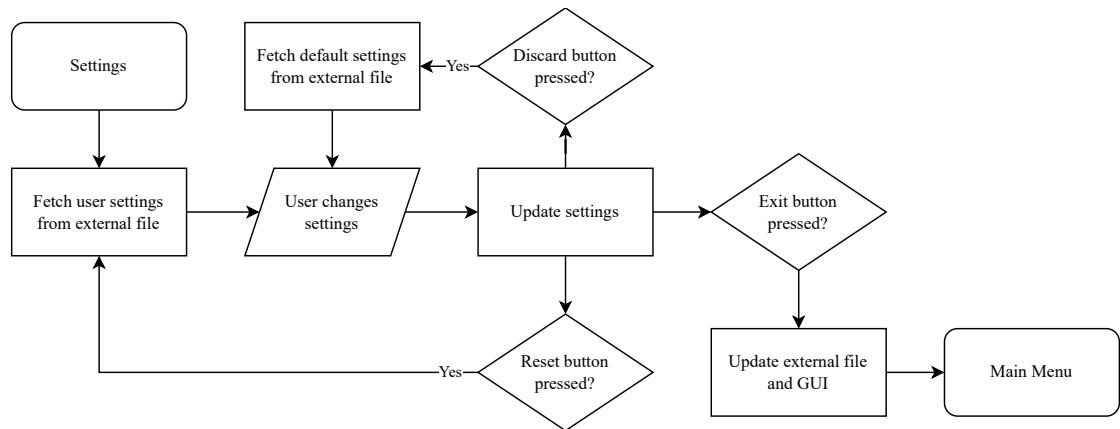


Figure 2.4: Flowchart for Settings

### 2.1.3 Past Games Browser



Figure 2.5: Browser screen prototype

The Past Games Browser menu displays a list of previously played games to be replayed. When selecting a game, the replay will render out the saved FEN string into a board position identical to the one played previously, except the user is limited to replaying back and forth between recorded moves. The menu also offers the functionality of sorting games in terms of time, game length etc.

For the GUI, previous games will be displayed on a strip, scrolled through by a horizontal slider. Information about the game will be displayed for each instance, along with the option to copy the FEN string to be stored locally or to be entered into the Review screen. When choosing a past game, a green border will appear to show the current selection, and double clicking enters the user into the full replay mode. While replaying the game, the GUI will appear identical to an actual game. However, the user will be limited to scrolling throughout the moves via the left and right arrow keys.

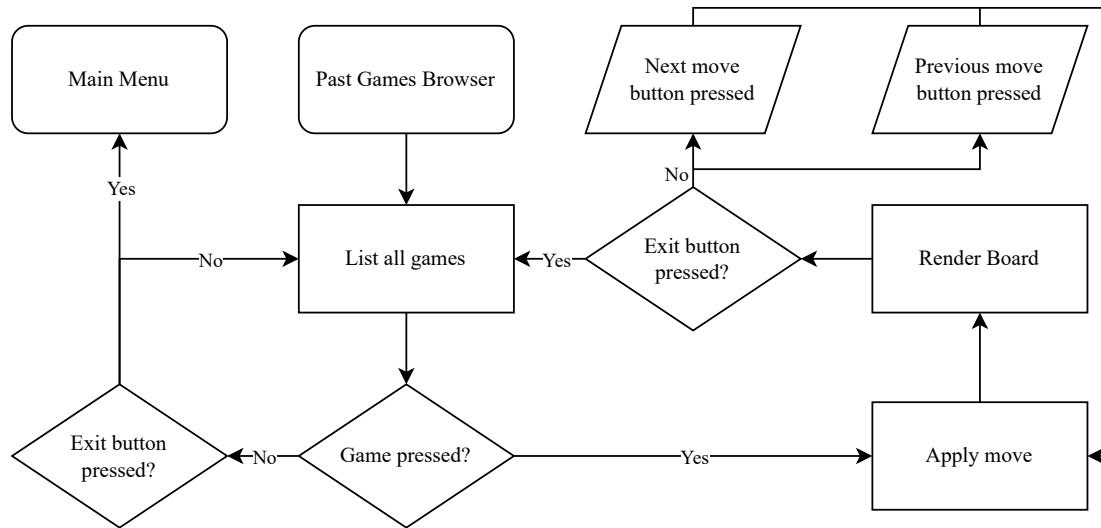


Figure 2.6: Flowchart for Browser

#### 2.1.4 Config

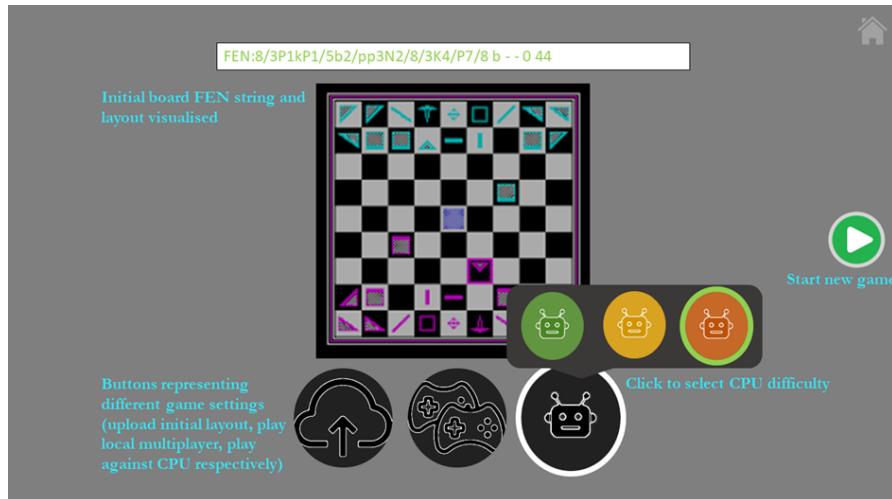


Figure 2.7: Config screen prototype

The config screen comes prior to the actual gameplay screen. Here, the player will be able to change game settings such as toggling the CPU player, time duration, playing as white or black etc.

The config menu is loaded with the default starting position. However, players may enter their own FEN string as an initial position, with the central board updating responsively to give a visual representation of the layout. Players are presented with the additional options to play against a friend, or against a CPU, which displays a drop-down list when pressed to select the CPU difficulty.

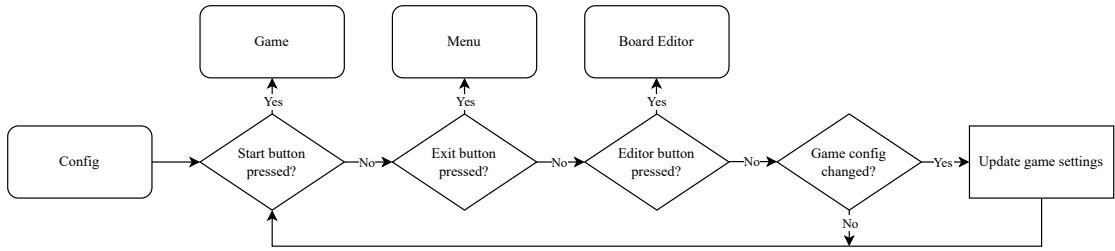


Figure 2.8: Flowchart for Config

### 2.1.5 Game



Figure 2.9: Game screen prototype

During the game, handling of the game logic, such as calculating player turn, calculating CPU moves or laser trajectory, will be computed by the program internally, rendering the updated GUI accordingly in a responsive manner to provide a seamless user experience.

In the game screen, the board is positioned centrally on the screen, surrounded by accompanying widgets displaying information on the current state of the game. The main elements include:

- Status text - displays information on the game state and prompts for each player move
- Rotation buttons - allows each player to rotate the selected piece by 90° for their move
- Timer - displays available time left for each player
- Draw and forfeit buttons - for the named functionalities, confirmed by pressing twice
- Piece display - displays material captured from the opponent for each player

Additionally, the current selected piece will be highlighted, and the available squares to move to will also contain a circular visual cue. Pieces will either be moved by clicking the

target square, or via a drag-and-drop mechanism, accompanied by responsive audio cues. These implementations aim to improve user-friendliness and intuitiveness of the program.

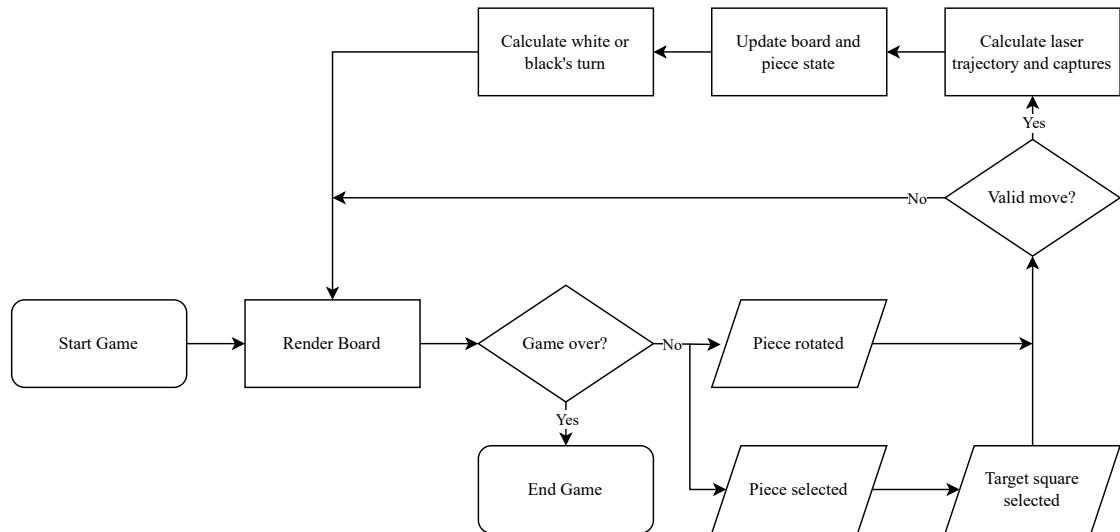


Figure 2.10: Flowchart for Game

### 2.1.6 Board Editor

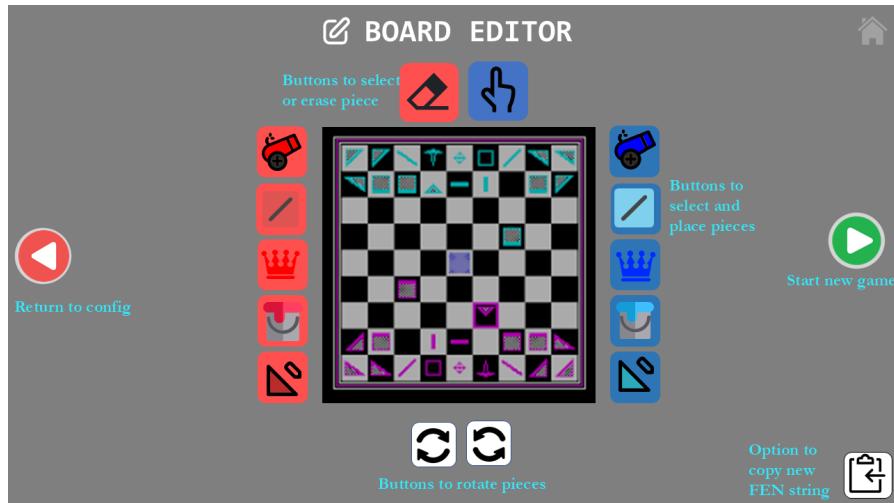


Figure 2.11: Editor screen prototype

The editor screen is used to configure the starting position of the board. Controls should include the ability to place all piece types of either colour, to erase pieces, and easy board manipulation shortcuts such as dragging pieces or emptying the board.

For the GUI, the buttons should clearly represent their functionality, through the use of icons and appropriate colouring (e.g. red for delete).

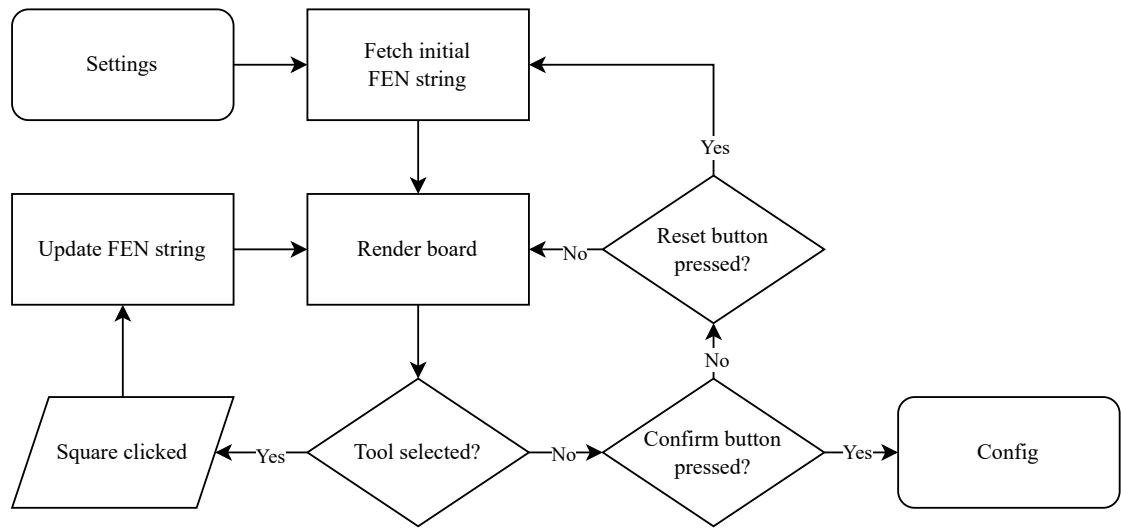


Figure 2.12: Flowchart for board editor

## 2.2 Algorithms and Techniques

### 2.2.1 Minimax

Minimax is a backtracking algorithm commonly used in zero-sum games used to determine the score according to an evaluation function, after a certain number of perfect moves. Minimax aims to minimize the maximum advantage possible for the opponent, thereby minimizing a player's possible loss in a worst-case scenario. It is implemented using a recursive depth-first search, alternating between minimizing and maximizing the player's advantage in each recursive call.

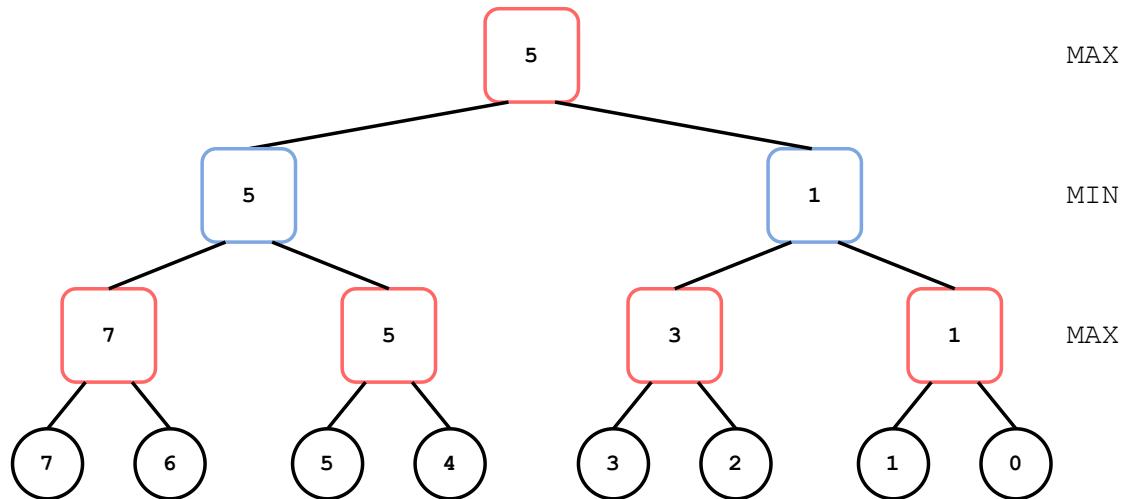


Figure 2.13: Example minimax tree

For the example minimax tree show in Figure 2.13, starting from the bottom leaf node

evaluations, the maximising player would choose the highest values (7, 5, 3, 1). From those values, the minimizing player would choose the lowest values (5, 1). The final value chosen by the maximum player would therefore be the highest of the two, 5.

Implementation in the form of pseudocode is shown below:

---

**Algorithm 1** Minimax pseudocode
 

---

```

function MINIMAX(node, depth, maximisingPlayer)
  if depth = 0 OR node equals game over then
    return EVALUATE
  end if

  if maximisingPlayer then
    value ← −∞
    for child of node do
      value ← MAX(value, MINIMAX(child, depth − 1, false))
    end for
    return value
  else
    value ← +∞
    for child of node do
      value ← MIN(value, MINIMAX(child, depth − 1, true))
    end for
    return value
  end if
end function
```

---

## 2.2.2 Minimax improvements

### Alpha-beta pruning

Alpha-beta pruning is a search algorithm that aims to decrease the number of nodes evaluated by the minimax algorithm. Alpha-beta pruning stops evaluating a move in the game tree when one refutation is found in its child nodes, proving the node to be worse than previously-examined alternatives. It does this without any potential of pruning away a better move. The algorithm maintains two values: alpha and beta. Alpha ( $\alpha$ ), the upper bound, is the highest value that the maximising player is guaranteed of; Beta ( $\beta$ ), the lower bound, is the lowest value that the minimizing player is guaranteed of. If the condition  $\alpha \geq \beta$  for a node being evaluated, the evaluation process halts and its remaining children nodes are ‘pruned’.

This is shown in the following maximising example:

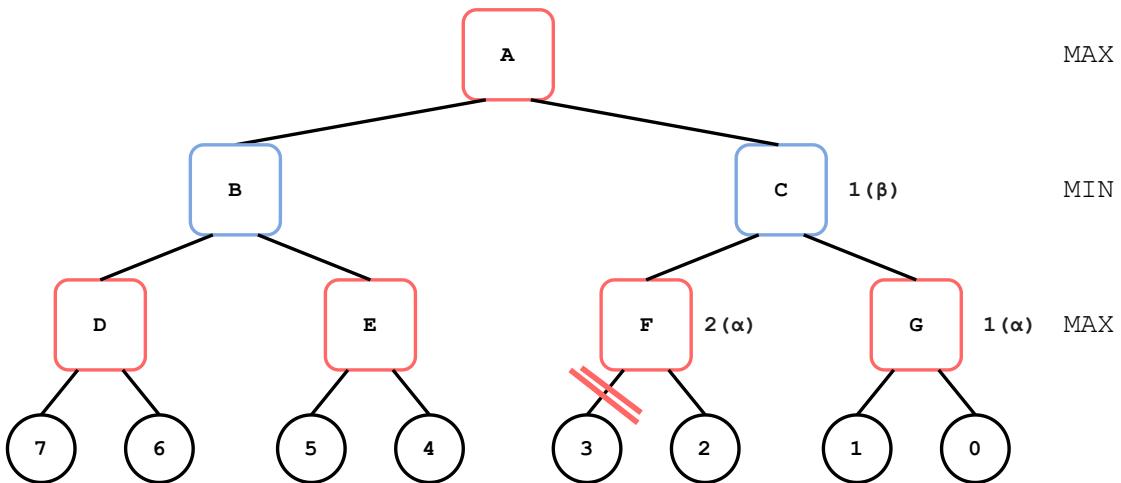


Figure 2.14: Example minimax tree with alpha-beta pruning

Since minimax is a depth-first search algorithm, nodes  $C$  and  $G$  and their  $\alpha$  and  $\beta$  have already been searched. Next, at node  $F$ , the current  $\alpha$  and  $\beta$  are  $-\infty$  and 1 respectively, since the  $\beta$  is passed down from node  $C$ . Searching the first leaf node, the  $\alpha$  subsequently becomes  $\alpha = \max(-\infty, 2)$ . This means that the maximising player at this depth is already guaranteed an evaluation of 2 or greater. Since we know that the minimising player at the depth above is guaranteed a value of 1, there is no point in continuing to search node  $F$ , a node that returns a value of 2 or greater. Hence at node  $F$ , where  $\alpha \geq \beta$ , the branches are pruned.

Alpha-beta pruning therefore prunes insignificant nodes by maintain an upper bound  $\alpha$  and lower bound  $\beta$ . This is an essential optimization as a simple minimax tree increases exponentially in size with each depth ( $O(b^d)$ , with branching factor  $b$  and  $d$  ply depth), and alpha-beta reduces this and the associated computational time considerably.

The pseudocode implementation is shown below:

---

**Algorithm 2** Minimax with alpha-beta pruning pseudocode

---

```

function MINIMAX(node, depth,  $\alpha$ ,  $\beta$ , maximisingPlayer)
    if depth = 0 OR node equals game over then
        return EVALUATE
    end if

    if maximisingPlayer then
        value  $\leftarrow -\infty$ 
        for child of node do
            value  $\leftarrow \text{MAX}(\text{value}, \text{MINIMAX}(\text{child}, \text{depth} - 1, \alpha, \beta, \text{false}))$ 
            if value >  $\beta$  then break
            end if
             $\alpha \leftarrow \text{MAX}(\alpha, \text{value})$ 
        end for
        return value
    else
        value  $\leftarrow +\infty$ 
        for child of node do
            value  $\leftarrow \text{MIN}(\text{value}, \text{MINIMAX}(\text{child}, \text{depth} - 1, \alpha, \beta, \text{true}))$ 
            if value <  $\alpha$  then break
            end if
             $\beta \leftarrow \text{MIN}(\beta, \text{value})$ 
        end for
        return value
    end if
end function

```

---

**Transposition Tables & Zobrist Hashing**

Transition tables, a memoisation technique, again greatly reduces the number of moves searched. During a brute-force minimax search with a depth greater than 1, the same positions may be searched multiple times, as the same position can be reached from different sequences of moves. A transposition table caches these same positions (transpositions), along with its associated evaluations, meaning commonly reached positions are not unnecessarily re-searched.

Flags and depth are also stored alongside the evaluation. Depth is required as if the current search comes across a cached position with an evaluation calculated at a lower depth than the current search, the evaluation may be inaccurate. Flags are required for dealing with the uncertainty involved with alpha-beta pruning, and can be any of the following three.

**Exact** flag is used when a node is fully searched without pruning, and the stored and fetched evaluation is accurate.

**Lower** flag is stored when a node receives an evaluation greater than the  $\beta$ , and is subsequently pruned, meaning that the true evaluation could be higher than the value stored. We are thus storing the  $\alpha$  and not an exact value. Thus, when we fetch the cached value, we have to recheck if this value is greater than  $\beta$ . If so, we return the value and this branch is pruned (fail high); If not, nothing is returned, and the exact evaluation is calculated.

**Upper** flag is stored when a node receives an evaluation smaller than the  $\alpha$ , and is subsequently pruned, meaning that the true evaluation could be lower than the value stored. Similarly, when we fetch the cached value, we have to recheck if this value is lower than  $\alpha$ . Again, the current branch is pruned if so (fail low), and an exact evaluation is calculated if not.

The pseudocode implementation for transposition tables is shown below:

---

**Algorithm 3** Minimax with transposition table pseudocode

---

```

function MINIMAX(node, depth,  $\alpha$ ,  $\beta$ , maximisingPlayer)
    hash_key  $\leftarrow$  HASH(node)
    entry  $\leftarrow$  GETENTRY(hash_key)

    if entry.hash_key = hash_key AND entry.hash_key  $\geq$  depth then
        if entry.hash_key = EXACT then
            return entry.value
        else if entry.hash_key = LOWER then
             $\alpha \leftarrow \text{MAX}(\alpha, \text{entry.value})$ 
        else if entry.hash_key = UPPER then
             $\beta \leftarrow \text{MIN}(\beta, \text{entry.value})$ 
        end if
        if  $\alpha \geq \beta$  then
            return entry.value
        end if
    end if

    ...normal minimax...

    entry.value  $\leftarrow$  value
    entry.depth  $\leftarrow$  depth
    if value  $\leq \alpha$  then
        entry.flag  $\leftarrow$  UPPER
    else if value  $\geq \beta$  then
        entry.flag  $\leftarrow$  LOWER
    else
        entry.flag  $\leftarrow$  EXACT
    end if

    return value
end function

```

---

The current board position will be used as the index for a transposition table entry. To convert our board state and bitboards into a valid index, Zobrist hashing may be used. For every square on the chessboard, a random integer is assigned to every piece type (12 in our case, 6 piece type, times 2 for both colours). To initialise a hash, the random integer associated with the piece on a specific square undergoes a XOR operation with the existing hash. The hash is incrementally update with XOR operations every move, instead of being recalculated from scratch improving computational efficiency. Using XOR operations also allows moves to be reversed, proving useful for the functionality to scroll through previous moves. A Zobrist hash is also a better candidate than FEN strings in checking for threefold-repetition, as they are less

intensive to calculate for every move.

The pseudocode implementation for Zobrist hashing is shown below:

---

**Algorithm 4** Zobrist hashing pseudocode
 

---

*RANDOMINTS represents a pre-initialised array of random integers for each piece type for each square*

```

function HASH _ BOARD(board)
    hash ← 0
    for each square on board do
        if square is not empty then
            hash ⊕ RANDOMINTS[square][piece on square]
        end if
    end for
    return hash
end function

function UPDATEHASH(hash, move)
    hash ⊕ RANDOMINTS[source square][piece]
    hash ⊕ RANDOMINTS[destination square][piece]
    if red to move then
        hash ⊕ hash for red to move ▷ Hash needed for move colour, as two identical positions
        are different if the colour to move is different
    end if
    return hash
end function
  
```

---

### Iterative Deepening

Iterative deepening builds upon the previous alpha-beta and caching improvements. A search is initiated at a depth of one ply, which upon finishing, another starts at depth two, three, and increases until the max depth is reached or time allocated is up. Although this means that more nodes are searched, the improvements come from the fact that the best move (PV-Move) found by a lower depth search, can be used as the first move searched on the next higher depth search. This increases the chance of pruning, reducing the net number of nodes searched, and also provides a 'fallback' move if a higher depth search is interrupted.

#### 2.2.3 Board Representation

##### FEN string

Forsyth-Edwards Notation (FEN) notation provides all information on a particular position in a chess game. I intend to implement methods parsing and generating FEN strings in my program, in order to load desired starting positions and save games for later play. Deviating from the classic 6-part format, a custom FEN string format will be required for our laser chess game, accommodating its different rules from normal chess.

Our custom format implementation is show by the example below:

sc3ncfancpb2/2pc7/3Pd7/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa

r

Our FEN string format contains two parts, denoted by the space between them:

- Part 1: Describes the location of each piece. The construction of this part is defined by the following rules:
  - The board is read from top-left to bottom-right, row by row
  - A number represents the number of empty squares before the next piece
  - A capital letter represents a blue piece, and a lowercase letter represents a red piece
  - The letters *F*, *R*, *P*, *N*, *S* stand for the pieces Pharaoh, Scarab, Pyramid, Anubis and Sphinx respectively
  - Each piece letter is followed by the lowercase letters *a*, *b*, *c* or *d*, representing a 0°, 90°, 180° and 270° degree rotation respectively
- Part 2: States the active colour, *b* means blue to move, *r* means red to move

Having inputted the desired FEN string board configuration in the config menu, the bitboards for each piece will be initialised with the following functions:

---

**Algorithm 5** FEN string pseudocode
 

---

```

function PARSE_FEN_STRING(fen_string, board)
  part_1, part_2 ← SPLIT(fen_string)
  rank ← 8
  file ← 0

  for character in part_1 do
    square ← rank × 8 + file
    if character is alphabetic then
      if character is lower then
        board.bitboards[red][character] | 1 << character
      else
        board.bitboards[blue][character] | 1 << character
      end if
    else if character is numeric then
      file ← file + character
    else if character is / then
      rank ← rank - 1
      file ← file + 1
    else
      file ← file + 1
    end if

    if part_2 is b then
      board.active_colour ← b
    else
      board.active_colour ← r
    end if
  end for
end function
  
```

---

The function first processes every piece and corresponding square in the FEN string, modifying each piece bitboard using a bitwise OR operator, with a 1 shifted over to the correctly occupied square using a Left-Shift operator. For the second part, the active colour property of the board class is initialised to the correct player.

### Bitboards

Bitboards are an array of bits representing a position or state of a board game. Multiple bitboards are used with each representing a different property of the game (e.g. scarab position and scarab rotation), and can be masked together or transformed to answer queries about positions. Bitboards offer an efficient board representation, its performance primarily arising from the speed of parallel bitwise operations used to transform bitboards. To map each board square to a bit in each number, we will assign each square from left to right, with the least significant bit (LSB) assigned to the bottom-left square (A1), and the most significant bit (MSB) to the top-right square (J8).

<b>8</b>	70	71	72	73	74	75	76	77	78	79
<b>7</b>	60	61	62	63	64	65	66	67	68	69
<b>6</b>	50	51	52	53	54	55	56	57	58	59
<b>5</b>	40	41	42	43	44	45	46	47	48	49
<b>4</b>	30	31	32	33	34	35	36	37	38	39
<b>3</b>	20	21	22	23	24	25	26	27	28	29
<b>2</b>	10	11	12	13	14	15	16	17	18	19
<b>1</b>	0	1	2	3	4	5	6	7	8	9
	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>g</b>	<b>h</b>	<b>j</b>	<b>k</b>

Figure 2.15: Square to bit position mapping

Firstly, we need to initialise each bitboard and place 1s in the correct squares occupied by pieces. This is achieved whilst parsing the FEN-string, as shown in Algorithm 5. Secondly, we should implement an approach to calculate possible moves using our computed bitboards. We can begin by producing a bitboard containing the locations of all pieces, achieved through combining every piece bitboard with bitwise OR operations:

```
all_pieces_bitboard = white_pharaoh_bitboard | black_pharaoh_bitboard |
                     white_scarab_bitboard ...
```

Now, we can utilize this aggregated bitboard to calculate possible positional moves for each piece. For each piece, we can shift the entire bitboard to an adjacent target square (since every piece can only move one adjacent square per turn), and perform a bitwise AND operator with the bitboard containing all pieces, to determine if the target square is already occupied by an existing piece. For example, if we want to compute if the square to the left of our selected piece

is available to move to, we will first shift every bit right (as the lowest square index is the LSB on the right, see diagram above), as demonstrated in the following 5x5 example:

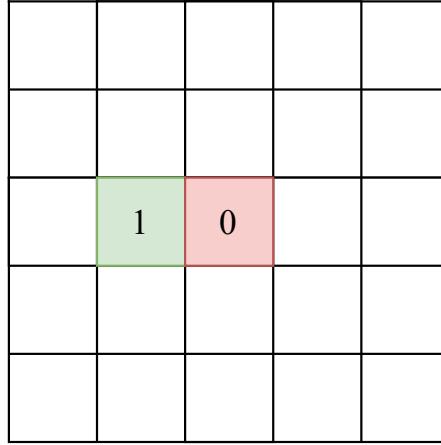


Figure 2.16: `shifted_bitboard = piece_bitboard >> 1`

Where green represents the target square shifted into, and orange where the piece used to be. We can then perform a bitwise AND operation with the complement of the all pieces bitboard, where a square with a result of 1 represents an available target square to move to.

```
available_squares_right = (piece_bitboard >> 1) & ~all_pieces_bitboard
```

However, if the piece is on the leftmost A file, and is shifted to the right, it will be teleported onto the J file on the rank below, which is not a valid move. To prevent these erroneous moves for pieces on the edge of the board, we can utilise an A file mask to mask away any valid moves, as demonstrated below:

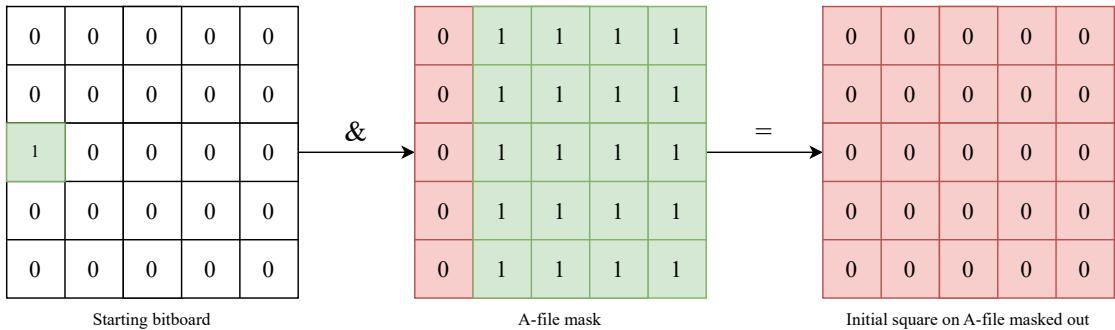


Figure 2.17: A-file mask example

This approach uses the logic that a piece on the A file can never move to a square on the left. Therefore, when calculating if a piece can move to a square on the left, we apply a bitwise AND operator with a mask where every square on the A file is 0; If a piece was on the A file, it will become 0, leaving no possible target squares to move to. The same approach can be mirrored for the far-right J file.

In theory, we do not need to implement the same solution for clipping in regards to ranks, as they are removed automatically by overflow or underflow when shifting bits too far. Our final function to calculate valid moves combines all the logic above: Shifting the selected piece in all 9 adjacent directions by their corresponding bits, masking away pieces trying to move into the edge of the board, combining them with a bitwise OR operator, and finally masking it with the all pieces bitboard to detect which squares are not currently occupied:

---

**Algorithm 6** Finding valid moves pseudocode

---

```

function FIND_VALID_MOVES(selected_square)
    masked_a_square ← selected_square & A_FILE_MASK
    masked_j_square ← selected_square & J_FILE_MASK

    top_left ← masked_a_square << 9
    top_left ← masked_a_square << 9
    top_middle ← selected_square << 10
    top_right ← masked_ << 11
    middle_right ← masked_ << 1
    bottom_right ← masked_ >> 9
    bottom_middle ← selected_square >> 10
    bottom_left ← masked_a_square >> 11
    middle_left ← masked_a_square >> 1

    possible_moves = top_left | top_middle | top_right | middle_right | bottom_right |
    bottom_middle | bottom_left | middle_left
    valid_moves = possible_moves & ~ ALL_PIECES_BITBOARD

    return valid_moves
end function

```

---

#### 2.2.4 Evaluation Function

The evaluation function is a heuristic algorithm to determine the relative value of a position. It outputs a real number corresponding to the advantage given to a player if reaching the analysed position, usually at a leaf node in the minimax tree. The evaluation function therefore provides the values on which minimax works on to compute an optimal move.

In the majority of evaluation functions, the most significant factor determining the evaluation is the material balance, or summation of values of the pieces. The hand-crafted evaluation function is then optimised by tuning various other positional weighted terms, such as board control and king safety.

#### Material Value

Since laser chess is not widely documented, I have assigned relative strength values to each piece according to my experience playing the game:

- Pharaoh -  $\infty$
- Scarab - 200
- Anubis - 110

- Pyramid - 100

To find the number of pieces, we can iterate through the piece bitboard with the following popcount function:

---

**Algorithm 7** Popcount pseudocode

---

```
function POPCOUNT(bitboard)
    count ← 0
    while bitboard do
        count ← count + 1
        bitboard ← bitboard&(bitboard - 1)
    end while
    return count
end function
```

---

Algorithm 7 continually resets the left-most 1 bit, incrementing a counter for each loop. Once the number of pieces has been established, we multiply this number by the piece value. Repeating this for every piece type, we can thus obtain a value for the total piece value on the board.

### Piece-Square Tables

A piece in normal chess can differ in strength based on what square it is occupying. For example, a knight near the center of the board, controlling many squares, is stronger than a knight on the rim. Similarly, we can implement positional value for Laser Chess through Piece-Square Tables. PSQTs are one-dimensional arrays, with each item representing a value for a piece type on that specific square, encoding both material value and positional simultaneously. Each array will consist of 80 base values representing the piece's material value, with a bonus or penalty added on top for the location of the piece on each square. For example, the following PSQT is for the pharaoh piece type on an example 5x5 board:

0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Piece index

-10	-10	-10	-10	-10
-10	-10	-10	-10	-10
-5	-5	-5	-5	-5
0	0	0	0	0
5	5	5	5	5

Used to reference positional value in PSQT

Figure 2.18: PSQT showing the bonus position value gained for the square occupied by a pharaoh

For asymmetrical PSQTs, we would ideally like to label the board identically from both player's point of views. Although the PSQTs are displayed from the blue player's perspective (blue pharaoh at the bottom of the board), it uses indexes from the red player's perspective, as

arrays and lists are defined with index 0 being at the topleft of the board. We would like to flip the PSQTs to be reused with blue indexes, so that a generic algorithm can be used to sum up and calculate the total positional values for both players.

To utilise a PSQT for blue pieces, a special ‘FLIP’ table can be implemented:

<b>8</b>	70	71	72	73	74	75	76	77	78	79
<b>7</b>	60	61	62	63	64	65	66	67	68	69
<b>6</b>	50	51	52	53	54	55	56	57	58	59
<b>5</b>	40	41	42	43	44	45	46	47	48	49
<b>4</b>	30	31	32	33	34	35	36	37	38	39
<b>3</b>	20	21	22	23	24	25	26	27	28	29
<b>2</b>	10	11	12	13	14	15	16	17	18	19
<b>1</b>	0	1	2	3	4	5	6	7	8	9

**a    b    c    d    e    f    g    h    j    k**

Figure 2.19: FLIP table used to map a blue piece index to the red player’s perspective

The FLIP table is just an array of indexes, mapping every blue player’s index onto the corresponding red index. The following expression utilises the FLIP table to retrieve a blue player’s value from the red player’s PSQT:

```
blue_psqt_value = PHAROAH_PSQT[FLIP[square]]
```

The following function retrieves an array of bitboards representing piece positions from the board class, then sums up all the values of these pieces for both players, referencing the corresponding PSQT:

**Algorithm 8** Calculating positional value pseudocode

---

```

function CALCULATE_POSITIONAL_VALUE(bitboards, colour)
    positional_score  $\leftarrow$  0
    for all pieces do
        for square in bitboards[piece] do
            if square = 1 then
                if colour is blue then
                    positional_score  $\leftarrow$  positional_score + PSQT[piece][square]
                else
                    positional_score  $\leftarrow$  positional_score + PSQT[piece][FLIP[square]]
                end if
            end if
        end for
    end for
    return positional_score
end function

```

---

**Using valid squares**

Using Algorithm 6 for finding valid moves, we can implement two more improvements for our evaluation function: Mobility and King Safety.

**Mobility** is the number of legal moves a player has for a given position. This is advantageous in most cases, with a positive correlation between mobility and the strength of a position. To implement this, we simply loop over all pieces of the active colour, and sum up the number of valid moves obtained from the previous algorithm.

**King safety** (Pharaoh safety) describes the level of protection of the pharaoh, being the piece that determines a win or loss. In normal chess, this would be achieved usually by castling, or protection via position or with other pieces. Similarly, since the only way to lose in Laser Chess is via a laser, having pieces surrounding the pharaoh, either to reflect the laser or to be sacrificed, is a sensible tactic and improves king safety. Thus, a value for king safety can be achieved by finding the number of valid moves a pharaoh can make, and subtracting them from the maximum possible of moves (8) to find the number of surrounding pieces.

### 2.2.5 Shadow Mapping

Following the client's requirement for engaging visuals, I have decided to implement shadow mapping for my program, especially as lasers are the main focus of the game. Shadow mapping is a technique used to create graphical hard shadows, with the use of a depth buffer map. I have chosen to implement shadow mapping, instead of alternative lighting techniques such as ray casting and ray marching, as its efficiency is more suitable for real-time usage, and results are visually decent enough for my purposes.

For typical 3D shadow mapping, the standard approach is as follows:

1. Render the scene from the light's point of view
2. Extract a depth buffer texture from the render
3. Compare the distance of a pixel from the light to the value stored in the depth texture

4. If greater, there must be an obstacle in the way reducing the depth map value, therefore that pixel must be in shadow

To implement shadow casting for my 2D game, I have modified some steps and arrived on the final following workflow:

1. Render the scene with only occluding objects shown
2. Crop texture to align the center to the light position
3. To create a 1D depth map, transform Cartesian to polar coordinates, and increase the distance from the origin until a collision with an occluding object
4. Using polar coordinates for the real texture, compare the z-depth to the corresponding value from the depth map
5. Additively blend the light colour if z-depth is less than the depth map value

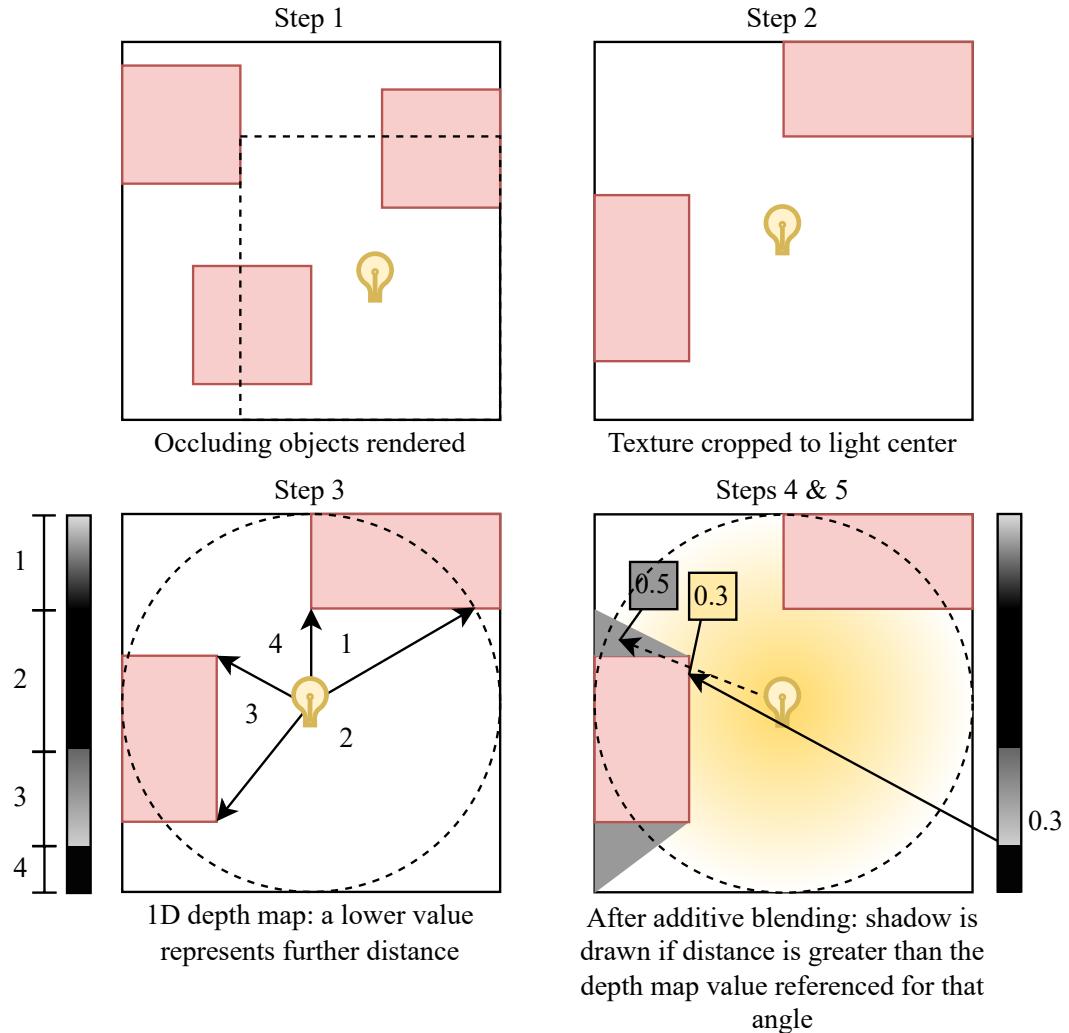


Figure 2.20: Workflow for 2D shadow mapping

Our method requires a coordinate transformation from Cartesian to polar, and vice versa. Polar to Cartesian transformation can be achieved with trigonometry, forming a right-angled triangle in the center and using the following two equations:

$$x = r \cos(\theta)$$

$$y = r \sin(\theta)$$

Cartesian to polar can also similarly be achieved with the right-angled triangle, finding the radius with the Pythagorean theorem, and the angle with arctan. However, since the range of the arctan function is only a half-circle ( $\frac{\pi}{2} < \theta < \frac{3\pi}{2}$ ), we will have to use the atan2 function, which accounts for the negative quadrants, or the following:

$$\theta = 2 \arctan \left( \frac{r - x}{y} \right)$$

There are several disadvantages to shadow mapping. The relevant ones for us are Aliasing and Shadow Acne:

**Aliasing** occurs when the texture size for the depth map is smaller than the light map, causing shadows to be scaled up and rendered with jagged edges.

**Shadow Acne** occurs when the depth from the depth map is so close to the light map value, that precision errors cause unnecessary shadows to be rendered.

These problems can be mitigated by increasing the size of the shadow map size. However, due to memory and hardware constraints, I will have to find a compromised resolution to balance both artifacting and acuity.

### Soft Shadows

The approach above is used only for calculating hard shadows. However, in real-life scenarios, lights are not modelled as a single particle, but instead emitted from a wide light source. This creates an umbra and penumbra, resulting in soft shadows.

To emulate this in our game, we could calculate penumbra values with various methods, however, due to hardware constraints and simplicity again, I have chosen to use the following simpler method:

1. Sample the depth map multiple times, from various differing angles
2. Sum the results using a normal distribution
3. Blur the final result proportional to the length from the center

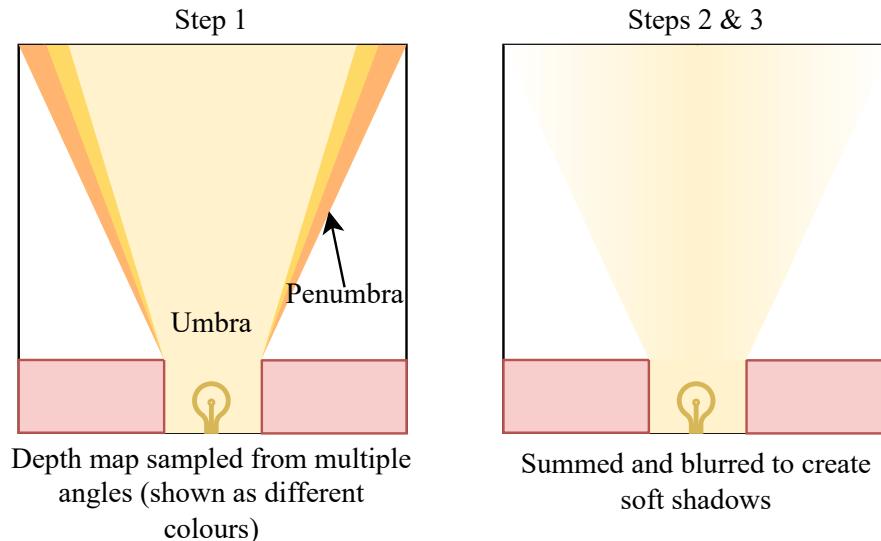


Figure 2.21: Workflow for 2D soft shadows

This method progressively blurs the shadow as the distance from the main shadow (umbra) increases, which results in a convincing estimation while being less computationally intensive.

### 2.2.6 Multithreading

In order to fulfill Objective 6 of a responsive GUI, I will have to employ multi-threading. Since python runs on a single thread natively, code is executed serially, meaning that a time consuming function such as minimax will prevent the running of another GUI-drawing function until it is finished, hence freezing the program. To overcome this, multi-threading can execute both functions in parallel on different threads, meaning the GUI-drawing thread can run while minimax is being computed, and stay responsive. To pass data between threads, since memory is shared between threads, arrays and queues can be used to store results from threads. The following flowchart shows my chosen approach to keep the GUI responsive while minimax is being computed:

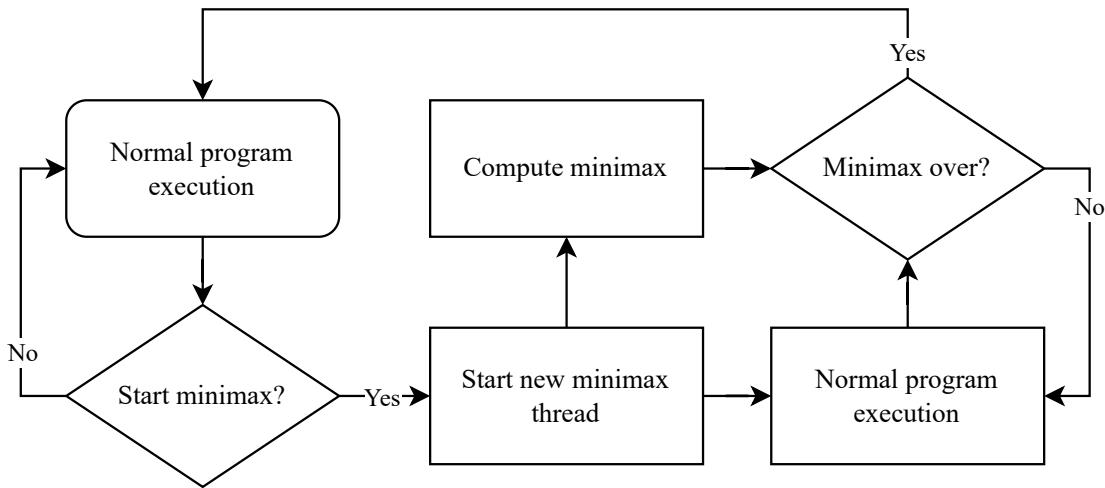


Figure 2.22: Multi-threading for minimax

## 2.3 Data Structures

### 2.3.1 Database

To achieve Objective 2 and stores previous games, I have opted to use a relational database. Choosing between different relational database, I have decided to use SQLite, since it does not require additional server softwares, has good performance with low memory requirements, and adequate for my use cases, with others such as Postgres being overkill.

#### DDL

Only a single entity will be required for my program, a table to store games. The table schema will be defined as follows:

Table: games			
Field	Key	Data Type	Validation
game_id	Primary	INT	NOT NULL
winner		INT	
cpu_depth		INT	

number_of_moves	INT	NOT NULL
cpu_enabled	BOOL	NOT NULL
moves	TEXT	NOT NULL
initial_board_configuration	TEXT	NOT NULL
time	FLOAT	
created_dt	TIMESTAMP	NOT NULL

Table 2.1: Data table scheme for *games* table

All fields are either generated or retrieved from the board class, with the exception of the moves attribute, which will need to be encoded into a suitable data type such as a string. All attributes are also independent of each other<sup>1</sup>, and so the the table therefore adheres to the third normal form.

To create the entity, a `CREATE` statement like the following can be used:

---

```

1   CREATE TABLE games(
2       id INTEGER PRIMARY KEY,
3       winner INTEGER,
4       cpu_depth INTEGER,
5       time real NOT NULL,
6       moves TEXT NOT NULL,
7       cpu_enabled INTEGER NOT NULL,
8       created_dt TIMESTAMP NOT NULL,
9       number_of_moves INTEGER NOT NULL,
10      initial_fen_string TEXT NOT NULL,
11  )

```

---

Removing an entity can also be done in a similar fashion:

---

```

1   DROP TABLE games

```

---

Migrations are a version control system to track incremental changes to the schema of a database. Since there is no popular SQL Python-binding libraries that support migrations, I will just be using a manual solution of creating python files that represent a change in my schema, defining functions that make use of SQL `ALTER` statements. This allows me to keep track of any changes, and rollback to a previous schema.

## DML

To insert a new game entry into the table, an `INSERT` statement can be used with the provided array, where the appropiate arguments are binded to the correct attribute via ? placeholders when run.

---

```

1   INSERT INTO games (
2       cpu_enabled,
3       cpu_depth,
4       winner,
5       time,
6       number_of_moves,
7       moves,
8       initial_fen_string,

```

---

<sup>1</sup>There is a case to be made for *moves* and *number\_of\_moves*, however I have included *number\_of\_moves* to save the computational effort of parsing the moves for every game just to display it on the browser preview section.

---

```

9         created_dt
10    )
11  VALUES  (?, ?, ?, ?, ?, ?, ?, ?, ?)
```

---

Moreover, we will need to fetch the number of total game entries in the table to be displayed to the user. To do this, the aggregate function `COUNT` can be used, which is supported by all SQL databases.

---

```
1  SELECT COUNT(*) FROM games
```

---

## Pagination

When there are a large number of entries in the table, it would be appropriate to display all the games to the user in a paginated form, where they can scroll between different pages and groups of games. There are multiple methods to paginate data, such as using `LIMIT` and `OFFSET` clauses, or cursor-based pagination, but I have opted to use the `ROW_NUMBER()` function.

`ROW_NUMBER()` is a window function that assigns a sequential integer to a query's result set. If I were to query the entire table, each row would be assigned an integer that could be used to check if the row is in the bounds for the current page, and therefore be displayed. Moreover, the use of an `ORDER BY` clause enables sorting of the output rows, allowing the user to choose what order the games are presented in based on an attribute such as number of moves. A `PARTITION BY` clause will also be used to group the results base on an attribute such as winner prior to sorting, if the user wants to search for games based on multiple criteria with greater ease.

The start row and end row will be passed as parameters to the placeholders in the SQL statement, calculated by multiplying the page number by the number of games per page.

---

```

1  SELECT * FROM
2    (SELECT ROW_NUMBER() OVER (
3      PARTITION BY attribute1
4      ORDER BY attribute2 ASC
5    ) AS row_num, * FROM games)
6  WHERE row_num >= ? AND row_num <= ?
```

---

## Security

Security measures such as database file permissions and encryption are common for a SQL database. However, since SQLite is a serverless database, and my program runs without any need for an internet connection, the risk of vulnerabilities is greatly reduced. Additionally, the game data stored on my database is frankly inconsequential, so going to great lengths to protect it wouldn't be to best use of my time. Nevertheless, my SQL Python-binding does support the user of placeholdeers for parameteres, thereby addressing the risk of SQL injection attacks.

### 2.3.2 Linked Lists

Another data structure I intend to implement is linked lists. This will be integrated into widgets such as the carousel or multiple icon button widget, since these will contain a variable number of items, and where  $O(1)$  random access is not a priority. Since moving back and forth between nodes is a must for a carousel widget, the linked list will be doubly-linked, with each node containing to its previous and next node. The list will also need to loop, with the next pointer of the last node pointing back to the first node, making it a circular linked list.

The following pseudocode outlines the basic functionality of the linked list:

---

**Algorithm 9** Circular doubly linked list pseudocode

---

```
function INSERT_AT_FRONT(node)
    if head is none then
        head ← node
        node.next ← node.previous ← head
    else
        node.next ← head
        node.previous ← head.previous
        head.previous.next ← node
        head.previous ← node

        head ← node
    end if
end function
```

**Require:**  $\text{LEN}(list) > 0$ 

```
function DATA_IN_LIST(data)
    current_node ← head.next
    while current_node ≠ head do
        if current_node.data = data then
            return True
        end if
        current_node ← current_node.next
    end while
    return False
end function
```

**Require:** Data in list

```
function REMOVE(data)
    current_node ← head
    while current_node.data ≠ data do
        current_node ← current_node.next
    end while

    current_node.previous.next ← current_node.next
    current_node.next.previous ← current_node.previous

    delete current_node
end function
```

---

### 2.3.3 Stack

Being a data structure with LIFO ordering, a stack is used for handling moves in the review screen. Starting with full stack of moves, every move undone pops an element off the stack to be processed. This move is then pushed onto a second stack. Therefore, cycling between moves requires pushing and popping between the two stacks, as shown in Figure 2.23. The same functionality can be achieved using a queue, but I have chosen to use two stacks as it is simpler

to implement, as being able to quickly check the number of items in each will come in handy.

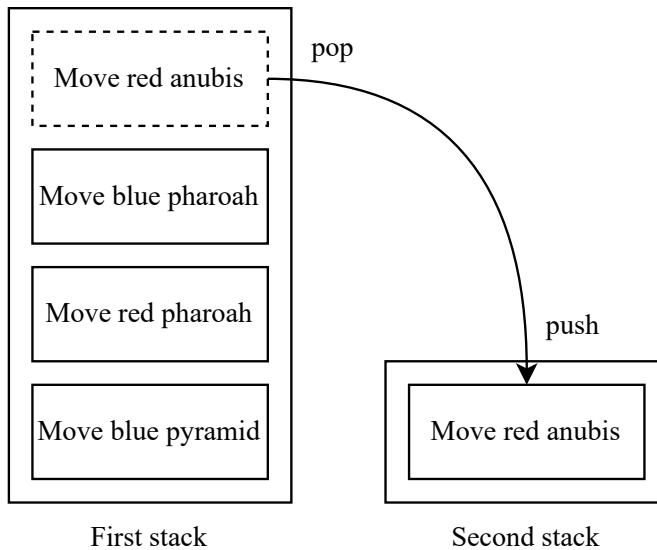


Figure 2.23: *Move red anubis* is undone and pushed onto the second stack

## 2.4 Classes

I will be using an Object-Oriented Programming (OOP) paradigm for my program. OOP reduces repetition of code, as inheritance can be used to abstract repetitive code into a base class, as shown in my widgets implementation. Testing and debugging classes will make my workflow more efficient. This section documents the base classes I am going to implement in my program.

### State

Since there will be multiple screens in my program as demonstrated in Figure 2.1, the State base class will be used to handle the logic for each screen. For each screen, void functions will be inherited and overwritten, each containing their own logic for that specific screen. For example, all screens will call the startup function in Table 2.2 to initialise variables needed for that screen. This polymorphism approach allows me to use another Control class to enable easy switching between screens, without worrying about the internal logic of that screen. Virtual methods also allow methods such as `draw` to be abstracted to the State superclass, reducing code in the inherited subclasses, while allowing them to override the methods and add their own logic.

Method Name	Description
<code>startup</code>	Initialise variables and functions when state set as displayed screen
<code>cleanup</code>	Cleanup any variables and functions when state removed from screen
<code>draw</code>	Draw to display
<code>update</code>	Update any variables for every game tick
<code>handle_resize</code>	Scale GUI when window resized
<code>get_event</code>	Receive pygame events as argument and process them

Table 2.2: Methods for State class

## Widget

I will be implementing my own widget system for creating the game GUI. This allows me to fully customise all graphical elements on the screen, and also create a resizing system that adheres to Objective 6. The default pygame rescaling options also simply resize elements without accounting for aspect ratios or resolution, and I could not find a library that suits my needs. Having a bespoke GUI implementation also justifies my use of Pygame over other Python frameworks.

I will be utilising the Pygame sprite system for my GUI. All GUI widgets will be subclasses inheriting from the base Widget class, which itself is a subclass of the Pygame sprite class. Since Pygame sprites are drawn via a spritegroup class, I will also have to create a custom subclass inheriting that as well. As with the State class, polymorphism will allow the spritegroup class to render all widgets regardless of their functionality. Each widget will override their base methods, especially the draw (set\_image) method, for their own needs. Additionally, I will use getter and setter methods, used with the `@property` decorator in python, to compute attributes mainly used for resizing widgets. This allows me to expose common variables, and to reduce code repetition.

Method Name	Description
set_image	Render widget to internal image attribute for pygame sprite class
set_geometry	Set position and size of image
set_screen_size	Set screen size for resizing purposes
get_event	Receives pygame events and processes them
screen_size*	Returns screen size in pixels
position*	Returns topleft of widget rect
size*	Returns size of widget in pixels
margin*	Returns distance between border and actual widget image
border_width*	Returns border width
border_radius*	Returns border radius for rounded corners
font_size*	Returns font size for text-based widgets

\* represents getter method / property

Table 2.3: Methods for Widget class

I will also employ multiple inheritance to combine different base class functionalities together. For example, I will create a pressable base class, designed to be subclassed along with the widget class. This will provide attributes and methods for widgets that support clicking and dragging. Following Python's Method Resolution Order (MRO), additional base classes should be referenced first, having priority over the base Widget class.

Method Name	Description
get_event	Receives Pygame events and sets current state accordingly
set_state	Sets current Pressable state, called by <code>get_event</code>
set_colours	Set fill colour according to widget Pressable state
current_state*	Returns current Pressable state (e.g. hovered, pressed etc.)

---

**Method Name | Description**


---



---

 \* represents getter method / property
 

---

Table 2.4: Methods for example Pressable class

**Game**

For my game screen, I will be utilising the Model-View-Controller architectural pattern (MVC). MVC defines three interconnected parts, the model processing information, the view showing the information, and the controlling receiving user inputs and connecting the two. This will allow me to decompose the development process into individual parts for the game logic, graphics and user input, speeding up the development process and making testing easier. It also allows me to implement multiple views, for the pause and win screens as well. For MVC, I will have to implement a game model class, a game controller class, and three classes for each view (game, pause, win). Using aggregation, these will be initially connected and handled by the game state class. For the following methods, I have only showed those pertinent to the MVC pattern:

<b>Method Name</b>	<b>Description</b>
get_event	Receives Pygame events and passes them onto the correct part's event handler
handle_game_event	Receives events and notifies the game model and game view
handle_pause_event	Receives events and notifies the pause view
handle_win_event	Receives events and notifies the win view
...	...

Table 2.5: Methods for Controller class

<b>Method Name</b>	<b>Description</b>
process_model_event	Receives events from the model and calls the relevant method to display that information
convert_mouse_pos	Sends controller class information of widget under mouse
draw	Draw information to display
handle_resize	Scale GUI when window resized
...	...

Table 2.6: Methods for View class

<b>Method Name</b>	<b>Description</b>
register_listener	Subscribes method on view instance to an event type, so that the method receives and processes that event everytime <code>alert_listener</code> is called
alert_listener	Sends event to all subscribed instances
toggle_win	Sends event for win view
toggle_pause	Sends event for pause view
...	...

**Method Name | Description**

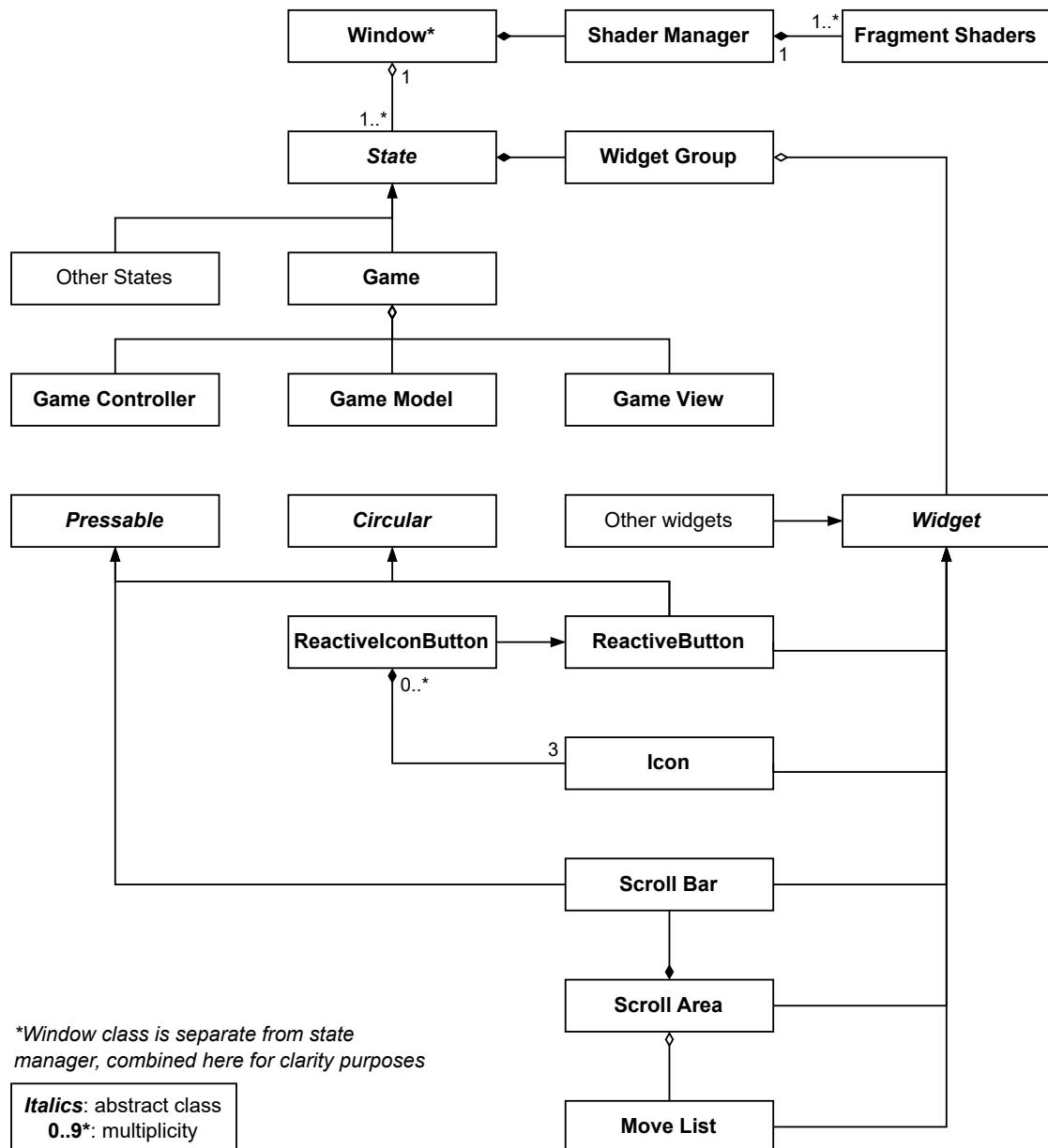
---

Table 2.7: Methods for Model class

**Shaders**

To use ModernGL with Pygame, I have created classes for each fragment shader, controlled by a main shader manager class. The fragment shader classes will rely on composition: The shader manager creates the fragment shader class; Every fragment shader class takes their shader manager parent instance as an argument, and runs methods on it to produce the final output.

### 2.4.1 Class Diagram



View complete class diagram for all widgets.

View complete class diagram for entire program.

View alternate class diagram for entire program.

(Generated with Pyreverse)

# Chapter 3

## Technical Solution

3.1	File Tree Diagram . . . . .	47
3.2	Summary of Complexity . . . . .	48
3.3	Overview . . . . .	48
3.3.1	Main . . . . .	48
3.3.2	Loading Screen . . . . .	49
3.3.3	Helper functions . . . . .	51
3.3.4	Theme . . . . .	59
3.4	GUI . . . . .	60
3.4.1	Laser . . . . .	60
3.4.2	Particles . . . . .	63
3.4.3	Widget Bases . . . . .	66
3.4.4	Widgets . . . . .	75
3.5	Game . . . . .	87
3.5.1	Model . . . . .	87
3.5.2	View . . . . .	92
3.5.3	Controller . . . . .	98
3.5.4	Board . . . . .	103
3.5.5	Bitboards . . . . .	108
3.6	CPU . . . . .	114
3.6.1	Minimax . . . . .	114
3.6.2	Alpha-beta Pruning . . . . .	116
3.6.3	Transposition Table . . . . .	118
3.6.4	Iterative Deepening . . . . .	119
3.6.5	Evaluator . . . . .	120
3.6.6	Multithreading . . . . .	123
3.6.7	Zobrist Hashing . . . . .	124
3.6.8	Cache . . . . .	125
3.7	States . . . . .	127
3.7.1	Review . . . . .	127
3.8	Database . . . . .	133
3.8.1	DDL . . . . .	133
3.8.2	DML . . . . .	134
3.9	Shaders . . . . .	137
3.9.1	Shader Manager . . . . .	137

3.9.2	Bloom . . . . .	142
3.9.3	Rays . . . . .	145

### 3.1 File Tree Diagram

To help navigate through the source code, I have included the following directory tree diagram, along with comments to explain the general purpose of code contained within specific directories and Python files.

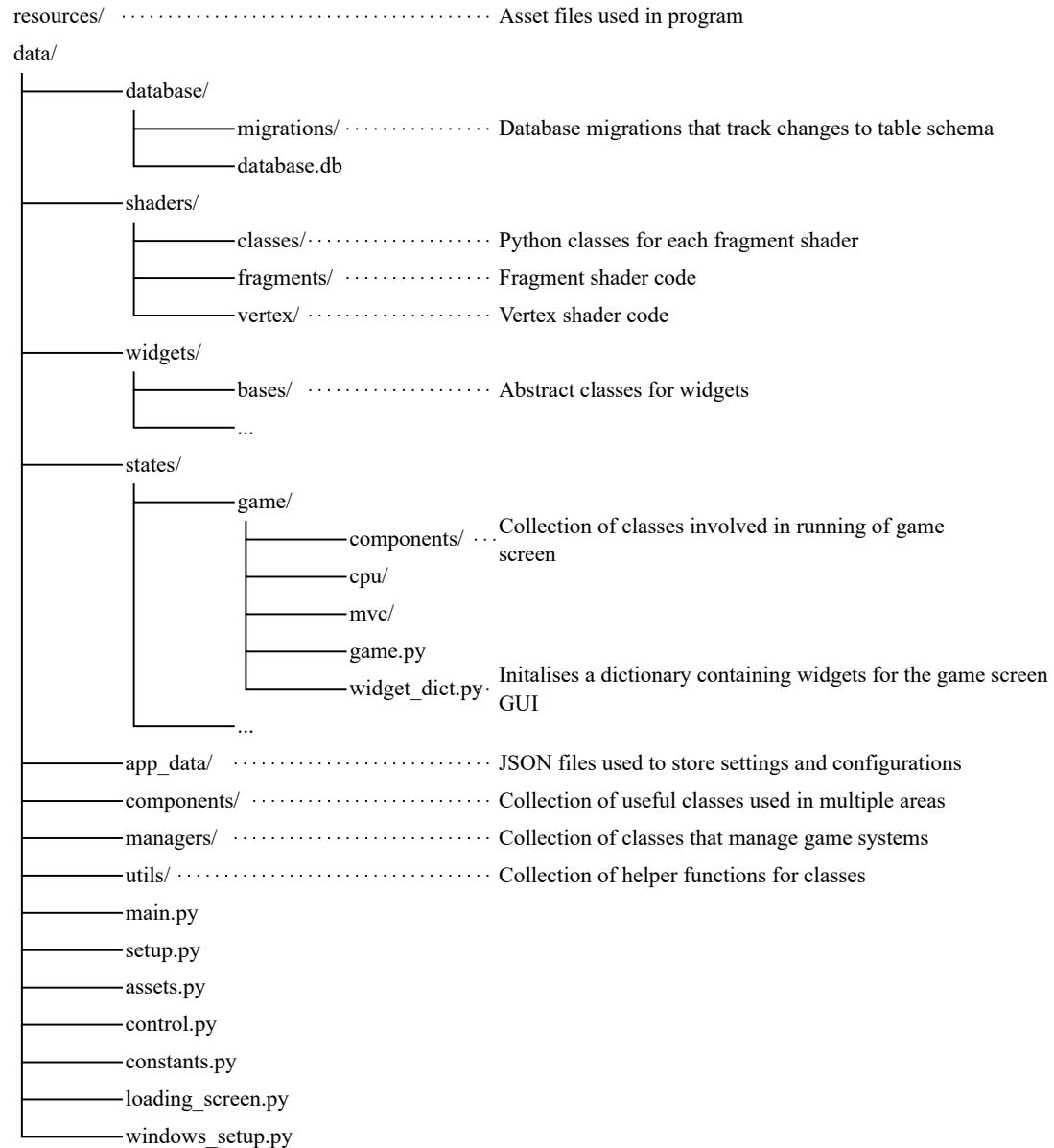


Figure 3.1: File tree diagram

## 3.2 Summary of Complexity

- Minimax improvements (3.6.2 and 3.6.3 and 3.6.4)
- Shadow mapping and coordinate transformations (3.9.3)
- Recursive Depth-First Search tree traversal (3.3.4 and 3.6.1)
- Circular doubly-linked list and stack (3.4.3 and 3.7.1)
- Multipass shaders and gaussian blur (3.9.2)
- Aggregate and Window SQL functions (3.8.2)
- OOP techniques (3.4.3 and 3.4.4)
- Multithreading (3.3.2 and 3.6.6)
- Bitboards (3.5.5)
- Zobrist hashing (3.6.7)
- (File handling and JSON parsing) (3.3.3)
- (Dictionary recursion) (3.3.4)
- (Dot product) (3.3.3 and 3.9.2)

## 3.3 Overview

### 3.3.1 Main

The file `main.py` is run by the root file `run.py`. Here resources-intensive classes such as the state and asset files are initialised, while the program displays a loading screen to hide the loading process. The main game loop is then executed.

`main.py`

```

1 from sys import platform
2 # Initialises Pygame
3 import data.setup
4
5 # Windows OS requires some configuration for Pygame to scale GUI continuously
6     # while window is being resized
6 if platform == 'win32':
7     import data.windows_setup as win_setup
8 from data.loading_screen import LoadingScreen
9
10 states = [None, None]
11
12 def load_states():
13     """
14         Initialises instances of all screens, executed on another thread with results
15         being stored to the main thread by modifying a mutable such as the states list
16     """
17     from data.control import Control
18     from data.states.game.game import Game
19     from data.states.menu.menu import Menu
      from data.states.settings.settings import Settings

```

```

20     from data.states.config.config import Config
21     from data.states.browser.browser import Browser
22     from data.states.review.review import Review
23     from data.states.editor.editor import Editor
24
25     state_dict = {
26         'menu': Menu(),
27         'game': Game(),
28         'settings': Settings(),
29         'config': Config(),
30         'browser': Browser(),
31         'review': Review(),
32         'editor': Editor()
33     }
34
35     app = Control()
36
37     states[0] = app
38     states[1] = state_dict
39
40 loading_screen = LoadingScreen(load_states)
41
42 def main():
43     """
44     Executed by run.py, starts main game loop
45     """
46     app, state_dict = states
47
48     if platform == 'win32':
49         win_setup.set_win_resize_func(app.update_window)
50
51     app.setup_states(state_dict, 'menu')
52     app.main_game_loop()

```

### 3.3.2 Loading Screen

Multithreading is used to separate the loading screen GUI from the resources intensive actions in main.py, to keep the GUI responsive. The easing function `easeOutBack` is also used to animate the logo.

`loading_screen.py`

```

1 import pygame
2 import threading
3 import sys
4 from pathlib import Path
5 from data.helpers.load_helpers import load_gfx, load_sfx
6 from data.managers.window import window
7 from data.managers.audio import audio
8
9 FPS = 30
10 start_ticks = pygame.time.get_ticks()
11 logo_gfx_path = (Path(__file__).parent / '../resources/graphics/gui/icons/logo/
12     logo.png').resolve()
12 sfx_path_1 = (Path(__file__).parent / '../resources/sfx/loading_screen/
13     loading_screen_1.wav').resolve()
13 sfx_path_2 = (Path(__file__).parent / '../resources/sfx/loading_screen/
14     loading_screen_2.wav').resolve()
14
15 def easeOutBack(progress):
16     """

```

```

17     Represents a cubic function for easing the logo position.
18     Starts quickly and has small overshoot, then ends slowly.
19
20     Args:
21         progress (float): x-value for cubic function ranging from 0-1.
22
23     Returns:
24         float:  $2.70x^3 + 1.70x^2 + 0x + 1$ , where x is time elapsed.
25         """
26         c2 = 1.70158
27         c3 = 2.70158
28
29         return c3 * ((progress - 1) ** 3) + c2 * ((progress - 1) ** 2) + 1
30
31 class LoadingScreen:
32     def __init__(self, target_func):
33         """
34             Creates new thread, and sets the load_state() function as its target.
35             Then starts draw loop for the loading screen.
36
37             Args:
38                 target_func (Callable): function to be run on thread.
39                 """
40         self._clock = pygame.time.Clock()
41         self._thread = threading.Thread(target=target_func)
42         self._thread.start()
43
44         self._logo_surface = load_gfx(logo_gfx_path)
45         self._logo_surface = pygame.transform.scale(self._logo_surface, (96, 96))
46         audio.play_sfx(load_sfx(sfx_path_1))
47         audio.play_sfx(load_sfx(sfx_path_2))
48
49         self.run()
50
51     @property
52     def logo_position(self):
53         duration = 1000
54         displacement = 50
55         elapsed_ticks = pygame.time.get_ticks() - start_ticks
56         progress = min(1, elapsed_ticks / duration)
57         center_pos = ((window.screen.size[0] - self._logo_surface.size[0]) / 2, (
58             window.screen.size[1] - self._logo_surface.size[1]) / 2)
59
59         return (center_pos[0], center_pos[1] + displacement - displacement *
60             easeOutBack(progress))
60
61     @property
62     def logo_opacity(self):
63         return min(255, (pygame.time.get_ticks() - start_ticks) / 5)
64
65     @property
66     def duration_not_over(self):
67         return (pygame.time.get_ticks() - start_ticks) < 1500
68
69     def event_loop(self):
70         """
71             Handles events for the loading screen, no user input is taken except to
72             quit the game.
73             """
74             for event in pygame.event.get():
75                 if event.type == pygame.QUIT:
76                     pygame.quit()

```

```

76             sys.exit()
77
78     def draw(self):
79         """
80             Draws logo to screen.
81         """
82         window.screen.fill((0, 0, 0))
83
84         self._logo_surface.set_alpha(self.logo_opacity)
85         window.screen.blit(self._logo_surface, self.logo_position)
86
87         window.update()
88
89     def run(self):
90         """
91             Runs while the thread is still setting up our screens, or the minimum
92             loading screen duration is not reached yet.
93         """
94         while self._thread.is_alive() or self.duration_not_over:
95             self.event_loop()
96             self.draw()
97             self._clock.tick(FPS)

```

### 3.3.3 Helper functions

These files provide useful functions for different classes.

`asset_helpers.py` (Functions used for assets and pygame Surfaces)

```

1  import pygame
2  from PIL import Image
3  from functools import cache
4  from random import randint
5  import math
6
7  @cache
8  def scale_and_cache(image, target_size):
9      """
10         Caches image when resized repeatedly.
11
12     Args:
13         image (pygame.Surface): Image surface to be resized.
14         target_size (tuple[float, float]): New image size.
15
16     Returns:
17         pygame.Surface: Resized image surface.
18     """
19     return pygame.transform.scale(image, target_size)
20
21 @cache
22 def smoothscale_and_cache(image, target_size):
23     """
24         Same as scale_and_cache, but with the Pygame smoothscale function.
25
26     Args:
27         image (pygame.Surface): Image surface to be resized.
28         target_size (tuple[float, float]): New image size.
29
30     Returns:
31         pygame.Surface: Resized image surface.
32     """
33     return pygame.transform.smoothscale(image, target_size)

```

```

34
35 def gif_to_frames(path):
36     """
37     Uses the PIL library to break down GIFs into individual frames.
38
39     Args:
40         path (str): Directory path to GIF file.
41
42     Yields:
43         PIL.Image: Single frame.
44     """
45     try:
46         image = Image.open(path)
47
48         first_frame = image.copy().convert('RGBA')
49         yield first_frame
50         image.seek(1)
51
52         while True:
53             current_frame = image.copy()
54             yield current_frame
55             image.seek(image.tell() + 1)
56     except EOFError:
57         pass
58
59 def get_perimeter_sample(image_size, number):
60     """
61     Used for particle drawing class, generates roughly equally distributed points
62     around a rectangular image surface's perimeter.
63
64     Args:
65         image_size (tuple[float, float]): Image surface size.
66         number (int): Number of points to be generated.
67
68     Returns:
69         list[tuple[int, int], ...]: List of random points on perimeter of image
70         surface.
71     """
72     perimeter = 2 * (image_size[0] + image_size[1])
73     # Flatten perimeter to a single number representing the distance from the top-
74     # middle of the surface going clockwise, and create a list of equally spaced
75     # points
76     perimeter_offsets = [(image_size[0] / 2) + (i * perimeter / number) for i in
77                           range(0, number)]
78     pos_list = []
79
80     for perimeter_offset in perimeter_offsets:
81         # For every point, add a random offset
82         max_displacement = int(perimeter / (number * 4))
83         perimeter_offset += randint(-max_displacement, max_displacement)
84
85         if perimeter_offset > perimeter:
86             perimeter_offset -= perimeter
87
88         # Convert 1D distance back into 2D points on image surface perimeter
89         if perimeter_offset < image_size[0]:
90             pos_list.append((perimeter_offset, 0))
91         elif perimeter_offset < image_size[0] + image_size[1]:
92             pos_list.append((image_size[0], perimeter_offset - image_size[0]))
93         elif perimeter_offset < image_size[0] + image_size[1] + image_size[0]:
94             pos_list.append((perimeter_offset - image_size[0] - image_size[1],
95                             image_size[1]))
```

```

90         else:
91             pos_list.append((0, perimeter - perimeter_offset))
92     return pos_list
93
94 def get_angle_between_vectors(u, v, deg=True):
95     """
96     Uses the dot product formula to find the angle between two vectors.
97
98     Args:
99         u (list[int, int]): Vector 1.
100        v (list[int, int]): Vector 2.
101       deg (bool, optional): Return results in degrees. Defaults to True.
102
103    Returns:
104        float: Angle between vectors.
105    """
106    dot_product = sum(i * j for i, j in zip(u, v))
107    u_magnitude = math.sqrt(u[0] ** 2 + u[1] ** 2)
108    v_magnitude = math.sqrt(v[0] ** 2 + v[1] ** 2)
109
110    cos_angle = dot_product / (u_magnitude * v_magnitude)
111    radians = math.acos(min(max(cos_angle, -1), 1))
112
113    if deg:
114        return math.degrees(radians)
115    else:
116        return radians
117
118 def get_rotational_angle(u, v, deg=True):
119     """
120     Get bearing angle relative to positive x-axis centered on second vector.
121
122     Args:
123         u (list[int, int]): Vector 1.
124         v (list[int, int]): Vector 2, set as center of axes.
125       deg (bool, optional): Return results in degrees. Defaults to True.
126
127    Returns:
128        float: Bearing angle between vectors.
129    """
130    radians = math.atan2(u[1] - v[1], u[0] - v[0])
131
132    if deg:
133        return math.degrees(radians)
134    else:
135        return radians
136
137 def get_vector(src_vertex, dest_vertex):
138     """
139     Get vector describing translation between two points.
140
141     Args:
142         src_vertex (list[int, int]): Source vertex.
143         dest_vertex (list[int, int]): Destination vertex.
144
145     Returns:
146         tuple[int, int]: Vector between the two points.
147    """
148    return (dest_vertex[0] - src_vertex[0], dest_vertex[1] - src_vertex[1])
149
150 def get_next_corner(vertex, image_size):
151     """

```

```

152     Used in particle drawing system, finds coordinates of the next corner going
153     clockwise, given a point on the perimeter.
154
155     Args:
156         vertex (list[int, int]): Point on perimeter.
157         image_size (list[int, int]): Image size.
158
159     Returns:
160         list[int, int]: Coordinates of corner on perimeter.
161         """
162         corners = [(0, 0), (image_size[0], 0), (image_size[0], image_size[1]), (0,
163             image_size[1])]
164
165         if vertex in corners:
166             return corners[(corners.index(vertex) + 1) % len(corners)]
167
168         if vertex[1] == 0:
169             return (image_size[0], 0)
170         elif vertex[0] == image_size[0]:
171             return image_size
172         elif vertex[1] == image_size[1]:
173             return (0, image_size[1])
174         elif vertex[0] == 0:
175             return (0, 0)
176
177     def pil_image_to_surface(pil_image):
178         """
179         Args:
180             pil_image (PIL.Image): Image to be converted.
181
182         Returns:
183             pygame.Surface: Converted image surface.
184         """
185         return pygame.image.frombytes(pil_image.tobytes(), pil_image.size, pil_image.
186             mode).convert()
187
188     def calculate_frame_index(elapsed_milliseconds, start_index, end_index, fps):
189         """
190         Determine frame of animated GIF to be displayed.
191
192         Args:
193             elapsed_milliseconds (int): Milliseconds since GIF started playing.
194             start_index (int): Start frame of GIF.
195             end_index (int): End frame of GIF.
196             fps (int): Number of frames to be played per second.
197
198         Returns:
199             int: Displayed frame index of GIF.
200         """
201         ms_per_frame = int(1000 / fps)
202         return start_index + ((elapsed_milliseconds // ms_per_frame) % (end_index -
203             start_index))
204
205     def draw_background(screen, background, current_time=0):
206         """
207         Draws background to screen
208
209         Args:
210             screen (pygame.Surface): Screen to be drawn to
211             background (list[pygame.Surface, ...] | pygame.Surface): Background to be
212                 drawn, if GIF, list of surfaces indexed to select frame to be drawn
213             current_time (int, optional): Used to calculate frame index for GIF.

```

```

    Defaults to 0.
209 """
210     if isinstance(background, list):
211         # Animated background passed in as list of surfaces, calculate_frame_index
212         # used to get index of frame to be drawn
213         frame_index = calculate_frame_index(current_time, 0, len(background), fps
214 =8)
215         scaled_background = scale_and_cache(background[frame_index], screen.size)
216         screen.blit(scaled_background, (0, 0))
217     else:
218         scaled_background = scale_and_cache(background, screen.size)
219         screen.blit(scaled_background, (0, 0))

220 def get_highlighted_icon(icon):
221 """
222     Used for pressable icons, draws overlay on icon to show as pressed.
223
224     Args:
225         icon (pygame.Surface): Icon surface.
226
227     Returns:
228         pygame.Surface: Icon with overlay drawn on top.
229     """
230     icon_copy = icon.copy()
231     overlay = pygame.Surface((icon.get_width(), icon.get_height()), pygame.
232 SRCALPHA)
233     overlay.fill((0, 0, 0, 128))
234     icon_copy.blit(overlay, (0, 0))
235     return icon_copy

```

data\_helpers.py (Functions used for file handling and JSON parsing)

```

1 import json
2 from pathlib import Path
3
4 module_path = Path(__file__).parent
5 default_file_path = (module_path / '../app_data/default_settings.json').resolve()
6 user_file_path = (module_path / '../app_data/user_settings.json').resolve()
7 themes_file_path = (module_path / '../app_data/themes.json').resolve()
8
9 def load_json(path):
10 """
11     Args:
12         path (str): Path to JSON file.
13
14     Raises:
15         Exception: Invalid file.
16
17     Returns:
18         dict: Parsed JSON file.
19     """
20     try:
21         with open(path, 'r') as f:
22             file = json.load(f)
23
24         return file
25     except:
26         raise Exception('Invalid JSON file (data_helpers.py)')
27
28 def get_user_settings():
29     return load_json(user_file_path)

```

```

30
31 def get_default_settings():
32     return load_json(default_file_path)
33
34 def get_themes():
35     return load_json(themes_file_path)
36
37 def update_user_settings(data):
38     """
39     Rewrites JSON file for user settings with new data.
40
41     Args:
42         data (dict): Dictionary storing updated user settings.
43
44     Raises:
45         Exception: Invalid file.
46     """
47     try:
48         with open(user_file_path, 'w') as f:
49             json.dump(data, f, indent=4)
50     except:
51         raise Exception('Invalid JSON file (data_helpers.py)')

widget_helpers.py (Files used for creating widgets)

1 import pygame
2 from math import sqrt
3
4 def create_slider(size, fill_colour, border_width, border_colour):
5     """
5     Creates surface for sliders.
6
7     Args:
8         size (list[int, int]): Image size.
9         fill_colour (pygame.Color): Fill (inner) colour.
10        border_width (float): Border width.
11        border_colour (pygame.Color): Border colour.
12
13    Returns:
14        pygame.Surface: Slider image surface.
15    """
16    gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
17    border_rect = pygame.FRect((0, 0, gradient_surface.width, gradient_surface.
18                                height))
19
20    # Draws rectangle with a border radius half of image height, to draw an
21    # rectangle with semicircular cap (obround)
22    pygame.draw.rect(gradient_surface, fill_colour, border_rect, border_radius=int(
23        size[1] / 2))
24    pygame.draw.rect(gradient_surface, border_colour, border_rect, width=int(
25        border_width), border_radius=int(size[1] / 2))
26
27    return gradient_surface
28
29 def create_slider_gradient(size, border_width, border_colour):
30     """
31     Draws surface for colour slider, with a full colour gradient as fill colour.
32
33     Args:
34         size (list[int, int]): Image size.
35         border_width (float): Border width.

```

```

33         border_colour (pygame.Color): Border colour.
34
35     Returns:
36         pygame.Surface: Slider image surface.
37     """
38     gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
39
40     first_round_end = gradient_surface.height / 2
41     second_round_end = gradient_surface.width - first_round_end
42     gradient_y_mid = gradient_surface.height / 2
43
44     # Iterate through length of slider
45     for i in range(gradient_surface.width):
46         draw_height = gradient_surface.height
47
48         if i < first_round_end or i > second_round_end:
49             # Draw semicircular caps if x-distance less than or greater than
50             # radius of cap (half of image height)
51             distance_from_cutoff = min(abs(first_round_end - i), abs(i -
52             second_round_end))
53             draw_height = calculate_gradient_slice_height(distance_from_cutoff,
54             gradient_surface.height / 2)
55
56             # Get colour from distance from left side of slider
57             color = pygame.Color(0)
58             color.hsva = (int(360 * i / gradient_surface.width), 100, 100, 100)
59
60             draw_rect = pygame.FRect((0, 0, 1, draw_height - 2 * border_width))
61             draw_rect.center = (i, gradient_y_mid)
62
63             pygame.draw.rect(gradient_surface, color, draw_rect)
64
65     border_rect = pygame.FRect((0, 0, gradient_surface.width, gradient_surface.
66     height))
67     pygame.draw.rect(gradient_surface, border_colour, border_rect, width=int(
68     border_width), border_radius=int(size[1] / 2))
69
70     return gradient_surface
71
72 def calculate_gradient_slice_height(distance, radius):
73     """
74     Calculate height of vertical slice of semicircular slider cap.
75
76     Args:
77         distance (float): x-distance from center of circle.
78         radius (float): Radius of semicircle.
79
80     Returns:
81         float: Height of vertical slice.
82     """
83     return sqrt(radius ** 2 - distance ** 2) * 2 + 2
84
85 def create_slider_thumb(radius, colour, border_colour, border_width):
86     """
87     Creates surface with bordered circle.
88
89     Args:
90         radius (float): Radius of circle.
91         colour (pygame.Color): Fill colour.
92         border_colour (pygame.Color): Border colour.
93         border_width (float): Border width.
94
95

```

```

90     Returns:
91         pygame.Surface: Circle surface.
92     """
93     thumb_surface = pygame.Surface((radius * 2, radius * 2), pygame.SRCALPHA)
94     pygame.draw.circle(thumb_surface, border_colour, (radius, radius), radius,
95     width=int(border_width))
96     pygame.draw.circle(thumb_surface, colour, (radius, radius), (radius -
97     border_width))
98
99     return thumb_surface
100
101 def create_square_gradient(side_length, colour):
102     """
103     Creates a square gradient for the colour picker widget, gradient transitioning
104     between saturation and value.
105     Uses smoothscale to blend between colour values for individual pixels.
106
107     Args:
108         side_length (float): Length of a square side.
109         colour (pygame.Color): Colour with desired hue value.
110
111     Returns:
112         pygame.Surface: Square gradient surface.
113     """
114     square_surface = pygame.Surface((side_length, side_length))
115
116     mix_1 = pygame.Surface((1, 2))
117     mix_1.fill((255, 255, 255))
118     mix_1.set_at((0, 1), (0, 0, 0))
119     mix_1 = pygame.transform.smoothscale(mix_1, (side_length, side_length))
120
121     hue = colour.hsva[0]
122     saturated_rgb = pygame.Color(0)
123     saturated_rgb.hsva = (hue, 100, 100)
124
125     mix_2 = pygame.Surface((2, 1))
126     mix_2.fill((255, 255, 255))
127     mix_2.set_at((1, 0), saturated_rgb)
128     mix_2 = pygame.transform.smoothscale(mix_2, (side_length, side_length))
129
130     mix_1.blit(mix_2, (0, 0), special_flags=pygame.BLEND_MULT)
131
132     square_surface.blit(mix_1, (0, 0))
133
134     return square_surface
135
136 def create_switch(size, colour):
137     """
138     Creates surface for switch toggle widget.
139
140     Args:
141         size (list[int, int]): Image size.
142         colour (pygame.Color): Fill colour.
143
144     Returns:
145         pygame.Surface: Switch surface.
146     """
147     switch_surface = pygame.Surface((size[0], size[1]), pygame.SRCALPHA)
148     pygame.draw.rect(switch_surface, colour, (0, 0, size[0], size[1]),
149     border_radius=int(size[1] / 2))
150
151     return switch_surface

```

```

148
149 def create_text_box(size, border_width, colours):
150     """
151     Creates bordered textbox with shadow, flat, and highlighted vertical regions.
152
153     Args:
154         size (list[int, int]): Image size.
155         border_width (float): Border width.
156         colours (list[pygame.Color, ...]): List of 4 colours, representing border
157             colour, shadow colour, flat colour and highlighted colour.
158
159     Returns:
160         pygame.Surface: Textbox surface.
161
162     """
163     surface = pygame.Surface(size, pygame.SRCALPHA)
164
165     pygame.draw.rect(surface, colours[0], (0, 0, *size))
166     pygame.draw.rect(surface, colours[2], (border_width, border_width, size[0] - 2
167         * border_width, size[1] - 2 * border_width))
168     pygame.draw.rect(surface, colours[3], (border_width, border_width, size[0] - 2
169         * border_width, border_width))
170     pygame.draw.rect(surface, colours[1], (border_width, size[1] - 2 *
171         border_width, size[0] - 2 * border_width, border_width))
172
173     return surface
174

```

### 3.3.4 Theme

The theme manager file is responsible for providing an instance where the colour palette and dimensions for the GUI can be accessed. Values read from a JSON file are **recursively** flattened, with keys created from the dictionary hierarchy, and stored into the internal dictionary of a `ThemeManager` object.

`theme.py`

```

1 from data.helpers.data_helpers import get_themes, get_user_settings
2
3 themes = get_themes()
4 user_settings = get_user_settings()
5
6 def flatten_dictionary_generator(dictionary, parent_key=None):
7     """
8     Recursive depth-first search to yield all items in a dictionary.
9
10    Args:
11        dictionary (dict): Dictionary to be iterated through.
12        parent_key (str, optional): Prefix added to every key. Defaults to None.
13
14    Yields:
15        dict | tuple[str, str]: Another dictionary or key, value pair.
16    """
17    for key, value in dictionary.items():
18        if parent_key:
19            new_key = parent_key + key.capitalize()
20        else:
21            new_key = key
22
23        if isinstance(value, dict):
24            yield from flatten_dictionary_generator(value, new_key).items()
25        else:
26            yield new_key, value

```

```

27
28 def flatten_dictionary(dictionary, parent_key=''):
29     return dict(flatten_dictionary_generator(dictionary, parent_key))
30
31 class ThemeManager:
32     def __init__(self):
33         self.__dict__.update(flatten_dictionary(themes['colours']))
34         self.__dict__.update(flatten_dictionary(themes['dimensions']))
35
36     def __getitem__(self, arg):
37         """
38             Override default class's __getitem__ dunder method, to make retrieving an
39             instance attribute nicer with [] notation.
40
41         Args:
42             arg (str): Attribute name.
43
44         Raises:
45             KeyError: Instance does not have requested attribute.
46
47         Returns:
48             str | int: Instance attribute.
49
50         item = self.__dict__.get(arg)
51
52         if item is None:
53             raise KeyError('(ThemeManager.__getitem__) Requested theme item not
54             found:', arg)
55
56 theme = ThemeManager()

```

## 3.4 GUI

### 3.4.1 Laser

The `LaserDraw` class draws the laser in both the game and review screens.

```

laser_draw.py
1 import pygame
2 from data.helpers.board_helpers import coords_to_screen_pos
3 from data.utils.enums import LaserType, Colour, ShaderType
4 from data.managers.animation import animation
5 from data.utils.assets import GRAPHICS, SFX
6 from data.utils.constants import EMPTY_BB
7 from data.managers.window import window
8 from data.managers.audio import audio
9
10 type_to_image = {
11     LaserType.END: ['laser_end_1', 'laser_end_2'],
12     LaserType.STRAIGHT: ['laser_straight_1', 'laser_straight_2'],
13     LaserType.CORNER: ['laser_corner_1', 'laser_corner_2']
14 }
15 GLOW_SCALE_FACTOR = 1.5
16
17 class LaserDraw:
18     def __init__(self, board_position, board_size):
19         self._board_position = board_position

```

```

21         self._square_size = board_size[0] / 10
22         self._laser_lists = []
23
24     @property
25     def firing(self):
26         return len(self._laser_lists) > 0
27
28     def add_laser(self, laser_result, laser_colour):
29         """
30             Adds a laser to the board.
31
32             Args:
33                 laser_result (Laser): Laser class instance containing laser trajectory
34                 info.
35                 laser_colour (Colour.RED | Colour.BLUE): Active colour of laser.
36
37             laser_path = laser_result.laser_path.copy()
38             laser_types = [LaserType.END]
39             # List of angles in degree to rotate the laser image surface when drawn
40             laser_rotation = [laser_path[0][1]]
41             laser_lights = []
42
43             # Iterates through every square laser passes through
44             for i in range(1, len(laser_path)):
45                 previous_direction = laser_path[i-1][1]
46                 current_coords, current_direction = laser_path[i]
47
48                 if current_direction == previous_direction:
49                     laser_types.append(LaserType.STRAIGHT)
50                     laser_rotation.append(current_direction)
51                 elif current_direction == previous_direction.get_clockwise():
52                     laser_types.append(LaserType.CORNER)
53                     laser_rotation.append(current_direction)
54                 elif current_direction == previous_direction.get_anticlockwise():
55                     laser_types.append(LaserType.CORNER)
56                     laser_rotation.append(current_direction.get_anticlockwise())
57
58             # Adds a shader ray effect on the first and last square of the laser
59             trajectory
60             if i in [1, len(laser_path) - 1]:
61                 abs_position = coords_to_screen_pos(current_coords, self.
62                 _board_position, self._square_size)
63                 laser_lights.append([
64                     (abs_position[0] / window.size[0], abs_position[1] / window.
65                     size[1]),
66                     0.35,
67                     (0, 0, 255) if laser_colour == Colour.BLUE else (255, 0, 0),
68                     ])
69
70             # Sets end laser draw type if laser hits a piece
71             if laser_result.hit_square_bitboard != EMPTY_BB:
72                 laser_types[-1] = LaserType.END
73                 laser_path[-1] = (laser_path[-1][0], laser_path[-2][1].get_opposite())
74                 laser_rotation[-1] = laser_path[-2][1].get_opposite()
75
76             audio.play_sfx(SFX['piece_destroy'])
77
78             laser_path = [(coords, rotation, type) for (coords, dir), rotation, type
79             in zip(laser_path, laser_rotation, laser_types)]
80             self._laser_lists.append((laser_path, laser_colour))
81
82             window.clear_effect(ShaderType.RAYS)

```

```

78         window.set_effect(ShaderType.RAYS, lights=laser_lights)
79         animation.set_timer(1000, self.remove_laser)
80
81     audio.play_sfx(SFX['laser_1'])
82     audio.play_sfx(SFX['laser_2'])
83
84     def remove_laser(self):
85         """
86             Removes a laser from the board.
87         """
88         self._laser_lists.pop(0)
89
90         if len(self._laser_lists) == 0:
91             window.clear_effect(ShaderType.RAYS)
92
93     def draw_laser(self, screen, laser_list, glow=True):
94         """
95             Draws every laser on the screen.
96
97             Args:
98                 screen (pygame.Surface): The screen to draw on.
99                 laser_list (list): The list of laser segments to draw.
100                glow (bool, optional): Whether to draw a glow effect. Defaults to True
101
102            laser_path, laser_colour = laser_list
103            laser_list = []
104            glow_list = []
105
106            for coords, rotation, type in laser_path:
107                square_x, square_y = coords_to_screen_pos(coords, self._board_position
108                , self._square_size)
109
110                image = GRAPHICS[type_to_image[type][laser_colour]]
111                rotated_image = pygame.transform.rotate(image, rotation.to_angle())
112                scaled_image = pygame.transform.scale(rotated_image, (self.
113                    _square_size + 1, self._square_size + 1)) # +1 to prevent rounding creating
114                    black lines
115                laser_list.append((scaled_image, (square_x, square_y)))
116
117                # Scales up the laser image surface as a glow surface
118                scaled_glow = pygame.transform.scale(rotated_image, (self._square_size
119                    * GLOW_SCALE_FACTOR, self._square_size * GLOW_SCALE_FACTOR))
120                offset = self._square_size * ((GLOW_SCALE_FACTOR - 1) / 2)
121                glow_list.append((scaled_glow, (square_x - offset, square_y - offset)))
122
123
124    def draw(self, screen):
125        """
126            Draws all lasers on the screen.
127
128            Args:
129                screen (pygame.Surface): The screen to draw on.
130            """
131            for laser_list in self._laser_lists:
132                self.draw_laser(screen, laser_list)

```

```

134
135     def handle_resize(self, board_position, board_size):
136         """
137             Handles resizing of the board.
138
139         Args:
140             board_position (tuple[int, int]): The new position of the board.
141             board_size (tuple[int, int]): The new size of the board.
142         """
143         self._board_position = board_position
144         self._square_size = board_size[0] / 10

```

### 3.4.2 Particles

The `ParticlesDraw` class draws particles in both the game and review screens. The particles are either fragmented pieces when destroyed, or laser particles emitted from the Sphinx. Particles are given custom velocity, rotation, opacity and size parameters.

`particles_draw.py`

```

1 import pygame
2 from random import randint
3 from data.helpers.asset_helpers import get_perimeter_sample, get_vector,
4     get_angle_between_vectors, get_next_corner
5 from data.states.game.components.piece_sprite import PieceSprite
6
7 class ParticlesDraw:
8     def __init__(self, gravity=0.2, rotation=180, shrink=0.5, opacity=150):
9         self._particles = []
10        self._glow_particles = []
11
12        self._gravity = gravity
13        self._rotation = rotation
14        self._shrink = shrink
15        self._opacity = opacity
16
17    def fragment_image(self, image, number):
18        image_size = image.get_rect().size
19        """
20            1. Takes an image surface and samples random points on the perimeter.
21            2. Iterates through points, and depending on the nature of two consecutive
22                points, finds a corner between them.
23            3. Draws a polygon with the points as the vertices to mask out the area
24                not in the fragment.
25
26        Args:
27            image (pygame.Surface): Image to fragment.
28            number (int): The number of fragments to create.
29
30        Returns:
31            list[pygame.Surface]: List of image surfaces with fragment of original
32            surface drawn on top.
33        """
34        center = image.get_rect().center
35        points_list = get_perimeter_sample(image_size, number)
36        fragment_list = []
37
38        points_list.append(points_list[0])
39
40        # Iterate through points_list, using the current point and the next one
41        for i in range(len(points_list) - 1):

```

```

38         vertex_1 = points_list[i]
39         vertex_2 = points_list[i + 1]
40         vector_1 = get_vector(center, vertex_1)
41         vector_2 = get_vector(center, vertex_2)
42         angle = get_angle_between_vectors(vector_1, vector_2)
43
44         cropped_image = pygame.Surface(image_size, pygame.SRCALPHA)
45         cropped_image.fill((0, 0, 0, 0))
46         cropped_image.blit(image, (0, 0))
47
48         corners_to_draw = None
49
50         if vertex_1[0] == vertex_2[0] or vertex_1[1] == vertex_2[1]: # Points
51             on the same side
52             corners_to_draw = 4
53
54         elif abs(vertex_1[0] - vertex_2[0]) == image_size[0] or abs(vertex_1
55             [1] - vertex_2[1]) == image_size[1]: # Points on opposite sides
56             corners_to_draw = 2
57
58         elif angle < 180: # Points on adjacent sides
59             corners_to_draw = 3
60
61     else:
62         corners_to_draw = 1
63
64     corners_list = []
65     for j in range(corners_to_draw):
66         if len(corners_list) == 0:
67             corners_list.append(get_next_corner(vertex_2, image_size))
68         else:
69             corners_list.append(get_next_corner(corners_list[-1],
70                 image_size))
71
72     pygame.draw.polygon(cropped_image, (0, 0, 0, 0), (center, vertex_2, *
73     corners_list, vertex_1))
74
75     fragment_list.append(cropped_image)
76
77     return fragment_list
78
79 def add_captured_piece(self, piece, colour, rotation, position, size):
80     """
81     Adds a captured piece to fragment into particles.
82
83     Args:
84         piece (Piece): The piece type.
85         colour (Colour): The active colour of the piece.
86         rotation (int): The rotation of the piece.
87         position (tuple[int, int]): The position where particles originate
88             from.
89         size (tuple[int, int]): The size of the piece.
90
91     """
92     piece_sprite = PieceSprite(piece, colour, rotation)
93     piece_sprite.set_geometry((0, 0), size)
94     piece_sprite.set_image()
95
96     particles = self.fragment_image(piece_sprite.image, 5)
97
98     for particle in particles:
99         self.add_particle(particle, position)

```

```

95     def add_sparks(self, radius, colour, position):
96         """
97             Adds laser spark particles.
98
99         Args:
100            radius (int): The radius of the sparks.
101            colour (Colour): The active colour of the sparks.
102            position (tuple[int, int]): The position where particles originate
103            from.
104            """
105            for i in range(randint(10, 15)):
106                velocity = [randint(-15, 15) / 10, randint(-20, 0) / 10]
107                random_colour = [min(max(val + randint(-20, 20), 0), 255) for val in
108                colour]
109                self._particles.append([None, [radius, random_colour], [*position],
110                velocity, 0])
111
112    def add_particle(self, image, position):
113        """
114            Adds a particle.
115
116        Args:
117            image (pygame.Surface): The image of the particle.
118            position (tuple): The position of the particle.
119            """
120            velocity = [randint(-15, 15) / 10, randint(-20, 0) / 10]
121
122            # Each particle is stored with its attributes: [surface, copy of surface,
123            position, velocity, lifespan]
124            self._particles.append([image, image.copy(), [*position], velocity, 0])
125
126    def update(self):
127        """
128            Updates each particle and its attributes.
129
130            for i in range(len(self._particles) - 1, -1, -1):
131                particle = self._particles[i]
132
133                #update position
134                particle[2][0] += particle[3][0]
135                particle[2][1] += particle[3][1]
136
137                #update lifespan
138                self._particles[i][4] += 0.01
139
140                if self._particles[i][4] >= 1:
141                    self._particles.pop(i)
142                    continue
143
144                if isinstance(particle[1], pygame.Surface): # Particle is a piece
145                    # Update velocity
146                    particle[3][1] += self._gravity
147
148                    # Update size
149                    image_size = particle[1].get_rect().size
150                    end_size = ((1 - self._shrink) * image_size[0], (1 - self._shrink)

```

```

    _rotation) * particle[4]
151
152     updated_image = pygame.transform.scale(pygame.transform.rotate(
particle[1], rotation), target_size)
153
154     elif isinstance(particle[1], list): # Particle is a spark
155         # Update size
156         end_radius = (1 - self._shrink) * particle[1][0]
157         target_radius = particle[1][0] - particle[4] * (particle[1][0] -
end_radius)
158
159         updated_image = pygame.Surface((target_radius * 2, target_radius *
2), pygame.SRCALPHA)
160         pygame.draw.circle(updated_image, particle[1][1], (target_radius,
target_radius), target_radius)
161
162         # Update opacity
163         alpha = 255 - particle[4] * (255 - self._opacity)
164
165         updated_image.fill((255, 255, 255, alpha), None, pygame.
BLEND_RGBA_MULT)
166
167         particle[0] = updated_image
168
169     def draw(self, screen):
170         """
171             Draws the particles, indexing the surface and position attributes for each
particle.
172
173         Args:
174             screen (pygame.Surface): The screen to draw on.
175         """
176         screen.blit([
177             (particle[0], particle[2]) for particle in self._particles
178         ])

```

### 3.4.3 Widget Bases

Widget bases are used as the base classes for my widgets system. They contain both attributes and getter methods that provide both basic functionalities such as size and position, and abstract methods to be overridden. These bases are designed to be used with **multiple inheritance**, where multiple bases can be combined to add functionality to the final widget. **Encapsulation** also allows me to simplify interactions between widgets, as using getter methods instead of protected attributes allows me to add logic while accessing an attribute, such as in `widget.py`, where the logic to fetch the parent surface instead of the windows screen is hidden within the base class.

#### Widget

All widgets are a subclass of the `Widget` class.

`widget.py`

```

1 import pygame
2 from data.utils.constants import SCREEN_SIZE
3 from data.managers.theme import theme
4 from data.utils.assets import DEFAULT_FONT
5
6 DEFAULT_SURFACE_SIZE = SCREEN_SIZE
7 REQUIRED_KWARGS = ['relative_position', 'relative_size']

```

```
8
9 class _Widget(pygame.sprite.Sprite):
10     def __init__(self, **kwargs):
11         """
12             Every widget has the following attributes:
13
14             surface (pygame.Surface): The surface the widget is drawn on.
15             raw_surface_size (tuple[int, int]): The initial size of the window screen,
16                 remains constant.
17             parent (_Widget, optional): The parent widget position and size is
18                 relative to.
19
20                 Relative to current surface:
21                 relative_position (tuple[float, float]): The position of the widget
22                     relative to its surface.
23                 relative_size (tuple[float, float]): The scale of the widget relative to
24                     its surface.
25
26                 Remains constant, relative to initial screen size:
27                 relative_font_size (float, optional): The relative font size of the widget
28
29                 relative_margin (float): The relative margin of the widget.
30                 relative_border_width (float): The relative border width of the widget.
31                 relative_border_radius (float): The relative border radius of the widget.
32
33             anchor_x (str): The horizontal anchor direction ('left', 'right', 'center').
34             anchor_y (str): The vertical anchor direction ('top', 'bottom', 'center').
35             fixed_position (tuple[int, int], optional): The fixed position of the
36                 widget in pixels.
37             border_colour (pygame.Color): The border color of the widget.
38             text_colour (pygame.Color): The text color of the widget.
39             fill_colour (pygame.Color): The fill color of the widget.
40             font (pygame.freetype.Font): The font used for the widget.
41             """
42             super().__init__()
43
44             for required_kwarg in REQUIRED_KWARGS:
45                 if required_kwarg not in kwargs:
46                     raise KeyError(f'({_Widget.__init__}) Required keyword "{required_kwarg}" not in base kwargs')
47
48             self._surface = None # Set in WidgetGroup, as needs to be reassigned every
49             frame
50             self._raw_surface_size = DEFAULT_SURFACE_SIZE
51
52             self._parent = kwargs.get('parent')
53
54             self._relative_font_size = None # Set in subclass
55
56             self._relative_position = kwargs.get('relative_position')
57             self._relative_margin = theme['margin'] / self._raw_surface_size[1]
58             self._relative_border_width = theme['borderWidth'] / self._raw_surface_size[1]
59             self._relative_border_radius = theme['borderRadius'] / self._raw_surface_size[1]
60
61             self._border_colour = pygame.Color(theme['borderPrimary'])
62             self._text_colour = pygame.Color(theme['textPrimary'])
63             self._fill_colour = pygame.Color(theme['fillPrimary'])
64             self._font = DEFAULT_FONT
```

```

59         self._anchor_x = kwargs.get('anchor_x') or 'left'
60         self._anchor_y = kwargs.get('anchor_y') or 'top'
61         self._fixed_position = kwargs.get('fixed_position')
62         scale_mode = kwargs.get('scale_mode') or 'both'
63
64     if kwargs.get('relative_size'):
65         match scale_mode:
66             case 'height':
67                 self._relative_size = kwargs.get('relative_size')
68             case 'width':
69                 self._relative_size = ((kwargs.get('relative_size')[0] * self.
70 surface_size[0]) / self.surface_size[1], (kwargs.get('relative_size')[1] *
71 self.surface_size[0]) / self.surface_size[1])
72             case 'both':
73                 self._relative_size = ((kwargs.get('relative_size')[0] * self.
74 surface_size[0]) / self.surface_size[1], kwargs.get('relative_size')[1])
75             case _:
76                 raise ValueError('_Widget.__init__) Unknown scale mode:', scale_mode)
77         else:
78             self._relative_size = (1, 1)
79
80     if 'margin' in kwargs:
81         self._relative_margin = kwargs.get('margin') / self._raw_surface_size[1]
82
83         if (self._relative_margin * 2) > min(self._relative_size[0], self._relative_size[1]):
84             raise ValueError('_Widget.__init__) Margin larger than specified size!')
85
86         if 'border_width' in kwargs:
87             self._relative_border_width = kwargs.get('border_width') / self._raw_surface_size[1]
88
89         if 'border_radius' in kwargs:
90             self._relative_border_radius = kwargs.get('border_radius') / self._raw_surface_size[1]
91
92         if 'border_colour' in kwargs:
93             self._border_colour = pygame.Color(kwargs.get('border_colour'))
94
95         if 'fill_colour' in kwargs:
96             self._fill_colour = pygame.Color(kwargs.get('fill_colour'))
97
98         if 'text_colour' in kwargs:
99             self._text_colour = pygame.Color(kwargs.get('text_colour'))
100
101     @property
102     def surface_size(self):
103         """
104             Gets the size of the surface widget is drawn on.
105             Can be either the window size, or another widget size if assigned to a parent.
106
107             Returns:
108                 tuple[int, int]: The size of the surface.
109             """
110             if self._parent:

```

```

111         return self._parent.size
112     else:
113         return self._raw_surface_size
114
115     @property
116     def position(self):
117         """
118             Gets the position of the widget.
119             Accounts for fixed position attribute, where widget is positioned in
120             pixels regardless of screen size.
121             Accounts for anchor direction, where position attribute is calculated
122             relative to one side of the screen.
123
124             Returns:
125                 tuple[int, int]: The position of the widget.
126             """
127
128         x, y = None, None
129         if self._fixed_position:
130             x, y = self._fixed_position
131         if x is None:
132             x = self._relative_position[0] * self.surface_size[0]
133         if y is None:
134             y = self._relative_position[1] * self.surface_size[1]
135
136         if self._anchor_x == 'left':
137             x = x
138         elif self._anchor_x == 'right':
139             x = self.surface_size[0] - x - self.size[0]
140         elif self._anchor_x == 'center':
141             x = (self.surface_size[0] / 2 - self.size[0] / 2) + x
142
143         if self._anchor_y == 'top':
144             y = y
145         elif self._anchor_y == 'bottom':
146             y = self.surface_size[1] - y - self.size[1]
147         elif self._anchor_y == 'center':
148             y = (self.surface_size[1] / 2 - self.size[1] / 2) + y
149
150         # Position widget relative to parent, if exists.
151         if self._parent:
152             return (x + self._parent.position[0], y + self._parent.position[1])
153
154     @property
155     def size(self):
156         return (self._relative_size[0] * self.surface_size[1], self._relative_size
157 [1] * self.surface_size[1])
158
159     @property
160     def margin(self):
161         return self._relative_margin * self._raw_surface_size[1]
162
163     @property
164     def border_width(self):
165         return self._relative_border_width * self._raw_surface_size[1]
166
167     @property
168     def border_radius(self):
169         return self._relative_border_radius * self._raw_surface_size[1]

```

```

170     def font_size(self):
171         return self._relative_font_size * self.surface_size[1]
172
173     def set_image(self):
174         """
175             Abstract method to draw widget.
176         """
177         raise NotImplementedError
178
179     def set_geometry(self):
180         """
181             Sets the position and size of the widget.
182         """
183         self.rect = self.image.get_rect()
184
185         if self._anchor_x == 'left':
186             if self._anchor_y == 'top':
187                 self.rect.topleft = self.position
188             elif self._anchor_y == 'bottom':
189                 self.rect.topleft = self.position
190             elif self._anchor_y == 'center':
191                 self.rect.topleft = self.position
192         elif self._anchor_x == 'right':
193             if self._anchor_y == 'top':
194                 self.rect.topleft = self.position
195             elif self._anchor_y == 'bottom':
196                 self.rect.topleft = self.position
197             elif self._anchor_y == 'center':
198                 self.rect.topleft = self.position
199         elif self._anchor_x == 'center':
200             if self._anchor_y == 'top':
201                 self.rect.topleft = self.position
202             elif self._anchor_y == 'bottom':
203                 self.rect.topleft = self.position
204             elif self._anchor_y == 'center':
205                 self.rect.topleft = self.position
206
207     def set_surface_size(self, new_surface_size):
208         """
209             Sets the new size of the surface widget is drawn on.
210
211             Args:
212                 new_surface_size (tuple[int, int]): The new size of the surface.
213
214         self._raw_surface_size = new_surface_size
215
216     def process_event(self, event):
217         """
218             Abstract method to handle events.
219
220             Args:
221                 event (pygame.Event): The event to process.
222
223         raise NotImplementedError

```

## Circular

The `Circular` class provides an internal circular linked list, giving functionality to support widgets which rotate between text/icons. `circular.py`

```
1 from data.components.circular_linked_list import CircularLinkedList
```

```
2
3 class _Circular:
4     def __init__(self, items_dict, **kwargs):
5         # The key, value pairs are stored within a dictionary, while the keys to
6         # access them are stored within circular linked list.
7         self._items_dict = items_dict
8         self._keys_list = CircularLinkedList(list(items_dict.keys()))
9
10    @property
11    def current_key(self):
12        """
13            Gets the current head node of the linked list, and returns a key stored as
14            the node data.
15            Returns:
16                Data of linked list head.
17            """
18        return self._keys_list.get_head().data
19
20    @property
21    def current_item(self):
22        """
23            Gets the value in self._items_dict with the key being self.current_key.
24            Returns:
25                Value stored with key being current head of linked list.
26            """
27        return self._items_dict[self.current_key]
28
29    def set_next_item(self):
30        """
31            Sets the next item in as the current item.
32            """
33        self._keys_list.shift_head()
34
35    def set_previous_item(self):
36        """
37            Sets the previous item as the current item.
38            """
39        self._keys_list.unshift_head()
40
41    def set_to_key(self, key):
42        """
43            Sets the current item to the specified key.
44            Args:
45                key: The key to set as the current item.
46            Raises:
47                ValueError: If no nodes within the circular linked list contains the
48                key as its data.
49            """
50            if self._keys_list.data_in_list(key) is False:
51                raise ValueError('(_Circular.set_to_key) Key not found:', key)
52
53            for _ in range(len(self._items_dict)):
54                if self.current_key == key:
55                    self.set_image()
56                    self.set_geometry()
57                    return
58
59            self.set_next_item()
```

### Circular Linked List

As described in Section 2.3.2, the `CircularLinkedList` class implements a **circular doubly-linked list**. Used for the internal logic of the `Circular` class.

`circular_linked_list.py`

```

1  class Node:
2      def __init__(self, data):
3          self.data = data
4          self.next = None
5          self.previous = None
6
7  class CircularLinkedList:
8      def __init__(self, list_to_convert=None):
9          """
10             Initialises a CircularLinkedList object.
11
12             Args:
13                 list_to_convert (list, optional): Creates a linked list from existing
14                 items. Defaults to None.
15                 """
16
17             self._head = None
18
19             if list_to_convert:
20                 for item in list_to_convert:
21                     self.insert_at_end(item)
22
23     def __str__(self):
24         """
25             Returns a string representation of the circular linked list.
26
27             Returns:
28                 str: Linked list formatted as string.
29                 """
30
31             if self._head is None:
32                 return '| empty |'
33
34             characters = '| -> '
35             current_node = self._head
36             while True:
37                 characters += str(current_node.data) + ' -> '
38                 current_node = current_node.next
39
40             if current_node == self._head:
41                 characters += '|'
42             return characters
43
44     def insert_at_beginning(self, data):
45         """
46             Inserts a node at the beginning of the circular linked list.
47
48             Args:
49                 data: The data to insert.
50                 """
51
52             new_node = Node(data)
53
54             if self._head is None:
55                 self._head = new_node
56                 new_node.next = self._head
57                 new_node.previous = self._head
58             else:
59                 new_node.next = self._head

```

```

56         new_node.previous = self._head.previous
57         self._head.previous.next = new_node
58         self._head.previous = new_node
59
60         self._head = new_node
61
62     def insert_at_end(self, data):
63         """
64             Inserts a node at the end of the circular linked list.
65
66         Args:
67             data: The data to insert.
68         """
69         new_node = Node(data)
70
71         if self._head is None:
72             self._head = new_node
73             new_node.next = self._head
74             new_node.previous = self._head
75         else:
76             new_node.next = self._head
77             new_node.previous = self._head.previous
78             self._head.previous.next = new_node
79             self._head.previous = new_node
80
81     def insert_at_index(self, data, index):
82         """
83             Inserts a node at a specific index in the circular linked list.
84             The head node is taken as index 0.
85
86         Args:
87             data: The data to insert.
88             index (int): The index to insert the data at.
89
90         Raises:
91             ValueError: Index is out of range.
92         """
93         if index < 0:
94             raise ValueError('Invalid index! (CircularLinkedList.insert_at_index)')
95
96         if index == 0 or self._head is None:
97             self.insert_at_beginning(data)
98         else:
99             new_node = Node(data)
100            current_node = self._head
101            count = 0
102
103            while count < index - 1 and current_node.next != self._head:
104                current_node = current_node.next
105                count += 1
106
107            if count == (index - 1):
108                new_node.next = current_node.next
109                new_node.previous = current_node
110                current_node.next = new_node
111            else:
112                raise ValueError('Index out of range! (CircularLinkedList.
113                insert_at_index)')
114
115     def delete(self, data):
116         """

```

```
116     Deletes a node with the specified data from the circular linked list.
117
118     Args:
119         data: The data to delete.
120
121     Raises:
122         ValueError: No nodes in the list contain the specified data.
123     """
124     if self._head is None:
125         return
126
127     current_node = self._head
128
129     while current_node.data != data:
130         current_node = current_node.next
131
132         if current_node == self._head:
133             raise ValueError('Data not found in circular linked list! (' +
CircularLinkedList.delete)')
134
135         if self._head.next == self._head:
136             self._head = None
137         else:
138             current_node.previous.next = current_node.next
139             current_node.next.previous = current_node.previous
140
141     def data_in_list(self, data):
142         """
143             Checks if the specified data is in the circular linked list.
144
145             Args:
146                 data: The data to check.
147
148             Returns:
149                 bool: True if the data is in the list, False otherwise.
150         """
151         if self._head is None:
152             return False
153
154         current_node = self._head
155         while True:
156             if current_node.data == data:
157                 return True
158
159             current_node = current_node.next
160             if current_node == self._head:
161                 return False
162
163     def shift_head(self):
164         """
165             Shifts the head of the circular linked list to the next node.
166         """
167         self._head = self._head.next
168
169     def unshift_head(self):
170         """
171             Shifts the head of the circular linked list to the previous node.
172         """
173         self._head = self._head.previous
174
175     def get_head(self):
176         """
```

```

177     Gets the head node of the circular linked list.
178
179     Returns:
180         Node: The head node.
181     """
182     return self._head

```

### 3.4.4 Widgets

As described in Section 2.4, each state contains a `WIDGET_DICT` map, which contains and initialises each widget with their own attributes, and provides references to run methods on them in the state code. Each `WIDGET_DICT` is passed into a `WidgetGroup` object, which is responsible for drawing, resizing and handling all widgets for the current state. Below is a list of all the widgets I have implemented (See Section B.25):

- BoardThumbnailButton
- MultipleIconButton
- ReactiveIconButton
- BoardThumbnail
- ReactiveButton
- VolumeSlider
- ColourPicker
- ColourButton
- BrowserStrip
- PieceDisplay
- BrowserItem
- TextButton
- IconButton
- ScrollArea
- Chessboard
- TextInput
- Rectangle
- MoveList
- Dropdown
- Carousel
- Switch
- Timer
- Text
- Icon
- (`_ColourDisplay`)
- (`_ColourSquare`)
- (`_ColourSlider`)
- (`_SliderThumb`)
- (`_Scrollbar`)

### CustomEvent

The `CustomEvent` class is used to pass data between states and widgets. An event argument is passed into interactive widgets; When a widget wants to pass data back to the state, it returns the event, and adds any attributes that is required. The state then receives and handles these returned events accordingly.

#### `custom_event.py`

```

1 from data.utils.event_types import GameEventType, SettingsEventType,
   ConfigEventType, BrowserEventType, EditorEventType
2
3 # Required keyword arguments when creating a CustomEvent object with a specific
   EventType
4 required_args = {
5     GameEventType.BOARD_CLICK: ['coords'],
6     GameEventType.ROTATE PIECE: ['rotation_direction'],
7     GameEventType.SET LASER: ['laser_result'],
8     GameEventType.UPDATE_PIECES: ['move_notation'],
9     GameEventType.TIMER_END: ['active_colour'],
10    GameEventType.PIECE_DROP: ['coords', 'piece', 'colour', 'rotation',
11      'remove_overlay'],
11    SettingsEventType.COLOUR_SLIDER_SLIDE: ['colour'],

```

```

12     SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK: ['colour'],
13     SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK: ['colour'],
14     SettingsEventType.DROPODOWN_CLICK: ['selected_word'],
15     SettingsEventType.VOLUME_SLIDER_CLICK: ['volume', 'volume_type'],
16     SettingsEventType.SHADER_PICKER_CLICK: ['data'],
17     SettingsEventType.PARTICLES_CLICK: ['toggled'],
18     SettingsEventType.OPENGL_CLICK: ['toggled'],
19     ConfigEventType.TIME_TYPE: ['time'],
20     ConfigEventType.FEN_STRING_TYPE: ['time'],
21     ConfigEventType.CPU_DEPTH_CLICK: ['data'],
22     ConfigEventType.PVC_CLICK: ['data'],
23     ConfigEventType.PRESET_CLICK: ['fen_string'],
24     BrowserEventType.BROWSER_STRIP_CLICK: ['selected_index'],
25     BrowserEventType.PAGE_CLICK: ['data'],
26     EditorEventType.PICK_PIECE_CLICK: ['piece', 'active_colour'],
27     EditorEventType.ROTATE_PIECE_CLICK: ['rotation_direction'],
28 }
29
30 class CustomEvent():
31     def __init__(self, type, **kwargs):
32         self.__dict__.update(kwargs)
33         self.type = type
34
35     @classmethod
36     def create_event(event_cls, event_type, **kwargs):
37         """
38             @classmethod Factory method used to instance CustomEvent object, to check
39             for required keyword arguments
40
41             Args:
42                 event_cls (CustomEvent): Reference to own class.
43                 event_type: The state EventType.
44
45             Raises:
46                 ValueError: If required keyword argument for passed event type not
47                 present.
48                 ValueError: If keyword argument passed is not required for passed
49                 event type.
50
51             Returns:
52                 CustomEvent: Initialised CustomEvent instance.
53         """
54         if event_type in required_args:
55
56             for required_arg in required_args[event_type]:
57                 if required_arg not in kwargs:
58                     raise ValueError(f"Argument '{required_arg}' required for {event_type.name} event (GameEvent.create_event)")
59
60             for kwarg in kwargs:
61                 if kwarg not in required_args[event_type]:
62                     raise ValueError(f"Argument '{kwarg}' not included in
63             required_args dictionary for event '{event_type}'! (GameEvent.create_event)")
64
65         return event_cls(event_type, **kwargs)
66
67     else:
68         return event_cls(event_type)

```

### ReactiveIconButton

The `ReactiveIconButton` widget is a pressable button that changes the icon displayed when it is hovered or pressed.

`reactive_icon_button.py`

```

1 from data.widgets.reactive_button import ReactiveButton
2 from data.utils.constants import WidgetState
3 from data.widgets.icon import Icon
4
5 class ReactiveIconButton(ReactiveButton):
6     def __init__(self, base_icon, hover_icon, press_icon, **kwargs):
7         # Composition is used here, to initialise the Icon widgets for each widget
8         state
9             widgets_dict = {
10                 WidgetState.BASE: Icon(
11                     parent=kwargs.get('parent'),
12                     relative_size=kwargs.get('relative_size'),
13                     relative_position=(0, 0),
14                     icon=base_icon,
15                     fill_colour=(0, 0, 0, 0),
16                     border_width=0,
17                     margin=0,
18                     fit_icon=True,
19                 ),
20                 WidgetState.HOVER: Icon(
21                     parent=kwargs.get('parent'),
22                     relative_size=kwargs.get('relative_size'),
23                     relative_position=(0, 0),
24                     icon=hover_icon,
25                     fill_colour=(0, 0, 0, 0),
26                     border_width=0,
27                     margin=0,
28                     fit_icon=True,
29                 ),
30                 WidgetState.PRESS: Icon(
31                     parent=kwargs.get('parent'),
32                     relative_size=kwargs.get('relative_size'),
33                     relative_position=(0, 0),
34                     icon=press_icon,
35                     fill_colour=(0, 0, 0, 0),
36                     border_width=0,
37                     margin=0,
38                     fit_icon=True,
39                 )
40             }
41             super().__init__(
42                 widgets_dict=widgets_dict,
43                 **kwargs
44             )

```

### ReactiveButton

The `ReactiveButton` widget is the parent class for `ReactiveIconButton`. It provides the methods for clicking, rotating between widget states, positioning etc.

`reactive_button.py`

```

1 from data.components.custom_event import CustomEvent
2 from data.widgets.bases.pressable import _Pressable

```

```
3 from data.widgets.bases.circular import _Circular
4 from data.widgets.bases.widget import _Widget
5 from data.utils.constants import WidgetState
6
7 class ReactiveButton(_Pressable, _Circular, _Widget):
8     def __init__(self, widgets_dict, event, center=False, **kwargs):
9         # Multiple inheritance used here, to combine the functionality of multiple
10        super().__init__(
11            self,
12            event=event,
13            hover_func=lambda: self.set_to_key(WidgetState.HOVER),
14            down_func=lambda: self.set_to_key(WidgetState.PRESS),
15            up_func=lambda: self.set_to_key(WidgetState.BASE),
16            **kwargs
17        )
18        # Aggregation used to cycle between external widgets
19        _Circular.__init__(self, items_dict=widgets_dict)
20        _Widget.__init__(self, **kwargs)
21
22        self._center = center
23
24        self.initialise_new_colours(self._fill_colour)
25
26    @property
27    def position(self):
28        """
29            Overrides position getter method, to always position icon in the center if
30            self._center is True.
31
32            Returns:
33            list[int, int]: Position of widget.
34        """
35        position = super().position
36
37        if self._center:
38            self._size_diff = (self.size[0] - self.rect.width, self.size[1] - self.
39                rect.height)
40            return (position[0] + self._size_diff[0] / 2, position[1] + self.
41                _size_diff[1] / 2)
42        else:
43            return position
44
45    def set_image(self):
46        """
47            Sets current icon to image.
48        """
49        self.current_item.set_image()
50        self.image = self.current_item.image
51
52    def set_geometry(self):
53        """
54            Sets size and position of widget.
55        """
56        super().set_geometry()
57        self.current_item.set_geometry()
58        self.current_item.rect.topleft = self.rect.topleft
59
60    def set_surface_size(self, new_surface_size):
61        """
62            Overrides base method to resize every widget state icon, not just the
63            current one.
64
```

```

60
61     Args:
62         new_surface_size (list[int, int]): New surface size.
63
64     super().set_surface_size(new_surface_size)
65     for item in self._items_dict.values():
66         item.set_surface_size(new_surface_size)
67
68     def process_event(self, event):
69         """
70             Processes Pygame events.
71
72         Args:
73             event (pygame.Event): Event to process.
74
75         Returns:
76             CustomEvent: CustomEvent of current item, with current key included
77
78         widget_event = super().process_event(event)
79         self.current_item.process_event(event)
80
81         if widget_event:
82             return CustomEvent(**vars(widget_event), data=self.current_key)

```

## ColourSlider

The ColourSlider widget is instanced in the ColourPicker class. It provides a slider for changing between hues for the colour picker, using the functionality of the SliderThumb class.

`colour_slider.py`

```

1 import pygame
2 from data.helpers.widget_helpers import create_slider_gradient
3 from data.helpers.asset_helpers import smoothscale_and_cache
4 from data.widgets.slider_thumb import _SliderThumb
5 from data.widgets.bases.widget import _Widget
6 from data.utils.constants import WidgetState
7
8 class _ColourSlider(_Widget):
9     def __init__(self, relative_width, **kwargs):
10         super().__init__(relative_size=(relative_width, relative_width * 0.2), **
11                         kwargs)
12         # Initialise slider thumb.
13         self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
14                                     _border_colour)
15         self._selected_percent = 0
16         self._last_mouse_x = None
17
18         self._gradient_surface = create_slider_gradient(self.gradient_size, self.
19             border_width, self._border_colour)
20         self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
21
22     @property
23     def gradient_size(self):
24         return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
25
26     @property
27     def gradient_position(self):
28         return (self.size[1] / 2, self.size[1] / 4)

```

```
29     @property
30     def thumb_position(self):
31         return (self.gradient_size[0] * self._selected_percent, 0)
32
33     @property
34     def selected_colour(self):
35         colour = pygame.Color(0)
36         colour.hsva = (int(self._selected_percent * 360), 100, 100, 100)
37         return colour
38
39     def calculate_gradient_percent(self, mouse_pos):
40         """
41             Calculate what percentage slider thumb is at based on change in mouse
42             position.
43
44             Args:
45                 mouse_pos (list[int, int]): Position of mouse on window screen.
46
47             Returns:
48                 float: Slider scroll percentage.
49
50             if self._last_mouse_x is None:
51                 return
52
53             x_change = (mouse_pos[0] - self._last_mouse_x) / (self.gradient_size[0] -
54             2 * self.border_width)
55             return max(0, min(self._selected_percent + x_change, 1))
56
57     def relative_to_global_position(self, position):
58         """
59             Transforms position from being relative to widget rect, to window screen.
60
61             Args:
62                 position (list[int, int]): Position relative to widget rect.
63
64             Returns:
65                 list[int, int]: Position relative to window screen.
66
67             relative_x, relative_y = position
68             return (relative_x + self.position[0], relative_y + self.position[1])
69
70     def set_colour(self, new_colour):
71         """
72             Sets selected_percent based on the new colour's hue.
73
74             Args:
75                 new_colour (pygame.Color): New slider colour.
76
77             colour = pygame.Color(new_colour)
78             hue = colour.hsva[0]
79             self._selected_percent = hue / 360
80             self.set_image()
81
82     def set_image(self):
83         """
84             Draws colour slider to widget image.
85
86             # Scales initialised gradient surface instead of redrawing it everytime
87             set_image is called
88             gradient_scaled = smoothscale_and_cache(self._gradient_surface, self.
89             gradient_size)
```

```
87         self.image = pygame.transform.scale(self._empty_surface, (self.size))
88         self.image.blit(gradient_scaled, self.gradient_position)
89
90     # Resets thumb colour, image and position, then draws it to the widget
91     # image
92     self._thumb.initialise_new_colours(self.selected_colour)
93     self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
94     border_width)
95     self._thumb.set_position(self.relative_to_global_position((self.
96     thumb_position[0], self.thumb_position[1])))
97
98     thumb_surface = self._thumb.get_surface()
99     self.image.blit(thumb_surface, self.thumb_position)
100
101
102     def process_event(self, event):
103         """
104
105         Processes Pygame events.
106
107         Args:
108             event (pygame.Event): Event to process.
109
110         Returns:
111             pygame.Color: Current colour slider is displaying.
112         """
113
114     if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
115     MOUSEBUTTONUP]:
116         return
117
118     # Gets widget state before and after event is processed by slider thumb
119     before_state = self._thumb.state
120     self._thumb.process_event(event)
121     after_state = self._thumb.state
122
123     # If widget state changes (e.g. hovered -> pressed), redraw widget
124     if before_state != after_state:
125         self.set_image()
126
127     if event.type == pygame.MOUSEMOTION:
128         if self._thumb.state == WidgetState.PRESS:
129             # Recalculates slider colour based on mouse position change
130             selected_percent = self.calculate_gradient_percent(event.pos)
131             self._last_mouse_x = event.pos[0]
132
133             if selected_percent is not None:
134                 self._selected_percent = selected_percent
135
136             return self.selected_colour
137
138     if event.type == pygame.MOUSEBUTTONUP:
139         # When user stops scrolling, return new slider colour
140         self._last_mouse_x = None
141         return self.selected_colour
142
143     if event.type == pygame.MOUSEBUTTONDOWN or before_state != after_state:
144         # Redraws widget when slider thumb is hovered or pressed
145         return self.selected_colour
```

### TextInput

The `TextInput` widget is used for inputting fen strings and time controls.

**text\_input.py**

```
1 import pyperclip
2 import pygame
3 from data.utils.constants import WidgetState, INPUT_COLOURS
4 from data.components.custom_event import CustomEvent
5 from data.widgets.bases.pressable import _Pressable
6 from data.managers.logs import initialise_logger
7 from data.managers.animation import animation
8 from data.widgets.bases.box import _Box
9 from data.utils.enums import CursorMode
10 from data.managers.cursor import cursor
11 from data.managers.theme import theme
12 from data.widgets.text import Text
13
14 logger = initialise_logger(__name__)
15
16 class TextInput(_Box, _Pressable, Text):
17     def __init__(self, event, blinking_interval=530, validator=(lambda x: True),
18                  default='', placeholder='PLACEHOLDER TEXT', placeholder_colour=(200, 200, 200),
19                  , cursor_colour=theme['textSecondary'], **kwargs):
20         self._cursor_index = None
21         # Multiple inheritance used here, adding the functionality of pressing,
22         # and custom box colours, to the text widget
23         _Box.__init__(self, box_colours=INPUT_COLOURS)
24         _Pressable.__init__(
25             self,
26             event=None,
27             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
28             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
29             up_func=lambda: self.set_state_colour(WidgetState.BASE),
30             sfx=None
31         )
32         Text.__init__(self, text="", center=False, box_colours=INPUT_COLOURS[
33             WidgetState.BASE], **kwargs)
34
35         self.initialise_new_colours(self._fill_colour)
36         self.set_state_colour(WidgetState.BASE)
37
38         pygame.key.set_repeat(500, 50)
39
40         self._blinking_fps = 1000 / blinking_interval
41         self._cursor_colour = cursor.colour
42         self._cursor_colour_copy = cursor.colour
43         self._placeholder_colour = placeholder.colour
44         self._text.colour_copy = self._text.colour
45
46         self._placeholder_text = placeholder
47         self._is_placeholder = None
48
49         if default:
50             self._text = default
51             self.is_placeholder = False
52         else:
53             self._text = self._placeholder_text
54             self.is_placeholder = True
55
56         self._event = event
57         self._validator = validator
58         self._blinking_cooldown = 0
59
60         self._empty_cursor = pygame.Surface((0, 0), pygame.SRCALPHA)
```

```
57         self._resize_text()
58         self._set_image()
59         self._set_geometry()
60
61     @property
62     # Encapsulated getter method
63     def is_placeholder(self):
64         return self._is_placeholder
65
66     @is_placeholder.setter
67     # Encapsulated setter method, used to replace text colour if placeholder text
68     # is shown
69     def is_placeholder(self, is_true):
70         self._is_placeholder = is_true
71
72         if is_true:
73             self._text_colour = self._placeholder_colour
74         else:
75             self._text_colour = self._text_colour_copy
76
77     @property
78     def cursor_size(self):
79         cursor_height = (self.size[1] - self.border_width * 2) * 0.75
80         return (cursor_height * 0.1, cursor_height)
81
82     @property
83     def cursor_position(self):
84         current_width = (self.margin / 2)
85         for index, metrics in enumerate(self._font.get_metrics(self._text, size=
86             self.font_size)):
87             if index == self._cursor_index:
88                 return (current_width - self.cursor_size[0], (self.size[1] - self.
89                     cursor_size[1]) / 2)
90
91     @property
92     def text(self):
93         if self.is_placeholder:
94             return ''
95
96         return self._text
97
98     def relative_x_to_cursor_index(self, relative_x):
99         """
100             Calculates cursor index using mouse position relative to the widget
101             position.
102
103             Args:
104                 relative_x (int): Horizontal distance of the mouse from the left side
105                 of the widget.
106
107             Returns:
108                 int: Cursor index.
109
110             current_width = 0
111
112             for index, metrics in enumerate(self._font.get_metrics(self._text, size=
113                 self.font_size)):
```

```
112         glyph_width = metrics[4]
113
114     if current_width >= relative_x:
115         return index
116
117     current_width += glyph_width
118
119     return len(self._text)
120
121 def set_cursor_index(self, mouse_pos):
122     """
123     Sets cursor index based on mouse position.
124
125     Args:
126         mouse_pos (list[int, int]): Mouse position relative to window screen.
127     """
128
129     if mouse_pos is None:
130         self._cursor_index = mouse_pos
131         return
132
133     relative_x = mouse_pos[0] - (self.margin / 2) - self.rect.left
134     relative_x = max(0, relative_x)
135     self._cursor_index = self.relative_x_to_cursor_index(relative_x)
136
137 def focus_input(self, mouse_pos):
138     """
139     Draws cursor and sets cursor index when user clicks on widget.
140
141     Args:
142         mouse_pos (list[int, int]): Mouse position relative to window screen.
143     """
144
145     if self.is_placeholder:
146         self._text = ''
147         self.is_placeholder = False
148
149     self.set_cursor_index(mouse_pos)
150     self.set_image()
151     cursor.set_mode(CursorMode.IBEAM)
152
153 def unfocus_input(self):
154     """
155     Removes cursor when user unselects widget.
156
157     If self._text == '':
158         self._text = self._placeholder_text
159         self.is_placeholder = True
160         self.resize_text()
161
162     self.set_cursor_index(None)
163     self.set_image()
164     cursor.set_mode(CursorMode.ARROW)
165
166 def set_text(self, new_text):
167     """
168     Called by a state object to change the widget text externally.
169
170     Args:
171         new_text (str): New text to display.
172
173     Returns:
174         CustomEvent: Object containing the new text to alert state of a text
175         update.
```

```

173     """
174     super().set_text(new_text)
175     return CustomEvent(**vars(self._event), text=self.text)
176
177     def process_event(self, event):
178         """
179         Processes Pygame events.
180
181         Args:
182             event (pygame.Event): Event to process.
183
184         Returns:
185             CustomEvent: Object containing the new text to alert state of a text
186             update.
187         """
188         previous_state = self.get_widget_state()
189         super().process_event(event)
190         current_state = self.get_widget_state()
191
192         match event.type:
193             case pygame.MOUSEMOTION:
194                 if self._cursor_index is None:
195                     return
196
197                 # If mouse is hovering over widget, turn mouse cursor into an I-
198                 # beam
199                 if self.rect.collidepoint(event.pos):
200                     if cursor.get_mode() != CursorMode.IBEAM:
201                         cursor.set_mode(CursorMode.IBEAM)
202                 else:
203                     if cursor.get_mode() == CursorMode.IBEAM:
204                         cursor.set_mode(CursorMode.ARROW)
205
206             return
207
208             case pygame.MOUSEBUTTONDOWN:
209                 # When user selects widget
210                 if previous_state == WidgetState.PRESS:
211                     self.focus_input(event.pos)
212
213                 # When user unselects widget
214                 if current_state == WidgetState.BASE and self._cursor_index is not
215                 None:
216                     self.unfocus_input()
217                     return CustomEvent(**vars(self._event), text=self.text)
218
219             case pygame.KEYDOWN:
220                 if self._cursor_index is None:
221                     return
222
223                 # Handling Ctrl-C and Ctrl-V shortcuts
224                 if event.mod & (pygame.KMOD_CTRL):
225                     if event.key == pygame.K_c:
226                         pyperclip.copy(self.text)
227                         logger.info(f'COPIED {self.text}')
228
229                     elif event.key == pygame.K_v:
230                         pasted_text = pyperclip.paste()
231                         pasted_text = ''.join(char for char in pasted_text if 32
232                         <= ord(char) <= 127)
233                         self._text = self._text[:self._cursor_index] + pasted_text
234                         + self._text[self._cursor_index:]
235                         self._cursor_index += len(pasted_text)

```

```

230
231             elif event.key == pygame.K_BACKSPACE or event.key == pygame.
232                 K_DELETE:
233                     self._text = ''
234                     self._cursor_index = 0
235
236                     self.resize_text()
237                     self.set_image()
238                     self.set_geometry()
239
240             return
241
242         match event.key:
243             case pygame.K_BACKSPACE:
244                 if self._cursor_index > 0:
245                     self._text = self._text[:self._cursor_index - 1] +
246                     self._text[self._cursor_index:]
247                     self._cursor_index = max(0, self._cursor_index - 1)
248
249             case pygame.K_RIGHT:
250                 self._cursor_index = min(len(self._text), self.
251 _cursor_index + 1)
252
253             case pygame.K_LEFT:
254                 self._cursor_index = max(0, self._cursor_index - 1)
255
256             case pygame.K_ESCAPE:
257                 self.unfocus_input()
258                 return CustomEvent(**vars(self._event), text=self.text)
259
260             case pygame.K_RETURN:
261                 self.unfocus_input()
262                 return CustomEvent(**vars(self._event), text=self.text)
263
264             case _:
265                 if not event.unicode:
266                     return
267
268                 potential_text = self._text[:self._cursor_index] + event.
269                 unicode + self._text[self._cursor_index:]
270
271                 # Validator lambda function used to check if inputted text
272                 # is valid before displaying
273                 # e.g. Time control input has a validator function
274                 # checking if text represents a float
275                 if self._validator(potential_text) is False:
276                     return
277
278                 self._text = potential_text
279                 self._cursor_index += 1
280
281                 self._blinking_cooldown += 1
282                 animation.set_timer(500, lambda: self.subtract_blinking_cooldown
283 (1))
284
285             self.resize_text()
286             self.set_image()
287             self.set_geometry()
288
289     def subtract_blinking_cooldown(self, cooldown):
290         """
291             Subtracts blinking cooldown after certain timeframe. When

```

```

    blinking_cooldown is 1, cursor is able to be drawn.

285
286     Args:
287         cooldown (float): Duration before cursor can no longer be drawn.
288         """
289         self._blinking_cooldown = self._blinking_cooldown - cooldown
290
291     def set_image(self):
292         """
293             Draws text input widget to image.
294         """
295         super().set_image()
296
297         if self._cursor_index is not None:
298             scaled_cursor = pygame.transform.scale(self._empty_cursor, self.
299             cursor_size)
300             scaled_cursor.fill(self._cursor_colour)
301             self.image.blit(scaled_cursor, self.cursor_position)
302
303     def update(self):
304         """
305             Overrides based update method, to handle cursor blinking.
306         """
307         super().update()
308         # Calculate if cursor should be shown or not
309         cursor_frame = animation.calculate_frame_index(0, 2, self._blinking_fps)
310         if cursor_frame == 1 and self._blinking_cooldown == 0:
311             self._cursor_colour = (0, 0, 0, 0)
312         else:
313             self._cursor_colour = self._cursor_colour_copy
314         self.set_image()

```

## 3.5 Game

### 3.5.1 Model

As described in Section 2.4, this is the model class for my implementation of a **MVC architecture** for the game screen. It is responsible for processing user inputs through the game controller, processing the board and CPU, and sending information through the view class.

`game_model.py`

```

1  from random import getrandbits
2  from data.states.game.components.fen_parser import encode_fen_string
3  from data.states.game.widget_dict import GAME_WIDGETS
4  from data.states.game.cpu.cpu_thread import CPUTHread
5  from data.components.custom_event import CustomEvent
6  from data.helpers.bitboard_helpers import is_occupied
7  from data.helpers import input_helpers as ip_helpers
8  from data.states.game.components.board import Board
9  from data.states.game.components.move import Move
10 from data.utils.event_types import GameEventType
11 from data.managers.logs import initialise_logger
12 from data.managers.animation import animation
13 from data.states.game.cpu.engines import *
14 from data.utils.constants import EMPTY_BB
15 from data.utils.enums import Colour
16
17 logger = initialise_logger(__name__)
18

```

```

19 CPU_LIMIT_MS = 15000
20
21 class GameModel:
22     def __init__(self, game_config):
23         self._listeners = {
24             'game': [],
25             'win': [],
26             'pause': []
27         }
28         self.states = {
29             'CPU_ENABLED': game_config['CPU_ENABLED'],
30             'CPU_DEPTH': game_config['CPU_DEPTH'],
31             'AWAITING_CPU': False,
32             'WINNER': None,
33             'PAUSED': False,
34             'ACTIVE_COLOUR': game_config['COLOUR'],
35             'TIME_ENABLED': game_config['TIME_ENABLED'],
36             'TIME': game_config['TIME'],
37             'START_FEN_STRING': game_config['FEN_STRING'],
38             'MOVES': [],
39             'ZOBRIST_KEYS': []
40         }
41
42         self._board = Board(fen_string=game_config['FEN_STRING'])
43
44         self._cpu = IDMinimaxCPU(self.states['CPU_DEPTH'], self.cpu_callback,
45         verbose=False)
46         self._cpu_thread = CPUPhase(self._cpu)
47         self._cpu_thread.start()
48         self._cpu_move = None
49
50         logger.info(f'Initialising CPU depth of {self.states["CPU_DEPTH"]}')
51
52     def register_listener(self, listener, parent_class):
53         """
54         Registers listener method of another MVC class.
55
56         Args:
57             listener (callable): Listener callback function.
58             parent_class (str): Class name.
59         """
60         self._listeners[parent_class].append(listener)
61
62     def alert_listeners(self, event):
63         """
64         Alerts all registered classes of an event by calling their listener
65         function.
66
67         Args:
68             event (GameEventType): Event to pass as argument.
69
70         Raises:
71             Exception: If an unrecognised event tries to be passed onto listeners.
72         """
73         for parent_class, listeners in self._listeners.items():
74             match event.type:
75                 case GameEventType.UPDATE_PIECES:
76                     if parent_class in 'game':
77                         for listener in listeners: listener(event)
78
79                 case GameEventType.SET_LASER:
80                     if parent_class == 'game':
81
82

```

```

79                     for listener in listeners: listener(event)
80
81             case GameEventType.PAUSE_CLICK:
82                 if parent_class in ['pause', 'game']:
83                     for listener in listeners:
84                         listener(event)
85
86             case _:
87                 raise Exception('Unhandled event type (GameModel.
88 alert_listeners)')
89
90     def set_winner(self, colour=None):
91         """
92             Sets winner.
93
94             Args:
95                 colour (Colour, optional): Describes winner colour, or draw. Defaults
96                 to None.
97
98         self.states['WINNER'] = colour
99
100    def toggle_paused(self):
101        """
102            Toggles pause screen, and alerts pause view.
103
104        self.states['PAUSED'] = not self.states['PAUSED']
105        game_event = CustomEvent.create_event(GameEventType.PAUSE_CLICK)
106        self.alert_listeners(game_event)
107
108    def get_terminal_move(self):
109        """
110            Debugging method for inputting a move from the terminal.
111
112            Returns:
113                Move: Parsed move.
114
115        while True:
116            try:
117                move_type = ip_helpers.parse_move_type(input('Input move type (m/r
118 ): '))
119                src_square = ip_helpers.parse_notation(input("From: "))
120                dest_square = ip_helpers.parse_notation(input("To: "))
121                rotation = ip_helpers.parse_rotation(input("Enter rotation (a/b/c/
122 d): "))
123                return Move.instance_from_notation(move_type, src_square,
124                dest_square, rotation)
125            except ValueError as error:
126                logger.warning('Input error (Board.get_move): ' + str(error))
127
128    def make_move(self, move):
129        """
130            Takes a Move object and applies it to the board.
131
132            Args:
133                move (Move): Move to apply.
134
135            colour = self._board.bitboards.get_colour_on(move.src)
136            piece = self._board.bitboards.get_piece_on(move.src, colour)
137            # Apply move and get results of laser trajectory
138            laser_result = self._board.apply_move(move, add_hash=True)
139
140            self.alert_listeners(CustomEvent.create_event(GameEventType.SET_LASER,

```

```

        laser_result=laser_result))

136
137     # Sets new active colour and checks for a win
138     self.states['ACTIVE_COLOUR'] = self._board.get_active_colour()
139     self.set_winner(self._board.check_win())
140
141     move_notation = move.to_notation(colour, piece, laser_result,
142                                       hit_square_bitboard)
143
144     self.alert_listeners(CustomEvent.create_event(GameEventType.UPDATE_PIECES,
145                                               move_notation=move_notation))
146
147     # Adds move to move history list for review screen
148     self.states['MOVES'].append({
149         'time': {
150             Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
151             Colour.RED: GAME_WIDGETS['red_timer'].get_time()
152         },
153         'move': move_notation,
154         'laserResult': laser_result
155     })
156
157 def make_cpu_move(self):
158     """
159     Starts CPU calculations on the separate thread.
160     """
161     self.states['AWAITING_CPU'] = True
162
163     # Employ time management system to kill search if using an iterative
164     # deepening CPU
165     if isinstance(self._cpu, IDMinimaxCPU):
166         move_id = getrandbits(32)
167         self._cpu_thread.start_cpu(self.get_board(), id=move_id)
168         animation.set_timer(CPU_LIMIT_MS, lambda: self._cpu_thread.stop_cpu(id
169 =move_id))
170     else:
171         self._cpu_thread.start_cpu(self.get_board())
172
173 def cpu_callback(self, move):
174     """
175     Callback function passed to CPU thread. Called when CPU stops processing.
176
177     Args:
178         move (Move): Move that CPU found.
179
180     if self.states['WINNER'] is None:
181         # CPU move passed back to main thread by reassigning variable
182         self._cpu_move = move
183         self.states['AWAITING_CPU'] = False
184
185     def check_cpu(self):
186         """
187             Constantly checks if CPU calculations are finished, so that make_move can
188             be run on the main thread.
189
190             if self._cpu_move is not None:
191                 self.make_move(self._cpu_move)
192                 self._cpu_move = None
193
194     def kill_thread(self):
195         """
196             Interrupt and kill CPU thread.

```

```

192     """
193     self._cpu_thread.kill_thread()
194     self.states['AWAITING_CPU'] = False
195
196     def is_selectable(self, bitboard):
197         """
198             Checks if square is occupied by a piece of the current active colour.
199
200             Args:
201                 bitboard (int): Bitboard representing single square.
202
203             Returns:
204                 bool: True if square is occupied by a piece of the current active
205                 colour. False if not.
206         """
207
208         return is_occupied(self._board.bitboards.combined_colour_bitboards[self.
209                         states['ACTIVE_COLOUR']], bitboard)
209
210     def get_available_moves(self, bitboard):
211         """
212             Gets all surrounding empty squares. Used for drawing overlay.
213
214             Args:
215                 bitboard (int): Bitboard representing single center square.
216
217             Returns:
218                 int: Bitboard representing all empty surrounding squares.
219         """
220
221         if (bitboard & self._board.get_all_active_pieces()) != EMPTY_BB:
222             return self._board.get_valid_squares(bitboard)
223
224         return EMPTY_BB
225
226     def get_piece_list(self):
227         """
228             Returns:
229                 list[Piece, ...]: Array of all pieces on the board.
230
231         return self._board.get_piece_list()
232
233     def get_piece_info(self, bitboard):
234         """
235             Args:
236                 bitboard (int): Square containing piece.
237
238             Returns:
239                 tuple[Colour, Rotation, Piece]: Piece information.
240
241         colour = self._board.bitboards.get_colour_on(bitboard)
242         rotation = self._board.bitboards.get_rotation_on(bitboard)
243         piece = self._board.bitboards.get_piece_on(bitboard, colour)
244         return (piece, colour, rotation)
245
246     def get_fen_string(self):
247         return encode_fen_string(self._board.bitboards)
248
249     def get_board(self):
250         return self._board

```

### 3.5.2 View

As described in Section 2.4, the view class is responsible for displaying changes to information regarding the gameplay. The `process_model_event` procedure is registered with the model class, which executes it whenever the display needs to be updated (e.g. piece move), and the appropriate handling function within the view class is called by mapping the event type to the corresponding handler function.

`game_view.py`

```

1 import pygame
2 from data.utils.enums import Colour, StatusText, Miscellaneous, ShaderType
3 from data.states.game.components.overlay_draw import OverlayDraw
4 from data.states.game.components.capture_draw import CaptureDraw
5 from data.states.game.components.piece_group import PieceGroup
6 from data.states.game.components.laser_draw import LaserDraw
7 from data.states.game.components.father import DragAndDrop
8 from data.helpers.bitboard_helpers import bitboard_to_coords
9 from data.helpers.board_helpers import screen_pos_to_coords
10 from data.states.game.widget_dict import GAME_WIDGETS
11 from data.components.custom_event import CustomEvent
12 from data.components.widget_group import WidgetGroup
13 from data.utils.event_types import GameEventType
14 from data.managers.window import window
15 from data.managers.audio import audio
16 from data.utils.assets import SFX
17
18 class GameView:
19     def __init__(self, model):
20         self._model = model
21         self._hide_pieces = False
22         self._selected_coords = None
23         self._event_to_func_map = {
24             GameEventType.UPDATE_PIECES: self.handle_update_pieces,
25             GameEventType.SET LASER: self.handle_set_laser,
26             GameEventType.PAUSE_CLICK: self.handle_pause,
27         }
28
29         # Register model event handling with process_model_event()
30         self._model.register_listener(self.process_model_event, 'game')
31
32         # Initialise WidgetGroup with map of widgets
33         self._widget_group = WidgetGroup(GAME_WIDGETS)
34         self._widget_group.handle_resize(window.size)
35         self.initialise_widgets()
36
37         self._laser_draw = LaserDraw(self.board_position, self.board_size)
38         self._overlay_draw = OverlayDraw(self.board_position, self.board_size)
39         self._drag_and_drop = DragAndDrop(self.board_position, self.board_size)
40         self._capture_draw = CaptureDraw(self.board_position, self.board_size)
41         self._piece_group = PieceGroup()
42         self.handle_update_pieces()
43
44         self.set_status_text(StatusText.PLAYER_MOVE)
45
46     @property
47     def board_position(self):
48         return GAME_WIDGETS['chessboard'].position
49
50     @property
51     def board_size(self):
52         return GAME_WIDGETS['chessboard'].size

```

```

53
54     @property
55     def square_size(self):
56         return self.board_size[0] / 10
57
58     def initialise_widgets(self):
59         """
60             Run methods on widgets stored in GAME_WIDGETS dictionary to reset them.
61         """
62         GAME_WIDGETS['move_list'].reset_move_list()
63         GAME_WIDGETS['move_list'].kill()
64         GAME_WIDGETS['help'].kill()
65         GAME_WIDGETS['tutorial'].kill()
66
67         GAME_WIDGETS['scroll_area'].set_image()
68
69         GAME_WIDGETS['chessboard'].refresh_board()
70
71         GAME_WIDGETS['blue_piece_display'].reset_piece_list()
72         GAME_WIDGETS['red_piece_display'].reset_piece_list()
73
74     def set_status_text(self, status):
75         """
76             Sets text on status text widget.
77
78             Args:
79                 status (StatusText): The game stage for which text should be displayed
80             for.
81         """
82         match status:
83             case StatusText.PLAYER_MOVE:
84                 GAME_WIDGETS['status_text'].set_text(f"{self._model.states['ACTIVE_COLOUR'].name}'s turn to move")
85             case StatusText.CPU_MOVE:
86                 GAME_WIDGETS['status_text'].set_text("CPU calculating a crazy move")
87             case StatusText.WIN:
88                 if self._model.states['WINNER'] == Miscellaneous.DRAW:
89                     GAME_WIDGETS['status_text'].set_text("Game is a draw! Boring")
90                 else:
91                     GAME_WIDGETS['status_text'].set_text(f"{self._model.states['WINNER'].name} won!")
92             case StatusText.DRAW:
93                 GAME_WIDGETS['status_text'].set_text("Game is a draw! Boring...")
94
95     def handle_resize(self):
96         """
97             Handles resizing of the window.
98         """
99         self._overlay_draw.handle_resize(self.board_position, self.board_size)
100        self._capture_draw.handle_resize(self.board_position, self.board_size)
101        self._piece_group.handle_resize(self.board_position, self.board_size)
102        self._laser_draw.handle_resize(self.board_position, self.board_size)
103        self._laser_draw.handle_resize(self.board_position, self.board_size)
104        self._widget_group.handle_resize(window.size)
105
106        if self._laser_draw.firing:
107            self.update_laser_mask()
108
109    def handle_update_pieces(self, event=None):
110        """
111

```

```

110     Callback function to update pieces after move.
111
112     Args:
113         event (GameEventType, optional): If updating pieces after player move,
114             event contains move information. Defaults to None.
115         toggle_timers (bool, optional): Toggle timers on and off for new
116             active colour. Defaults to True.
117             """
118
119         piece_list = self._model.get_piece_list()
120         self._piece_group.initialise_pieces(piece_list, self.board_position, self.
121             board_size)
122
123         if event:
124             GAME_WIDGETS['move_list'].append_to_move_list(event.move_notation)
125             GAME_WIDGETS['scroll_area'].set_image()
126             audio.play_sfx(SFX['piece_move'])
127
128         if self._model.states['ACTIVE_COLOUR'] == Colour.BLUE:
129             self.set_status_text(StatusText.PLAYER_MOVE)
130         elif self._model.states['CPU_ENABLED'] is False:
131             self.set_status_text(StatusText.PLAYER_MOVE)
132         else:
133             self.set_status_text(StatusText.CPU_MOVE)
134
135         if self._model.states['TIME_ENABLED']:
136             self.toggle_timer(self._model.states['ACTIVE_COLOUR'], True)
137             self.toggle_timer(self._model.states['ACTIVE_COLOUR'].
138                 get_flipped_colour(), False)
139
140         if self._model.states['WINNER'] is not None:
141             self.handle_game_end()
142
143     def handle_game_end(self, play_sfx=True):
144         self.toggle_timer(self._model.states['ACTIVE_COLOUR'], False)
145         self.toggle_timer(self._model.states['ACTIVE_COLOUR'].get_flipped_colour()
146             , False)
147
148         if self._model.states['WINNER'] == Miscellaneous.DRAW:
149             self.set_status_text(StatusText.DRAW)
150         else:
151             self.set_status_text(StatusText.WIN)
152
153         if play_sfx:
154             audio.play_sfx(SFX['sphinx_destroy_1'])
155             audio.play_sfx(SFX['sphinx_destroy_2'])
156             audio.play_sfx(SFX['sphinx_destroy_3'])
157
158     def handle_set_laser(self, event):
159             """
160             Callback function to draw laser after move.
161
162             Args:
163                 event (GameEventType): Contains laser trajectory information.
164             """
165
166         laser_result = event.laser_result
167
168         # If laser has hit a piece
169         if laser_result.hit_square_bitboard:
170             coords_to_remove = bitboard_to_coords(laser_result.hit_square_bitboard
171             )
172
173             self._piece_group.remove_piece(coords_to_remove)

```

```

166         if laser_result.piece_colour == Colour.BLUE:
167             GAME_WIDGETS['red_piece_display'].add_piece(laser_result.piece_hit)
168     )
169     elif laser_result.piece_colour == Colour.RED:
170         GAME_WIDGETS['blue_piece_display'].add_piece(laser_result.
171             piece_hit)
172
173     # Draw piece capture GFX
174     self._capture_draw.add_capture(
175         laser_result.piece_hit,
176         laser_result.piece_colour,
177         laser_result.piece_rotation,
178         coords_to_remove,
179         laser_result.laser_path[0][0],
180         self._model.states['ACTIVE_COLOUR']
181     )
182
183     self._laser_draw.add_laser(laser_result, self._model.states['ACTIVE_COLOUR'])
184   ])
185
186     self.update_laser_mask()
187
188 def handle_pause(self, event=None):
189 """
190     Callback function for pausing timer.
191
192 Args:
193     event (None): Event argument not used.
194
195     is_active = not(self._model.states['PAUSED'])
196     self.toggle_timer(self._model.states['ACTIVE_COLOUR'], is_active)
197
198 def initialise_timers(self):
199 """
200
201     Initialises both timers with the correct amount of time and starts the
202     timer for the active colour.
203
204     if self._model.states['TIME_ENABLED']:
205         GAME_WIDGETS['blue_timer'].set_time(self._model.states['TIME'] * 60 *
206         1000)
207         GAME_WIDGETS['red_timer'].set_time(self._model.states['TIME'] * 60 *
208         1000)
209     else:
210         GAME_WIDGETS['blue_timer'].kill()
211         GAME_WIDGETS['red_timer'].kill()
212
213     self.toggle_timer(self._model.states['ACTIVE_COLOUR'], True)
214
215 def toggle_timer(self, colour, is_active):
216 """
217
218     Stops or resumes timer.
219
220     Args:
221         colour (Colour): Timer to toggle.
222         is_active (bool): Whether to pause or resume timer.
223
224     if colour == Colour.BLUE:
225         GAME_WIDGETS['blue_timer'].set_active(is_active)
226     elif colour == Colour.RED:
227         GAME_WIDGETS['red_timer'].set_active(is_active)
228
229 def update_laser_mask(self):
230 """

```

```

222     Uses pygame.mask to create a mask for the pieces.
223     Used for occluding the ray shader.
224     """
225     temp_surface = pygame.Surface(window.size, pygame.SRCALPHA)
226     self._piece_group.draw(temp_surface)
227     mask = pygame.mask.from_surface(temp_surface, threshold=127)
228     mask_surface = mask.to_surface(unsetcolor=(0, 0, 0, 255), setcolor=(255,
229     0, 0, 255))
230     window.set_apply_arguments(ShaderType.RAYS, occlusion=mask_surface)
231
232     def draw(self):
233         """
234         Draws GUI and pieces onto the screen.
235         """
236         self._widget_group.update()
237         self._capture_draw.update()
238
239         self._widget_group.draw()
240         self._overlay_draw.draw(window.screen)
241
242         if self._hide_pieces is False:
243             self._piece_group.draw(window.screen)
244
245         self._laser_draw.draw(window.screen)
246         self._drag_and_drop.draw(window.screen)
247         self._capture_draw.draw(window.screen)
248
249     def process_model_event(self, event):
250         """
251             Registered listener function for handling GameModel events.
252             Each event is mapped to a callback function, and the appropriate one is run
253
254         Args:
255             event (GameEventType): Game event to process.
256
257         Raises:
258             KeyError: If an unrecognised event type is passed as the argument.
259         """
260         try:
261             self._event_to_func_map.get(event.type)(event)
262         except:
263             raise KeyError('Event type not recognized in Game View (GameView.
process_model_event):', event.type)
264
265     def set_overlay_coords(self, available_coords_list, selected_coord):
266         """
267             Set board coordinates for potential moves overlay.
268
269         Args:
270             available_coords_list (list[tuple[int, int]], ...): Array of
271             coordinates
272             selected_coord (list[int, int]): Coordinates of selected piece.
273
274             self._selected_coords = selected_coord
275             self._overlay_draw.set_selected_coords(selected_coord)
276             self._overlay_draw.set_available_coords(available_coords_list)
277
278     def get_selected_coords(self):
279         return self._selected_coords

```

```
280     def set_dragged_piece(self, piece, colour, rotation):
281         """
282             Passes information of the dragged piece to the dragging drawing class.
283
284         Args:
285             piece (Piece): Piece type of dragged piece.
286             colour (Colour): Colour of dragged piece.
287             rotation (Rotation): Rotation of dragged piece.
288         """
289         self._drag_and_drop.set_dragged_piece(piece, colour, rotation)
290
291     def remove_dragged_piece(self):
292         """
293             Stops drawing dragged piece when user lets go of piece.
294         """
295         self._drag_and_drop.remove_dragged_piece()
296
297     def convert_mouse_pos(self, event):
298         """
299             Passes information of what mouse cursor is interacting with to a
300             GameController object.
301
302         Args:
303             event (pygame.Event): Mouse event to process.
304
305         Returns:
306             CustomEvent | None: Contains information what mouse is doing.
307         """
308         clicked_coords = screen_pos_to_coords(event.pos, self.board_position, self
309             .board_size)
310
311         if event.type == pygame.MOUSEBUTTONDOWN:
312             if clicked_coords:
313                 return CustomEvent.create_event(GameEventType.BOARD_CLICK, coords=
314                     clicked_coords)
315
316         else:
317             return None
318
319         elif event.type == pygame.MOUSEBUTTONUP:
320             if self._drag_and_drop.dragged_sprite:
321                 piece, colour, rotation = self._drag_and_drop.get_dragged_info()
322                 piece_dragged = self._drag_and_drop.remove_dragged_piece()
323                 return CustomEvent.create_event(GameEventType.PIECE_DROP, coords=
324                     clicked_coords, piece=piece, colour=colour, rotation=rotation, remove_overlay=
325                     piece_dragged)
326
327     def add_help_screen(self):
328         """
329             Draw help overlay when player clicks on the help button.
330         """
331         self._widget_group.add(GAME_WIDGETS['help'])
332         self._widget_group.handle_resize(window.size)
333
334     def add_tutorial_screen(self):
335         """
336             Draw tutorial overlay when player clicks on the tutorial button.
337         """
338         self._widget_group.add(GAME_WIDGETS['tutorial'])
339         self._widget_group.handle_resize(window.size)
340         self._hide_pieces = True
```

```

337     def remove_help_screen(self):
338         GAME_WIDGETS['help'].kill()
339
340     def remove_tutorial_screen(self):
341         GAME_WIDGETS['tutorial'].kill()
342         self._hide_pieces = False
343
344     def process_widget_event(self, event):
345         """
346             Passes Pygame event to WidgetGroup to allow individual widgets to process
347             events.
348
349             Args:
350                 event (pygame.Event): Event to process.
351
352             Returns:
353                 CustomEvent | None: A widget event.
354
355         return self._widget_group.process_event(event)

```

### 3.5.3 Controller

As described in Section 2.4, the controller class is responsible for receiving external input through Pygame events, and processing them via the model and view classes.

`game_controller.py`

```

1 import pygame
2 from data.helpers import bitboard_helpers as bb_helpers
3 from data.utils.enums import MoveType, Miscellaneous
4 from data.states.game.components.move import Move
5 from data.utils.event_types import GameEventType
6 from data.managers.logs import initialise_logger
7
8 logger = initialise_logger(__name__)
9
10 class GameController:
11     def __init__(self, model, view, win_view, pause_view, to_menu, to_review,
12                  to_new_game):
13         self._model = model
14         self._view = view
15         self._win_view = win_view
16         self._pause_view = pause_view
17
18         self._to_menu = to_menu
19         self._to_review = to_review
20         self._to_new_game = to_new_game
21
22         self._view.initialise_timers()
23         self._win_view.set_win_type('CAPTURE')
24
25     def cleanup(self, next):
26         """
27             Handles game quit, either leaving to main menu or restarting a new game.
28
29             Args:
30                 next (str): New state to switch to.
31
32         self._model.kill_thread()
33
34         if next == 'menu':

```

```

34         self._to_menu()
35     elif next == 'game':
36         self._to_new_game()
37     elif next == 'review':
38         self._to_review()

39
40     def make_move(self, move):
41         """
42             Handles player move.
43
44             Args:
45                 move (Move): Move to make.
46
47             self._model.make_move(move)
48             self._view.set_overlay_coords([], None)
49
50             if self._model.states['CPU_ENABLED']:
51                 self._model.make_cpu_move()
52
53     def handle_pause_event(self, event):
54         """
55             Processes events when game is paused.
56
57             Args:
58                 event (GameEventType): Event to process.
59
60             Raises:
61                 Exception: If event type is unrecognised.
62
63             game_event = self._pause_view.convert_mouse_pos(event)
64
65             if game_event is None:
66                 return
67
68             match game_event.type:
69                 case GameEventType.PAUSE_CLICK:
70                     self._model.toggle_paused()
71
72                 case GameEventType.MENU_CLICK:
73                     self.cleanup('menu')
74
75                 case _:
76                     raise Exception('Unhandled event type (GameController.handle_event)')
77
78     def handle_winner_event(self, event):
79         """
80             Processes events when game is over.
81
82             Args:
83                 event (GameEventType): Event to process.
84
85             Raises:
86                 Exception: If event type is unrecognised.
87
88             game_event = self._win_view.convert_mouse_pos(event)
89
90             if game_event is None:
91                 return
92
93             match game_event.type:
94                 case GameEventType.MENU_CLICK:

```

```
95             self.cleanup('menu')
96         return
97
98     case GameEventType.GAME_CLICK:
99         self.cleanup('game')
100    return
101
102    case GameEventType.REVIEW_CLICK:
103        self.cleanup('review')
104
105    case _:
106        raise Exception('Unhandled event type (GameController.handle_event
107 )')
108
109    def handle_game_widget_event(self, event):
110        """
111            Processes events for game GUI widgets.
112
113        Args:
114            event (GameEventType): Event to process.
115
116        Raises:
117            Exception: If event type is unrecognised.
118
119        Returns:
120            CustomEvent | None: A widget event.
121        """
122
123        widget_event = self._view.process_widget_event(event)
124
125        if widget_event is None:
126            return None
127
128        match widget_event.type:
129            case GameEventType.ROTATE_PIECE:
130                src_coords = self._view.get_selected_coords()
131
132                if src_coords is None:
133                    logger.info('None square selected')
134                    return
135
136                move = Move.instance_from_coords(MoveType.ROTATE, src_coords,
137                                                src_coords, rotation_direction=widget_event.rotation_direction)
138                self.make_move(move)
139
140            case GameEventType.RESIGN_CLICK:
141                self._model.set_winner(self._model.states['ACTIVE_COLOUR'].get_flipped_colour())
142                self._view.handle_game_end(play_sfx=False)
143                self._win_view.set_win_type('RESIGN')
144
145            case GameEventType.DRAW_CLICK:
146                self._model.set_winner(Miscellaneous.DRAW)
147                self._view.handle_game_end(play_sfx=False)
148                self._win_view.set_win_type('DRAW')
149
150            case GameEventType.TIMER_END:
151                if self._model.states['TIME_ENABLED']:
152                    self._model.set_winner(widget_event.active_colour.get_flipped_colour())
153                    self._win_view.set_win_type('TIME')
154                    self._view.handle_game_end(play_sfx=False)
```

```

153         case GameEventType.MENU_CLICK:
154             self._cleanup('menu')
155
156         case GameEventType.HELP_CLICK:
157             self._view.add_help_screen()
158
159         case GameEventType.TUTORIAL_CLICK:
160             self._view.add_tutorial_screen()
161
162         case _:
163             raise Exception('Unhandled event type (GameController.handle_event'
164             ')')
165
166     return widget_event.type
167
168     def check_cpu(self):
169         """
170             Checks if CPU calculations are finished every frame.
171         """
172         if self._model.states['CPU_ENABLED'] and self._model.states['AWAITING_CPU']
173             ] is False:
174             self._model.check_cpu()
175
176     def handle_game_event(self, event):
177         """
178             Processes Pygame events for main game.
179
180             Args:
181                 event (pygame.Event): If event type is unrecognised.
182
183             Raises:
184                 Exception: If event type is unrecognised.
185
186             # Pass event for widgets to process
187             widget_event = self.handle_game_widget_event(event)
188
189             if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
190 KEYDOWN]:
191                 if event.type != pygame.KEYDOWN:
192                     game_event = self._view.convert_mouse_pos(event)
193                 else:
194                     game_event = None
195
196                 if game_event is None:
197                     if widget_event is None:
198                         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
199                             # If user releases mouse click not on a widget
200                             self._view.remove_help_screen()
201                             self._view.remove_tutorial_screen()
202                         if event.type == pygame.MOUSEBUTTONUP:
203                             # If user releases mouse click on neither a widget or
204                             board
205                             self._view.set_overlay_coords(None, None)
206
207             return
208
209             match game_event.type:
210                 case GameEventType.BOARD_CLICK:
211                     if self._model.states['AWAITING_CPU']:
212                         return
213
214                     clicked_coords = game_event.coords

```

```

211             clicked_bitboard = bb_helpers.coords_to_bitboard(
212             clicked_coords)
213
214             selected_coords = self._view.get_selected_coords()
215
216             if selected_coords:
217                 if clicked_coords == selected_coords:
218                     # If clicking on an already selected square, start
219                     # dragging piece on that square
220                     self._view.set_dragged_piece(*self._model.
221                     get_piece_info(clicked_bitboard))
222                     return
223
224             selected_bitboard = bb_helpers.coords_to_bitboard(
225             selected_coords)
226             available_bitboard = self._model.get_available_moves(
227             selected_bitboard)
228
229             if bb_helpers.is_occupied(clicked_bitboard,
230             available_bitboard):
231                 # If the newly clicked square is not the same as the
232                 # old one, and is an empty surrounding square, make a move
233                 move = Move.instance_from_coords(MoveType.MOVE,
234             selected_coords, clicked_coords)
235                 self.make_move(move)
236             else:
237                 # If the newly clicked square is not the same as the
238                 # old one, but is an invalid square, unselect the currently selected square
239                 self._view.set_overlay_coords(None, None)
240
241                 # Select hovered square if it is same as active colour
242                 elif self._model.is_selectable(clicked_bitboard):
243                     available_bitboard = self._model.get_available_moves(
244                     clicked_bitboard)
245                     self._view.set_overlay_coords(bb_helpers.
246                     bitboard_to_coords_list(available_bitboard), clicked_coords)
247                     self._view.set_dragged_piece(*self._model.get_piece_info(
248                     clicked_bitboard))
249
250             case GameEventType.PIECE_DROP:
251                 hovered_coords = game_event.coords
252
253                 # if piece is dropped onto the board
254                 if hovered_coords:
255                     hovered_bitboard = bb_helpers.coords_to_bitboard(
256                     hovered_coords)
257
258                     selected_coords = self._view.get_selected_coords()
259                     selected_bitboard = bb_helpers.coords_to_bitboard(
260                     selected_coords)
261                     available_bitboard = self._model.get_available_moves(
262                     selected_bitboard)
263
264                     if bb_helpers.is_occupied(hovered_bitboard,
265                     available_bitboard):
266                         # Make a move if mouse is hovered over an empty
267                         # surrounding square
268                         move = Move.instance_from_coords(MoveType.MOVE,
269                     selected_coords, hovered_coords)
270                         self.make_move(move)
271
272                     if game_event.remove_overlay:
273                         self._view.set_overlay_coords(None, None)
274

```

```

255             self._view.remove_dragged_piece()
256
257         case _:
258             raise Exception('Unhandled event type (GameController.
handle_event)', game_event.type)
259
260     def handle_event(self, event):
261         """
262             Pass a Pygame event to the correct handling function according to the
game state.
263
264         Args:
265             event (pygame.Event): Event to process.
266
267         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
MOUSEMOTION, pygame.KEYDOWN]:
268             if self._model.states['PAUSED']:
269                 self.handle_pause_event(event)
270             elif self._model.states['WINNER'] is not None:
271                 self.handle_winner_event(event)
272             else:
273                 self.handle_game_event(event)
274
275         if event.type == pygame.KEYDOWN:
276             if event.key == pygame.K_ESCAPE:
277                 self._model.toggle_paused()
278             elif event.key == pygame.K_l:
279                 logger.info('\nSTOPPING CPU')
280                 self._model._cpu_thread.stop_cpu() #temp

```

### 3.5.4 Board

The `Board` class implements the Laser Chess board, and is responsible for handling moves, captures, and win conditions.

`board.py`

```

1  from collections import defaultdict
2  from data.utils.constants import A_FILE_MASK, J_FILE_MASK, ONE_RANK_MASK,
EIGHT_RANK_MASK, EMPTY_BB
3  from data.utils.enums import Colour, Piece, Rank, File, MoveType,
RotationDirection, Miscellaneous
4  from data.states.game.components.bitboard_collection import BitboardCollection
5  from data.helpers import bitboard_helpers as bb_helpers
6  from data.states.game.components.laser import Laser
7  from data.states.game.components.move import Move
8
9
10 class Board:
11     def __init__(self, fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/
pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa b"):
12         self.bitboards = BitboardCollection(fen_string)
13         self.hash_list = [self.bitboards.get_hash()]
14
15     def __str__(self):
16         """
17             Returns a string representation of the board.
18
19             Returns:
20                 str: Board formatted as string.
21         """

```

```

22     characters = '8 '
23     pieces = defaultdict(int)
24
25     for rank_idx, rank in enumerate(reversed(Rank)):
26         for file_idx, file in enumerate(File):
27             mask = 1 << (rank * 10 + file)
28             blue_piece = self.bitboards.get_piece_on(mask, Colour.BLUE)
29             red_piece = self.bitboards.get_piece_on(mask, Colour.RED)
30
31             if blue_piece:
32                 pieces[blue_piece.value.upper()] += 1
33                 characters += f'{blue_piece.upper()} '
34             elif red_piece:
35                 pieces[red_piece.value] += 1
36                 characters += f'{red_piece} '
37             else:
38                 characters += '. '
39
40             characters += f'\n\n{7 - rank_idx} '
41     characters += 'A B C D E F G H I J\n\n'
42     characters += str(dict(pieces))
43     characters += f'\nCURRENT PLAYER TO MOVE: {self.bitboards.active_colour.
name}\n'
44     return characters
45
46     def get_piece_list(self):
47         """
48             Converts the board bitboards to a list of pieces.
49
50             Returns:
51                 list: List of Pieces.
52         """
53         return self.bitboards.convert_to_piece_list()
54
55     def get_active_colour(self):
56         """
57             Gets the active colour.
58
59             Returns:
60                 Colour: The active colour.
61         """
62         return self.bitboards.active_colour
63
64     def to_hash(self):
65         """
66             Gets the hash of the current board state.
67
68             Returns:
69                 int: A Zobrist hash.
70         """
71         return self.bitboards.get_hash()
72
73     def check_win(self):
74         """
75             Checks for a Pharaoh capture or threefold-repetition.
76
77             Returns:
78                 Colour | Miscellaneous: The winning colour, or Miscellaneous.DRAW.
79         """
80         for colour in Colour:
81             if self.bitboards.get_piece_bitboard(Piece.PHAROAH, colour) ==
EMPTY_BB:

```

```

82             return colour.get_flipped_colour()
83
84         if self.hash_list.count(self.hash_list[-1]) >= 3:
85             return Miscellaneous.DRAW
86
87         return None
88
89     def apply_move(self, move, fire_laser=True, add_hash=False):
90         """
91             Applies a move to the board.
92
93             Args:
94                 move (Move): The move to apply.
95                 fire_laser (bool): Whether to fire the laser after the move.
96                 add_hash (bool): Whether to add the board state hash to the hash list.
97
98             Returns:
99                 Laser: The laser trajectory result.
100
101            piece_symbol = self.bitboards.get_piece_on(move.src, self.bitboards.
102                active_colour)
103
104            if piece_symbol is None:
105                raise ValueError(f'Invalid move - no piece found on source square. {move}')
106            elif piece_symbol == Piece.SPHINX:
107                raise ValueError(f'Invalid move - sphinx piece is immovable. {move}')
108
109            if move.move_type == MoveType.MOVE:
110                possible_moves = self.get_valid_squares(move.src)
111                if bb_helpers.is_occupied(move.dest, possible_moves) is False:
112                    raise ValueError('Invalid move - destination square is occupied')
113
114                piece_rotation = self.bitboards.get_rotation_on(move.src)
115
116                self.bitboards.update_move(move.src, move.dest)
117                self.bitboards.update_rotation(move.src, move.dest, piece_rotation)
118
119            elif move.move_type == MoveType.ROTATE:
120                piece_symbol = self.bitboards.get_piece_on(move.src, self.bitboards.
121                    active_colour)
122                piece_rotation = self.bitboards.get_rotation_on(move.src)
123
124                if move.rotation_direction == RotationDirection.CLOCKWISE:
125                    new_rotation = piece_rotation.get_clockwise()
126                elif move.rotation_direction == RotationDirection.ANTICLOCKWISE:
127                    new_rotation = piece_rotation.get_anticlockwise()
128
129                self.bitboards.update_rotation(move.src, move.dest, new_rotation)
130
131            laser = None
132            if fire_laser:
133                laser = self.fire_laser(add_hash)
134
135            if add_hash:
136                self.hash_list.append(self.bitboards.get_hash())
137
138            self.bitboards.flip_colour()
139
140        return laser
141
142    def undo_move(self, move, laser_result):

```

```

141     """
142     Undoes a move on the board.
143
144     Args:
145         move (Move): The move to undo.
146         laser_result (Laser): The laser trajectory result.
147     """
148     self.bitboards.flip_colour()
149
150     if laser_result.hit_square_bitboard:
151         # Get info of destroyed piece, and add it to the board again
152         src = laser_result.hit_square_bitboard
153         piece = laser_result.piece_hit
154         colour = laser_result.piece_colour
155         rotation = laser_result.piece_rotation
156
157         self.bitboards.set_square(src, piece, colour)
158         self.bitboards.clear_rotation(src)
159         self.bitboards.set_rotation(src, rotation)
160
161     # Create new Move object that is the inverse of the passed move
162     if move.move_type == MoveType.MOVE:
163         reversed_move = Move.instance_from_bitboards(MoveType.MOVE, move.dest,
164             move.src)
165     elif move.move_type == MoveType.ROTATE:
166         reversed_move = Move.instance_from_bitboards(MoveType.ROTATE, move.src,
167             move.src, move.rotation_direction.get_opposite())
168
169     self.apply_move(reversed_move, fire_laser=False)
170     self.bitboards.flip_colour()
171
172     def remove_piece(self, square_bitboard):
173         """
174             Removes a piece from a given square.
175
176             Args:
177                 square_bitboard (int): The bitboard representation of the square.
178             """
179             self.bitboards.clear_square(square_bitboard, Colour.BLUE)
180             self.bitboards.clear_square(square_bitboard, Colour.RED)
181             self.bitboards.clear_rotation(square_bitboard)
182
183     def get_valid_squares(self, src_bitboard, colour=None):
184         """
185             Gets valid squares for a piece to move to.
186
187             Args:
188                 src_bitboard (int): The bitboard representation of the source square.
189                 colour (Colour, optional): The active colour of the piece.
190
191             Returns:
192                 int: The bitboard representation of valid squares.
193
194             target_top_left = (src_bitboard & A_FILE_MASK & EIGHT_RANK_MASK) << 9
195             target_top_middle = (src_bitboard & EIGHT_RANK_MASK) << 10
196             target_top_right = (src_bitboard & J_FILE_MASK & EIGHT_RANK_MASK) << 11
197             target_middle_right = (src_bitboard & J_FILE_MASK) << 1
198
199             target_bottom_right = (src_bitboard & J_FILE_MASK & ONE_RANK_MASK) >> 9
200             target_bottom_middle = (src_bitboard & ONE_RANK_MASK) >> 10
201             target_bottom_left = (src_bitboard & A_FILE_MASK & ONE_RANK_MASK)>> 11
202             target_middle_left = (src_bitboard & A_FILE_MASK) >> 1

```

```
201     possible_moves = target_top_left | target_top_middle | target_top_right |  
202     target_middle_right | target_bottom_right | target_bottom_middle |  
203     target_bottom_left | target_middle_left  
204     if colour is not None:  
205         valid_possible_moves = possible_moves & ~self.bitboards.  
combined_colour_bitboards[colour]  
206     else:  
207         valid_possible_moves = possible_moves & ~self.bitboards.  
combined_all_bitboard  
208     return valid_possible_moves  
209  
210 def get_mobility(self, colour):  
211     """  
212     Gets all valid squares for a given colour.  
213  
214     Args:  
215         colour (Colour): The colour of the pieces.  
216  
217     Returns:  
218         int: The bitboard representation of all valid squares.  
219     """  
220     active_pieces = self.get_all_active_pieces(colour)  
221     possible_moves = 0  
222  
223     for square in bb_helpers.occupied_squares(active_pieces):  
224         possible_moves += bb_helpers.pop_count(self.get_valid_squares(square))  
225  
226     return possible_moves  
227  
228 def get_all_active_pieces(self, colour=None):  
229     """  
230     Gets all active pieces for the current player.  
231  
232     Args:  
233         colour (Colour): Active colour of pieces to retrieve. Defaults to None  
234     """  
235  
236     Returns:  
237         int: The bitboard representation of all active pieces.  
238     """  
239     if colour is None:  
240         colour = self.bitboards.active_colour  
241  
242     active_pieces = self.bitboards.combined_colour_bitboards[colour]  
243     sphinx_bitboard = self.bitboards.get_piece_bitboard(Piece.SPHINX, colour)  
244     return active_pieces ^ sphinx_bitboard  
245  
246 def fire_laser(self, remove_hash):  
247     """  
248     Fires the laser and removes hit pieces.  
249  
250     Args:  
251         remove_hash (bool): Whether to clear the hash list if a piece is hit.  
252  
253     Returns:  
254         Laser: The result of firing the laser.  
255     """  
256     laser = Laser(self.bitboards)  
257
```

```

258         if laser.hit_square_bitboard:
259             self.remove_piece(laser.hit_square_bitboard)
260
261         if remove_hash:
262             self.hash_list = [] # Remove all hashes for threefold repetition,
263             as the position is impossible to be repeated after a piece is removed
264             return laser
265
266     def generate_square_moves(self, src):
267         """
268             Generates all valid moves for a piece on a given square.
269
270             Args:
271                 src (int): The bitboard representation of the source square.
272
273             Yields:
274                 Move: A valid move for the piece.
275
276         for dest in bb_helpers.occupied_squares(self.get_valid_squares(src)):
277             yield Move(MoveType.MOVE, src, dest)
278
279     def generate_all_moves(self, colour):
280         """
281             Generates all valid moves for a given colour.
282
283             Args:
284                 colour (Colour): The colour of the pieces.
285
286             Yields:
287                 Move: A valid move for the active colour.
288
289             sphinx_bitboard = self.bitboards.get_piece_bitboard(Piece.SPHINX, colour)
290             # Remove source squares for Sphinx pieces, as they cannot be moved
291             sphinx_masked_bitboard = self.bitboards.combined_colour_bitboards[colour]
292             ~ sphinx_bitboard
293
294             for square in bb_helpers.occupied_squares(sphinx_masked_bitboard):
295                 # Generate movement moves
296                 yield from self.generate_square_moves(square)
297
298                 # Generate rotational moves
299                 for rotation_direction in RotationDirection:
300                     yield Move(MoveType.ROTATE, square, rotation_direction=
301                         rotation_direction)

```

### 3.5.5 Bitboards

As described in Section 2.2.3, the `BitboardCollection` class uses helper functions found in `bitboard_helpers.py` such as `pop_count`, to initialise and manage bitboard transformations.

`bitboard_collection.py`

```

1  from data.utils.enums import Rank, File, Piece, Colour, Rotation, RotationIndex
2  from data.states.game.components.fen_parser import parse_fen_string
3  from data.states.game.cpu.zobrist_hasher import ZobristHasher
4  from data.helpers import bitboard_helpers as bb_helpers
5  from data.managers.logs import initialise_logger
6  from data.utils.constants import EMPTY_BB
7
8  logger = initialise_logger(__name__)
9

```

```

10  class BitboardCollection:
11      def __init__(self, fen_string):
12          self.piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char: EMPTY_BB for char in Piece}]
13          self.combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
14          self.combined_all_bitboard = EMPTY_BB
15          self.rotation_bitboards = [EMPTY_BB, EMPTY_BB]
16          self.active_colour = Colour.BLUE
17          self._hasher = ZobristHasher()
18
19      try:
20          if fen_string:
21              self.piece_bitboards, self.combined_colour_bitboards, self.
22              combined_all_bitboard, self.rotation_bitboards, self.active_colour =
23              parse_fen_string(fen_string)
24              self.initialise_hash()
25          except ValueError as error:
26              logger.error('Please input a valid FEN string:', error)
27              raise error
28
29      def __str__(self):
30          """
31              Returns a string representation of the bitboards.
32
33          Returns:
34              str: Bitboards formatted with piece type and colour shown.
35          """
36          characters = ''
37          for rank in reversed(Rank):
38              for file in File:
39                  bitboard = 1 << (rank * 10 + file)
40
41                  colour = self.get_colour_on(bitboard)
42                  piece = self.get_piece_on(bitboard, Colour.BLUE) or self.
43                  get_piece_on(bitboard, Colour.RED)
44
45                  if piece is not None:
46                      characters += f'{piece.upper() if colour == Colour.BLUE
47 else piece} '
48                  else:
49                      characters += '. '
50
51          characters += '\n\n'
52
53      return characters
54
55      def get_rotation_string(self):
56          """
57              Returns a string representation of the board rotations.
58
59          Returns:
60              str: Board formatted with only rotations shown.
61          """
62          characters = ''
63          for rank in reversed(Rank):
64
65              for file in File:
66                  mask = 1 << (rank * 10 + file)
67                  rotation = self.get_rotation_on(mask)
68                  has_piece = bb_helpers.is_occupied(self.combined_all_bitboard,
69                  mask)

```

```

66             if has_piece:
67                 characters += f'{rotation.upper()} '
68             else:
69                 characters += '. '
70
71         characters += '\n\n'
72
73     return characters
74
75 def initialise_hash(self):
76     """
77     Initialises the Zobrist hash for the current board state.
78     """
79     for piece in Piece:
80         for colour in Colour:
81             piece_bitboard = self.get_piece_bitboard(piece, colour)
82
83             for occupied_bitboard in bb_helpers.occupied_squares(
84                 piece_bitboard):
85                 self._hasher.apply_piece_hash(occupied_bitboard, piece, colour)
86
87             for bitboard in bb_helpers.loop_all_squares():
88                 rotation = self.get_rotation_on(bitboard)
89                 self._hasher.apply_rotation_hash(bitboard, rotation)
90
91             if self.active_colour == Colour.RED:
92                 self._hasher.apply_red_move_hash()
93
94     def flip_colour(self):
95         """
96         Flips the active colour and updates the Zobrist hash.
97         """
98         self.active_colour = self.active_colour.get_flipped_colour()
99
100        if self.active_colour == Colour.RED:
101            self._hasher.apply_red_move_hash()
102
103    def update_move(self, src, dest):
104        """
105        Updates the bitboards for a move.
106
107        Args:
108            src (int): The bitboard representation of the source square.
109            dest (int): The bitboard representation of the destination square.
110        """
111
112        piece = self.get_piece_on(src, self.active_colour)
113
114        self.clear_square(src, Colour.BLUE)
115        self.clear_square(dest, Colour.BLUE)
116        self.clear_square(src, Colour.RED)
117        self.clear_square(dest, Colour.RED)
118
119        self.set_square(dest, piece, self.active_colour)
120
121    def update_rotation(self, src, dest, new_rotation):
122        """
123        Updates the rotation bitboards for a move.
124
125        Args:
126            src (int): The bitboard representation of the source square.
127            dest (int): The bitboard representation of the destination square.

```

```

126         new_rotation (Rotation): The new rotation.
127     """
128     self.clear_rotation(src)
129     self.set_rotation(dest, new_rotation)
130
131     def clear_rotation(self, bitboard):
132     """
133         Clears the rotation for a given square.
134
135         Args:
136             bitboard (int): The bitboard representation of the square.
137         """
138         old_rotation = self.get_rotation_on(bitboard)
139         rotation_1, rotation_2 = self.rotation_bitboards
140         self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.clear_square(
141             rotation_1, bitboard)
142         self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.clear_square(
143             rotation_2, bitboard)
144
145         self._hasher.apply_rotation_hash(bitboard, old_rotation)
146
147     def clear_square(self, bitboard, colour):
148     """
149         Clears a square piece and rotation for a given colour.
150
151         Args:
152             bitboard (int): The bitboard representation of the square.
153             colour (Colour): The colour to clear.
154         """
155         piece = self.get_piece_on(bitboard, colour)
156
157         if piece is None:
158             return
159
160         piece_bitboard = self.get_piece_bitboard(piece, colour)
161         colour_bitboard = self.combined_colour_bitboards[colour]
162         all_bitboard = self.combined_all_bitboard
163
164         self.piece_bitboards[colour][piece] = bb_helpers.clear_square(
165             piece_bitboard, bitboard)
166         self.combined_colour_bitboards[colour] = bb_helpers.clear_square(
167             colour_bitboard, bitboard)
168         self.combined_all_bitboard = bb_helpers.clear_square(all_bitboard,
169             bitboard)
170
171         self._hasher.apply_piece_hash(bitboard, piece, colour)
172
173     def set_rotation(self, bitboard, rotation):
174     """
175         Sets the rotation for a given square.
176
177         Args:
178             bitboard (int): The bitboard representation of the square.
179             rotation (Rotation): The rotation to set.
180         """
181
182         rotation_1, rotation_2 = self.rotation_bitboards
183         self._hasher.apply_rotation_hash(bitboard, rotation)
184
185         match rotation:
186             case Rotation.UP:
187                 return
188             case Rotation.RIGHT:

```

```

183         self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.
184         set_square(rotation_1, bitboard)
185         return
186     case Rotation.DOWN:
187         self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.
188         set_square(rotation_2, bitboard)
189         return
190     case Rotation.LEFT:
191         self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.
192         set_square(rotation_1, bitboard)
193         self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.
194         set_square(rotation_2, bitboard)
195         return
196     case _:
197         raise ValueError('Invalid rotation input (bitboard.py):', rotation
198 )
199
200 def set_square(self, bitboard, piece, colour):
201     """
202     Sets a piece on a given square.
203
204     Args:
205         bitboard (int): The bitboard representation of the square.
206         piece (Piece): The piece to set.
207         colour (Colour): The colour of the piece.
208
209     """
210     piece_bitboard = self.get_piece_bitboard(piece, colour)
211     colour_bitboard = self.combined_colour_bitboards[colour]
212     all_bitboard = self.combined_all_bitboard
213
214     self.piece_bitboards[colour][piece] = bb_helpers.set_square(piece_bitboard
215 , bitboard)
216     self.combined_colour_bitboards[colour] = bb_helpers.set_square(
217 colour_bitboard, bitboard)
218     self.combined_all_bitboard = bb_helpers.set_square(all_bitboard, bitboard)
219
220     self._hasher.apply_piece_hash(bitboard, piece, colour)
221
222 def get_piece_bitboard(self, piece, colour):
223     """
224     Gets the bitboard for a piece type for a given colour.
225
226     Args:
227         piece (Piece): The piece bitboard to get.
228         colour (Colour): The colour of the piece.
229
230     Returns:
231         int: The bitboard representation for all squares occupied by that
232             piece type.
233
234     """
235     return self.piece_bitboards[colour][piece]
236
237 def get_piece_on(self, target_bitboard, colour):
238     """
239     Gets the piece on a given square for a given colour.
240
241     Args:
242         target_bitboard (int): The bitboard representation of the square.
243         colour (Colour): The colour of the piece.
244
245     Returns:
246         Piece: The piece on the square, or None if square is empty.

```

```

237     """
238     if not (bb_helpers.is_occupied(self.combined_colour_bitboards[colour],
239         target_bitboard)):
240         return None
241
242     return next(
243         (piece for piece in Piece if
244             bb_helpers.is_occupied(self.get_piece_bitboard(piece, colour),
245             target_bitboard)),
246         None)
247
248     def get_rotation_on(self, target_bitboard):
249         """
250         Gets the rotation on a given square.
251
252         Args:
253             target_bitboard (int): The bitboard representation of the square.
254
255         Returns:
256             Rotation: The rotation on the square.
257         """
258         rotationBits = [bb_helpers.is_occupied(self.rotation_bitboards[
259             RotationIndex.SECONDBIT], target_bitboard), bb_helpers.is_occupied(self.
260             rotation_bitboards[RotationIndex.FIRSTBIT], target_bitboard)]
261
262         match rotationBits:
263             case [False, False]:
264                 return Rotation.UP
265             case [False, True]:
266                 return Rotation.RIGHT
267             case [True, False]:
268                 return Rotation.DOWN
269             case [True, True]:
270                 return Rotation.LEFT
271
272     def get_colour_on(self, target_bitboard):
273         """
274         Gets the colour of the piece on a given square.
275
276         Args:
277             target_bitboard (int): The bitboard representation of the square.
278
279         Returns:
280             Colour: The colour of the piece on the square.
281         """
282         for piece in Piece:
283             if self.get_piece_bitboard(piece, Colour.BLUE) & target_bitboard != EMPTY_BB:
284                 return Colour.BLUE
285             elif self.get_piece_bitboard(piece, Colour.RED) & target_bitboard != EMPTY_BB:
286                 return Colour.RED
287
288     def get_piece_count(self, piece, colour):
289         """
290         Gets the count of a given piece type and colour.
291
292         Args:
293             piece (Piece): The piece to count.
294             colour (Colour): The colour of the piece.
295
296         Returns:

```

```

293         int: The number of that piece of that colour on the board.
294     """
295     return bb_helpers.pop_count(self.get_piece_bitboard(piece, colour))
296
297     def get_hash(self):
298         """
299         Gets the Zobrist hash of the current board state.
300
301         Returns:
302             int: The Zobrist hash.
303         """
304         return self._hasher.hash
305
306     def convert_to_piece_list(self):
307         """
308         Converts all bitboards to a list of pieces.
309
310         Returns:
311             list: Board represented as a 2D list of Piece and Rotation objects.
312         """
313         piece_list = []
314
315         for i in range(80):
316             if x := self.get_piece_on(1 << i, Colour.BLUE):
317                 rotation = self.get_rotation_on(1 << i)
318                 piece_list.append((x.upper(), rotation))
319             elif y := self.get_piece_on(1 << i, Colour.RED):
320                 rotation = self.get_rotation_on(1 << i)
321                 piece_list.append((y, rotation))
322             else:
323                 piece_list.append(None)
324
325         return piece_list

```

## 3.6 CPU

This section includes my implementation for the CPU engine run on minimax, including its various improvements and accessory classes.

Every CPU engine class is a subclass of a `BaseCPU` abstract class, and therefore contains the same attribute and method names. This means **polymorphism** can be used again to easily test and vary the difficulty by switching out which CPU engine is used.

The method `find_move` is called by the CPU thread. `search` is then called recursively to traverse the minimax tree, and find an optimal move. The move is then returned to `find_move` and passed and run with the callback function. A `stats` dictionary is also created in the base class, used to collect information for each search.

### 3.6.1 Minimax

As described in Section 2.2.1, the minimax engine uses **DFS** to traverse the game tree and evaluate node accordingly, by **recursively** calling the `search` function.

`minimax.py`

```

1 from random import choice
2 from data.states.game.cpu.base import BaseCPU
3 from data.utils.enums import Score, Colour
4

```

```

5  class MinimaxCPU(BaseCPU):
6      def __init__(self, max_depth, callback, verbose=False):
7          super().__init__(callback, verbose)
8          self._max_depth = max_depth
9
10     def find_move(self, board, stop_event):
11         """
12             Finds the best move for the current board state.
13
14         Args:
15             board (Board): The current board state.
16             stop_event (threading.Event): Event used to kill search from an
17                 external class.
18
19         Returns:
20             tuple[int, Move]: The best score and the best move found.
21
22         """
23
24     def search(self, board, depth, stop_event):
25         """
26             Recursively DFS through minimax tree with evaluation score.
27
28         Args:
29             board (Board): The current board state.
30             depth (int): The current search depth.
31             stop_event (threading.Event): Event used to kill search from an
32                 external class.
33
34         Returns:
35             tuple[int, Move]: The best score and the best move found.
36
37         if (base_case := super().search(board, depth, stop_event)):
38             return base_case
39
40         best_move = None
41
42         # Blue is the maximising player
43         if board.get_active_colour() == Colour.BLUE:
44             max_score = -Score.INFINITE
45
46             for move in board.generate_all_moves(Colour.BLUE):
47                 laser_result = board.apply_move(move)
48
49
50             new_score = self.search(board, depth - 1, stop_event)[0]
51
52             # if depth < self._max_depth:
53             #     print('DEPTH', depth, new_score, move)
54
55             if new_score > max_score:
56                 max_score = new_score
57                 best_move = move
58
59             if new_score == (Score.CHECKMATE + self._max_depth):
60                 board.undo_move(move, laser_result)
61             return max_score, best_move
62
63         elif new_score == max_score:
64             # If evaluated scores are equal, pick a random move

```

```

65             best_move = choice([best_move, move])
66
67         board.undo_move(move, laser_result)
68
69     return max_score, best_move
70
71 else:
72     min_score = Score.INFINITE
73
74     for move in board.generate_all_moves(Colour.RED):
75         laser_result = board.apply_move(move)
76         # print('DEPTH', depth, move)
77         new_score = self.search(board, depth - 1, stop_event)[0]
78
79         if new_score < min_score:
80             # print('setting new', new_score, move)
81             min_score = new_score
82             best_move = move
83
84         if new_score == (-Score.CHECKMATE - self._max_depth):
85             board.undo_move(move, laser_result)
86             return min_score, best_move
87
88     elif new_score == min_score:
89         best_move = choice([best_move, move])
90
91     board.undo_move(move, laser_result)
92
93 return min_score, best_move

```

### 3.6.2 Alpha-beta Pruning

As described in Section 2.2.2, the `ABMinimaxCPU` class introduces pruning to reduce the number of nodes evaluated during a minimax search.

`alpha_beta.py`

```

1 from data.states.game.cpu.move_orderer import MoveOrderer
2 from data.states.game.cpu.base import BaseCPU
3 from data.utils.enums import Score, Colour
4
5 class ABMinimaxCPU(BaseCPU):
6     def __init__(self, max_depth, callback, verbose=True):
7         super().__init__(callback, verbose)
8         self._max_depth = max_depth
9         self._orderer = MoveOrderer()
10
11     def initialise_stats(self):
12         """
13             Initialises the number of prunes to the statistics dictionary to be logged
14
15         """
16         super().initialise_stats()
17         self._stats['beta_prunes'] = 0
18         self._stats['alpha_prunes'] = 0
19
20     def find_move(self, board, stop_event):
21         """
22             Finds the best move for the current board state.
23
24         Args:

```

```

24         board (Board): The current board state.
25         stop_event (threading.Event): Event used to kill search from an
26         external class.
27         """
28         self.initialise_stats()
29         best_score, best_move = self.search(board, self._max_depth, -Score.
30         INFINITE, Score.INFINITE, stop_event)
31
32         if self._verbose:
33             self.print_stats(best_score, best_move)
34
35     def search(self, board, depth, alpha, beta, stop_event, hint=None,
36     laser_coords=None):
37         """
38         Recursively DFS through minimax tree while pruning branches using the
39         alpha and beta bounds.
40
41         Args:
42             board (Board): The current board state.
43             depth (int): The current search depth.
44             alpha (int): The upper bound value.
45             beta (int): The lower bound value.
46             stop_event (threading.Event): Event used to kill search from an
47             external class.
48
49             Returns:
50                 tuple[int, Move]: The best score and the best move found.
51             """
52             if (base_case := super().search(board, depth, stop_event)):
53                 return base_case
54
55             best_move = None
56
57             # Blue is the maximising player
58             if board.get_active_colour() == Colour.BLUE:
59                 max_score = -Score.INFINITE
60
61                 for move in self._orderer.get_moves(board, hint=hint, laser_coords=
62                 laser_coords):
63                     laser_result = board.apply_move(move)
64                     new_score = self.search(board, depth - 1, alpha, beta, stop_event,
65                     laser_coords=laser_result.pieces_on_trajectory)[0]
66
67                     if new_score > max_score:
68                         max_score = new_score
69                         best_move = move
70
71                     board.undo_move(move, laser_result)
72
73                     alpha = max(alpha, max_score)
74
75                     if beta <= alpha:
76                         self._stats['alpha_prunes'] += 1
77                         break
78
79             return max_score, best_move
80
81         else:
82             min_score = Score.INFINITE

```

```

79         for move in self._orderer.get_moves(board, hint=hint, laser_coords=
80             laser_coords):
81             laser_result = board.apply_move(move)
82             new_score = self.search(board, depth - 1, alpha, beta, stop_event,
83             laser_coords=laser_result.pieces_on_trajectory)[0]
84
85             if new_score < min_score:
86                 min_score = new_score
87                 best_move = move
88
89             board.undo_move(move, laser_result)
90
91             beta = min(beta, min_score)
92             if beta <= alpha:
93                 self._stats['beta_prunes'] += 1
94                 break
95
96         return min_score, best_move

```

### 3.6.3 Transposition Table

For adding transposition table functionality to my other engine classes, as described in Section 2.2.2, I have decided to use a mixin design architecture. This allows me to **reuse code** by adding mixins to many different classes, and inject additional transposition table methods and functionality into other engines.

`transposition_table.py`

```

1  from data.states.game.cpu.transposition_table import TranspositionTable
2  from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU
3
4  class TranspositionTableMixin:
5      def __init__(self, *args, **kwargs):
6          super().__init__(*args, **kwargs)
7          self._table = TranspositionTable()
8
9      def find_move(self, *args, **kwargs):
10         self._table = TranspositionTable()
11         super().find_move(*args, **kwargs)
12
13     def search(self, board, depth, alpha, beta, stop_event, hint=None,
14               laser_coords=None):
15         """
16             Searches transposition table for a cached move before running a full
17             search if necessary.
18             Caches the searched result.
19
20             Args:
21                 board (Board): The current board state.
22                 depth (int): The current search depth.
23                 alpha (int): The upper bound value.
24                 beta (int): The lower bound value.
25                 stop_event (threading.Event): Event used to kill search from an
26                 external class.
27
28             Returns:
29                 tuple[int, Move]: The best score and the best move found.
30             """
31         hash = board.to_hash()
32         score, move = self._table.get_entry(hash, depth, alpha, beta)

```

```

31     if score is not None:
32         self._stats['cache_hits'] += 1
33         self._stats['nodes'] += 1
34
35     return score, move
36 else:
37     # If board hash entry not found in cache, run a full search
38     score, move = super().search(board, depth, alpha, beta, stop_event,
39                                   hint)
40     self._table.insert_entry(score, move, hash, depth, alpha, beta)
41
42     return score, move
43
44 class TTMinimaxCPU(TranspositionTableMixin, ABMinimaxCPU):
45     def initialise_stats(self):
46         """
47             Initialises cache statistics to be logged.
48         """
49         super().initialise_stats()
50         self._stats['cache_hits'] = 0
51
52     def print_stats(self, score, move):
53         """
54             Logs the statistics for the search.
55
56             Args:
57                 score (int): The best score found.
58                 move (Move): The best move found.
59
60             # Calculate number of cached entries retrieved as a percentage of all
61             # nodes
62             self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
63             self._stats['nodes'], 3)
64             self._stats['cache_entries'] = len(self._table)
65             super().print_stats(score, move)

```

### 3.6.4 Iterative Deepening

As described in 2.2.2, the depth for each search is increased for each iteration through the for loop, with the best move found on one depth being used as the starting move for the following depth.

`iterative_deepening.py`

```

1  from copy import deepcopy
2  from random import choice
3  from data.states.game.cpu.engines.transposition_table import
4      TranspositionTableMixin
5  from data.states.game.cpu.transposition_table import TranspositionTable
6  from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU
7  from data.managers.logs import initialise_logger
8  from data.utils.enums import Score
9
10 logger = initialise_logger(__name__)
11
12 class IterativeDeepeningMixin:
13     def find_move(self, board, stop_event):
14         """
15             Iterates through increasing depths to find the best move.
16
17             Args:

```

```

17         board (Board): The current board state.
18         stop_event (threading.Event): Event used to kill search from an
19         external class.
20         """
21
22     best_move = None
23
24     for depth in range(1, self._max_depth + 1):
25         self.initialise_stats()
26
27         # Use copy of board as search can be terminated before all tested
28         # moves are undone
28         board_copy = deepcopy(board)
29
30         try:
31             best_score, best_move = self.search(board_copy, depth, -Score.
31             INFINITE, Score.INFINITE, stop_event, hint=best_move)
32         except TimeoutError:
33             # If allocated time is up, use previous depth's best move
34             logger.info(f'Terminated CPU search early at depth {depth}. Using
34             existing best move: {best_move}')
35
36         if best_move is None:
37             # If search is terminated at depth 0, use random move
38             best_move = choice(board_copy.generate_all_moves())
39             logger.warning('CPU terminated before any best move found!
39             Using random move.')
40
41         break
42
43         self._stats['ID_depth'] = depth
44
45         if self._verbose:
46             self.print_stats(best_score, best_move)
47
48         self._callback(best_move)
49
50 class IDMinimaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABMinimaxCPU)
50 :
51     def initialise_stats(self):
52         super().initialise_stats()
53         self._stats['cache_hits'] = 0
54
55     def print_stats(self, score, move):
56         self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
56             self._stats['nodes'], 3)
57         self._stats['cache_entries'] = len(self._table._table)
58         super().print_stats(score, move)

```

### 3.6.5 Evaluator

As described in Section 2.2.4, I have opted to separate the evaluation class into separate methods for each aspect of the evaluation, and amalgamating all of them to form one unified `evaluate` function, as this allows me to debug each function easily.

`evaluator.py`

```

1 from data.helpers.bitboard_helpers import pop_count, occupied_squares,
   bitboard_to_index
2 from data.states.game.components.psqt import PSQT, FLIP

```

```
3 from data.utils.enums import Colour, Piece, Score
4 from data.managers.logs import initialise_logger
5
6 logger = initialise_logger(__name__)
7
8 class Evaluator:
9     def __init__(self, verbose=True):
10         self._verbose = verbose
11
12     def evaluate(self, board, absolute=False):
13         """
14             Evaluates and returns a numerical score for the board state.
15
16         Args:
17             board (Board): The current board state.
18             absolute (bool): Whether to always return the absolute score from the
19             active colour's perspective (for NegaMax).
20
21         Returns:
22             int: Score representing advantage/disadvantage for the player.
23         """
24         blue_score = (
25             self.evaluate_material(board, Colour.BLUE),
26             self.evaluate_position(board, Colour.BLUE),
27             self.evaluate_mobility(board, Colour.BLUE),
28             self.evaluate_pharaoh_safety(board, Colour.BLUE)
29         )
30
31         red_score = (
32             self.evaluate_material(board, Colour.RED),
33             self.evaluate_position(board, Colour.RED),
34             self.evaluate_mobility(board, Colour.RED),
35             self.evaluate_pharaoh_safety(board, Colour.RED)
36         )
37
38         if self._verbose:
39             logger.info(f'Material: {blue_score[0]} | {red_score[0]}')
40             logger.info(f'Position: {blue_score[1]} | {red_score[1]}')
41             logger.info(f'Mobility: {blue_score[2]} | {red_score[2]}')
42             logger.info(f'Safety: {blue_score[3]} | {red_score[3]}')
43             logger.info(f'Overall score: {sum(blue_score) - sum(red_score)}')
44
45         if absolute and board.get_active_colour() == Colour.RED:
46             return sum(red_score) - sum(blue_score)
47         else:
48             return sum(blue_score) - sum(red_score)
49
50     def evaluate_material(self, board, colour):
51         """
52             Evaluates the material score for a given colour.
53
54         Args:
55             board (Board): The current board state.
56             colour (Colour): The colour to evaluate.
57
58         Returns:
59             int: Sum of all piece scores.
60         """
61         return (
62             Score.SPHINX * board.bitboards.get_piece_count(Piece.SPHINX, colour) +
63             Score.PYRAMID * board.bitboards.get_piece_count(Piece.PYRAMID, colour)
64         )
```

```

63         Score.ANUBIS * board.bitboards.get_piece_count(Piece.ANUBIS, colour) +
64         Score.SCARAB * board.bitboards.get_piece_count(Piece.SCARAB, colour)
65     )
66
67     def evaluate_position(self, board, colour):
68         """
69             Evaluates the positional score for a given colour.
70
71         Args:
72             board (Board): The current board state.
73             colour (Colour): The colour to evaluate.
74
75         Returns:
76             int: Score representing positional advantage/disadvantage.
77         """
78         score = 0
79
80         for piece in Piece:
81             if piece == Piece.SPHINX:
82                 continue
83
84             piece_bitboard = board.bitboards.get_piece_bitboard(piece, colour)
85
86             for bitboard in occupied_squares(piece_bitboard):
87                 index = bitboard_to_index(bitboard)
88                 # Flip PSQT if using from blue player's perspective
89                 index = FLIP[index] if colour == Colour.BLUE else index
90
91             score += PSQT[piece][index] * Score.POSITION
92
93         return score
94
95     def evaluate_mobility(self, board, colour):
96         """
97             Evaluates the mobility score for a given colour.
98
99         Args:
100             board (Board): The current board state.
101             colour (Colour): The colour to evaluate.
102
103         Returns:
104             int: Score on numerical representation of mobility.
105         """
106         number_of_moves = board.get_mobility(colour)
107         return number_of_moves * Score.MOVE
108
109     def evaluate_pharaoh_safety(self, board, colour):
110         """
111             Evaluates the safety of the Pharaoh for a given colour.
112
113         Args:
114             board (Board): The current board state.
115             colour (Colour): The colour to evaluate.
116
117         Returns:
118             int: Score representing mobility of the Pharaoh.
119         """
120         pharaoh_bitboard = board.bitboards.get_piece_bitboard(Piece.PHARAOH,
121         colour)
122
123         if pharaoh_bitboard:
124             pharaoh_available_moves = pop_count(board.get_valid_squares(

```

```

    pharaoh_bitboard, colour))
124         return (8 - pharaoh_available_moves) * Score.PHAROAH_SAFETY
125     else:
126         return 0

```

### 3.6.6 Multithreading

As described in Section 2.2.6, when the game starts, a `CPUThread` object is created with the selected CPU. The `start` method is called whenever it is the CPU's turn, passing the board as an argument to work on. Each run is also given a random ID, to ensure that only the right search is able to be forcibly terminated early. Using **multithreading** allows the game MVC to continue running smoothly while the CPU calculates its moves on a separate thread.

`cpu_thread.py`

```

1 import threading
2 import time
3 from data.managers.logs import initialise_logger
4
5 logger = initialise_logger(__name__)
6
7 class CPUThread(threading.Thread):
8     def __init__(self, cpu, verbose=False):
9         super().__init__()
10        self._stop_event = threading.Event()
11        self._running = True
12        self._verbose = verbose
13        self.daemon = True
14
15        self._board = None
16        self._cpu = cpu
17        self._id = None
18
19    def kill_thread(self):
20        """
21            Kills the CPU and terminates the thread by stopping the run loop.
22        """
23        self.stop_cpu(force=True)
24        self._running = False
25
26    def stop_cpu(self, id=None, force=False):
27        """
28            Kills the CPU's move search.
29
30            Args:
31                id (int, optional): Id of search to kill, only kills if matching.
32                force (bool, optional): Forcibly kill search regardless of id.
33        """
34
35        if self._id == id or force:
36            self._stop_event.set()
37            self._board = None
38
39    def start_cpu(self, board, id=None):
40        """
41            Starts the CPU's move search.
42
43            Args:
44                board (Board): The current board state.
45                id (int, optional): Id of current search.
46        """
47
48        self._stop_event.clear()

```

```

47         self._board = board
48         self._id = id
49
50     def run(self):
51         """
52             Periodically checks if the board variable is set.
53             If it is, then starts CPU search.
54         """
55         while self._running:
56             if self._board and self._cpu:
57                 self._cpu.find_move(self._board, self._stop_event)
58                 self.stop_cpu()
59             else:
60                 time.sleep(1)
61                 if self._verbose:
62                     logger.debug(f'(CPUThread.run) Thread {threading.get_native_id} idling... ')

```

### 3.6.7 Zobrist Hashing

As described in Section 2.2.2, the `ZobristHasher` class provides methods to successively **hash** a given board for every move played, with the initial hash being generated in the `Board` class.

`zobrist_hasher.py`

```

1  from random import randint
2  from data.helpers.bitboard_helpers import bitboard_to_index
3  from data.utils.enums import Piece, Colour, Rotation
4
5  # Initialise random values for each piece type on every square
6  # (5 x 2 colours) pieces + 4 rotations, for 80 squares
7  zobrist_table = [[randint(0, 2 ** 64) for i in range(14)] for j in range(80)]
8  # Hash for when the red player's move
9  red_move_hash = randint(0, 2 ** 64)
10
11 # Maps piece to the correct random value
12 piece_lookup = {
13     Colour.BLUE: {
14         piece: i for i, piece in enumerate(Piece)
15     },
16     Colour.RED: {
17         piece: i + 5 for i, piece in enumerate(Piece)
18     },
19 }
20
21 # Maps rotation to the correct random value
22 rotation_lookup = {
23     rotation: i + 10 for i, rotation in enumerate(Rotation)
24 }
25
26 class ZobristHasher:
27     def __init__(self):
28         self.hash = 0
29
30     def get_piece_hash(self, index, piece, colour):
31         """
32             Gets the random value for the piece type on the given square.
33
34             Args:
35                 index (int): The index of the square.
36                 piece (Piece): The piece on the square.

```

```

37         colour (Colour): The colour of the piece.
38
39     Returns:
40         int: A 64-bit value.
41     """
42     piece_index = piece_lookup[colour][piece]
43     return zobrist_table[index][piece_index]
44
45     def get_rotation_hash(self, index, rotation):
46         """
47             Gets the random value for the rotation on the given square.
48
49         Args:
50             index (int): The index of the square.
51             rotation (Rotation): The rotation on the square.
52             colour (Colour): The colour of the piece.
53
54         Returns:
55             int: A 64-bit value.
56         """
57     rotation_index = rotation_lookup[rotation]
58     return zobrist_table[index][rotation_index]
59
60     def apply_piece_hash(self, bitboard, piece, colour):
61         """
62             Updates the Zobrist hash with a new piece.
63
64         Args:
65             bitboard (int): The bitboard representation of the square.
66             piece (Piece): The piece on the square.
67             colour (Colour): The colour of the piece.
68         """
69     index = bitboard_to_index(bitboard)
70     piece_hash = self.get_piece_hash(index, piece, colour)
71     self.hash ^= piece_hash
72
73     def apply_rotation_hash(self, bitboard, rotation):
74         """
75             Updates the Zobrist hash with a new rotation.
76
77         Args:
78             bitboard (int): The bitboard representation of the square.
79             rotation (Rotation): The rotation on the square.
80         """
81     index = bitboard_to_index(bitboard)
82     rotation_hash = self.get_rotation_hash(index, rotation)
83     self.hash ^= rotation_hash
84
85     def apply_red_move_hash(self):
86         """
87             Applies the Zobrist hash for the red player's move.
88         """
89     self.hash ^= red_move_hash

```

### 3.6.8 Cache

As described in Section 2.2.2, the `TranspositionTable` class maintains an internal hash map to store already evaluated board positions. Since I have chosen to use a dictionary instead of an array, the Zobrist hash for the board can be used as the keys for the dictionary as is, as it doesn't correspond to the index position as will be the case if I use an array.

## transposition\_table.py

```

1  from data.utils.enums import TranspositionFlag
2
3  class TranspositionEntry:
4      def __init__(self, score, move, flag, hash_key, depth):
5          self.score = score
6          self.move = move
7          self.flag = flag
8          self.hash_key = hash_key
9          self.depth = depth
10
11 class TranspositionTable:
12     def __init__(self, max_entries=100000):
13         self._max_entries = max_entries
14         self._table = dict()
15
16     def calculate_entry_index(self, hash_key):
17         """
18             Gets the dictionary key for a given Zobrist hash.
19
20         Args:
21             hash_key (int): A Zobrist hash.
22
23         Returns:
24             int: Key for the given hash.
25         """
26         return hash_key
27
28     def insert_entry(self, score, move, hash_key, depth, alpha, beta):
29         """
30             Inserts an entry into the transposition table.
31
32         Args:
33             score (int): The evaluation score.
34             move (Move): The best move found.
35             hash_key (int): The Zobrist hash key.
36             depth (int): The depth of the search.
37             alpha (int): The upper bound value.
38             beta (int): The lower bound value.
39
40         Raises:
41             Exception: Invalid depth or score.
42         """
43         if depth == 0 or alpha < score < beta:
44             flag = TranspositionFlag.EXACT
45             score = score
46         elif score <= alpha:
47             flag = TranspositionFlag.UPPER
48             score = alpha
49         elif score >= beta:
50             flag = TranspositionFlag.LOWER
51             score = beta
52         else:
53             raise Exception('(TranspositionTable.insert_entry)')
54
55         self._table[self.calculate_entry_index(hash_key)] = TranspositionEntry(
56             score, move, flag, hash_key, depth)
57
58         if len(self._table) > self._max_entries:
59             # Removes the longest-existing entry to free up space for more up-to-
60             # date entries

```

```

59         # Expression to remove leftmost item taken from https://docs.python.
60         #org/3/library/collections.html#ordereddict-objects
61         (k := next(iter(self._table))), self._table.pop(k))
62
63     def get_entry(self, hash_key, depth, alpha, beta):
64         """
65             Gets an entry from the transposition table.
66
67             Args:
68                 hash_key (int): The Zobrist hash key.
69                 depth (int): The depth of the search.
70                 alpha (int): The alpha value for pruning.
71                 beta (int): The beta value for pruning.
72
73             Returns:
74                 tuple[int, Move] | tuple[None, None]: The evaluation score and the
75                 best move found, if entry exists.
76         """
77         index = self.calculate_entry_index(hash_key)
78
79         if index not in self._table:
80             return None, None
81
82         entry = self._table[index]
83
84         if entry.hash_key == hash_key and entry.depth >= depth:
85             if entry.flag == TranspositionFlag.EXACT:
86                 return entry.score, entry.move
87
88             if entry.flag == TranspositionFlag.LOWER and entry.score >= beta:
89                 return entry.score, entry.move
90
91         if entry.flag == TranspositionFlag.UPPER and entry.score <= alpha:
92             return entry.score, entry.move
93
94         return None, None

```

## 3.7 States

To switch between different screens, I have decided to use a state machine design pattern. This ensures that there is only one main game loop controlling movement between states, handled with the `Control` object. All `State` object contain a `next` and `previous` attribute to tell the `Control` class which screen to switch to, which also calls all `State` methods accordingly.

The `startup` method is called when switched to a new state, and `cleanup` when exiting. Within the `startup` function, the state widgets dictionary is passed into a `WidgetGroup` object. The `process_event` method is called on the `WidgetGroup` every frame to process user input, and handle the returned events accordingly. The `WidgetGroup` object can therefore be thought of as a controller, and the state as the model, and the widgets as the view.

### 3.7.1 Review

The `Review` state uses this logic to allow users to scroll through moves in their past games. All moves are stored in two stacks, as described in Section 2.3.3, and exchanged using `pop` and `append` (`push`) methods.

`review.py`

```

1 import pygame
2 from collections import deque
3 from data.states.game.components.capture_draw import CaptureDraw
4 from data.states.game.components.piece_group import PieceGroup
5 from data.states.game.components.laser_draw import LaserDraw
6 from data.helpers.bitboard_helpers import bitboard_to_coords
7 from data.helpers.browser_helpers import get_winner_string
8 from data.states.review.widget_dict import REVIEW_WIDGETS
9 from data.states.game.components.board import Board
10 from data.utils.event_types import ReviewEventType
11 from data.components.game_entry import GameEntry
12 from data.managers.logs import initialise_logger
13 from data.utils.constants import ShaderType
14 from data.managers.window import window
15 from data.utils.assets import MUSIC
16 from data.utils.enums import Colour
17 from data.control import _State
18
19 logger = initialise_logger(__name__)
20
21 class Review(_State):
22     def __init__(self):
23         super().__init__()
24
25         self._moves = deque()
26         self._popped_moves = deque()
27         self._game_info = {}
28
29         self._board = None
30         self._piece_group = None
31         self._laser_draw = None
32         self._capture_draw = None
33
34     def cleanup(self):
35         """
36             Cleanup function. Clears shader effects.
37         """
38         super().cleanup()
39
40         window.clear_apply_arguments(ShaderType.BLOOM)
41         window.clear_effect(ShaderType.RAYS)
42
43         return None
44
45     def startup(self, persist):
46         """
47             Startup function. Initialises all objects, widgets and game data.
48
49             Args:
50                 persist (dict): Dict containing game entry data.
51             """
52         super().startup(REVIEW_WIDGETS, MUSIC['review'])
53
54         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.BACKGROUND_WAVES)
55         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
56         REVIEW_WIDGETS['help'].kill()
57
58         self._moves = deque(GameEntry.parse_moves(persist.pop('moves', '')))
59         self._popped_moves = deque()
60         self._game_info = persist

```

```

61
62     self._board = Board(self._game_info['start_fen_string'])
63     self._piece_group = PieceGroup()
64     self._laser_draw = LaserDraw(self.board_position, self.board_size)
65     self._capture_draw = CaptureDraw(self.board_position, self.board_size)
66
67     self.initialise_widgets()
68     self.simulate_all_moves()
69     self.refresh_pieces()
70     self.refresh_widgets()
71
72     self.draw()
73
74     @property
75     def board_position(self):
76         return REVIEW_WIDGETS['chessboard'].position
77
78     @property
79     def board_size(self):
80         return REVIEW_WIDGETS['chessboard'].size
81
82     @property
83     def square_size(self):
84         return self.board_size[0] / 10
85
86     def initialise_widgets(self):
87         """
88             Initializes the widgets for a new game.
89         """
90         REVIEW_WIDGETS['move_list'].reset_move_list()
91         REVIEW_WIDGETS['move_list'].kill()
92         REVIEW_WIDGETS['scroll_area'].set_image()
93
94         REVIEW_WIDGETS['winner_text'].set_text(f'WINNER: {get_winner_string(self._game_info["winner"])}')
95         REVIEW_WIDGETS['blue_piece_display'].reset_piece_list()
96         REVIEW_WIDGETS['red_piece_display'].reset_piece_list()
97
98         if self._game_info['time_enabled']:
99             REVIEW_WIDGETS['timer_disabled_text'].kill()
100        else:
101            REVIEW_WIDGETS['blue_timer'].kill()
102            REVIEW_WIDGETS['red_timer'].kill()
103
104    def refresh_widgets(self):
105        """
106            Refreshes the widgets after every move.
107        """
108        REVIEW_WIDGETS['move_number_text'].set_text(f'MOVE NO: {(len(self._moves) / 2:.1f) / {(len(self._moves) + len(self._popped_moves)) / 2:.1f}}')
109        REVIEW_WIDGETS['move_colour_text'].set_text(f'{self.calculate_colour().name} TO MOVE')
110
111        if self._game_info['time_enabled']:
112            if len(self._moves) == 0:
113                REVIEW_WIDGETS['blue_timer'].set_time(float(self._game_info['time']) * 60 * 1000)
114                REVIEW_WIDGETS['red_timer'].set_time(float(self._game_info['time']) * 60 * 1000)
115            else:
116                REVIEW_WIDGETS['blue_timer'].set_time(float(self._moves[-1]['blue_time']) * 60 * 1000)

```

```

117         REVIEW_WIDGETS['red_timer'].set_time(float(self._moves[-1]['
118             red_time']) * 60 * 1000)
119
120     REVIEW_WIDGETS['scroll_area'].set_image()
121
122     def refresh_pieces(self):
123         """
124             Refreshes the pieces on the board.
125         """
126         self._piece_group.initialise_pieces(self._board.get_piece_list(), self.
127             board_position, self.board_size)
128
129     def simulate_all_moves(self):
130         """
131             Simulates all moves at the start of every game to obtain laser results and
132             fill up piece display and move list widgets.
133         """
134         for index, move_dict in enumerate(self._moves):
135             laser_result = self._board.apply_move(move_dict['move'], fire_laser=
136                 True)
137             self._moves[index]['laser_result'] = laser_result
138
139             if laser_result.hit_square_bitboard:
140                 if laser_result.piece_colour == Colour.BLUE:
141                     REVIEW_WIDGETS['red_piece_display'].add_piece(laser_result.
142                         piece_hit)
143                 elif laser_result.piece_colour == Colour.RED:
144                     REVIEW_WIDGETS['blue_piece_display'].add_piece(laser_result.
145                         piece_hit)
146
147             REVIEW_WIDGETS['move_list'].append_to_move_list(move_dict['
148                 unparsed_move'])
149
150     def calculate_colour(self):
151         """
152             Calculates the current active colour to move.
153
154             Returns:
155                 Colour: The current colour to move.
156
157             if self._game_info['start_fen_string'][-1].lower() == 'b':
158                 initial_colour = Colour.BLUE
159             elif self._game_info['start_fen_string'][-1].lower() == 'r':
160                 initial_colour = Colour.RED
161
162             if len(self._moves) % 2 == 0:
163                 return initial_colour
164             else:
165                 return initial_colour.get_flipped_colour()
166
167     def handle_move(self, move, add_piece=True):
168         """
169             Handles applying or undoing a move.
170
171             Args:
172                 move (dict): The move to handle.
173                 add_piece (bool): Whether to add the captured piece to the display.
174             Defaults to True.
175
176             laser_result = move['laser_result']
177             active_colour = self.calculate_colour()
178             self._laser_draw.add_laser(laser_result, laser_colour=active_colour)

```

```

171     if laser_result.hit_square_bitboard:
172         if laser_result.piece_colour == Colour.BLUE:
173             if add_piece:
174                 REVIEW_WIDGETS['red_piece_display'].add_piece(laser_result.
175                 piece_hit)
176             else:
177                 REVIEW_WIDGETS['red_piece_display'].remove_piece(laser_result.
178                 piece_hit)
179             elif laser_result.piece_colour == Colour.RED:
180                 if add_piece:
181                     REVIEW_WIDGETS['blue_piece_display'].add_piece(laser_result.
182                     piece_hit)
183             else:
184                 REVIEW_WIDGETS['blue_piece_display'].remove_piece(laser_result.
185                 piece_hit)
186
187         self._capture_draw.add_capture(
188             laser_result.piece_hit,
189             laser_result.piece_colour,
190             laser_result.piece_rotation,
191             bitboard_to_coords(laser_result.hit_square_bitboard),
192             laser_result.laser_path[0][0],
193             active_colour,
194             shake=False
195         )
196
197     def update_laser_mask(self):
198         """
199             Updates the laser mask for the light rays effect.
200         """
201
202         temp_surface = pygame.Surface(window.size, pygame.SRCALPHA)
203         self._piece_group.draw(temp_surface)
204         mask = pygame.mask.from_surface(temp_surface, threshold=127)
205         mask_surface = mask.to_surface(unsetcolor=(0, 0, 0, 255), setcolor=(255,
206             0, 0, 255))
207
208         window.set_apply_arguments(ShaderType.RAYS, occlusion=mask_surface)
209
210     def get_event(self, event):
211         """
212             Processes Pygame events.
213
214             Args:
215                 event (pygame.event.Event): The event to handle.
216             """
217
218         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
219             REVIEW_WIDGETS['help'].kill()
220
221         widget_event = self._widget_group.process_event(event)
222
223         if widget_event is None:
224             return
225
226         match widget_event.type:
227             case None:
228                 return
229
230             case ReviewEventType.MENU_CLICK:
231                 self.next = 'menu'
232                 self.done = True
233
234

```

```

228         case ReviewEventType.PREVIOUS_CLICK:
229             if len(self._moves) == 0:
230                 return
231
232             # Pop last applied move off first stack
233             move = self._moves.pop()
234             # Pushed onto second stack
235             self._popped_moves.append(move)
236
237             # Undo last applied move
238             self._board.undo_move(move['move'], laser_result=move['laser_result'])
239             self.handle_move(move, add_piece=False)
240             REVIEW_WIDGETS['move_list'].pop_from_move_list()
241
242             self.refresh_pieces()
243             self.refresh_widgets()
244             self.update_laser_mask()
245
246         case ReviewEventType.NEXT_CLICK:
247             if len(self._popped_moves) == 0:
248                 return
249
250             # Peek at second stack to get last undone move
251             move = self._popped_moves[-1]
252
253             # Reapply last undone move
254             self._board.apply_move(move['move'])
255             self.handle_move(move, add_piece=True)
256             REVIEW_WIDGETS['move_list'].append_to_move_list(move['unparsed_move'])
257
258             # Pop last undone move from second stack
259             self._popped_moves.pop()
260             # Push onto first stack
261             self._moves.append(move)
262
263             self.refresh_pieces()
264             self.refresh_widgets()
265             self.update_laser_mask()
266
267         case ReviewEventType.HELP_CLICK:
268             self._widget_group.add(REVIEW_WIDGETS['help'])
269             self._widget_group.handle_resize(window.size)
270
271     def handle_resize(self):
272         """
273             Handles resizing of the window.
274         """
275         super().handle_resize()
276         self._piece_group.handle_resize(self.board_position, self.board_size)
277         self._laser_draw.handle_resize(self.board_position, self.board_size)
278         self._capture_draw.handle_resize(self.board_position, self.board_size)
279
280         if self._laser_draw.firing:
281             self.update_laser_mask()
282
283     def draw(self):
284         """
285             Draws all components onto the window screen.
286         """
287         self._capture_draw.update()

```

```

288         self._widget_group.draw()
289         self._piece_group.draw(window.screen)
290         self._laser_draw.draw(window.screen)
291         self._capture_draw.draw(window.screen)

```

## 3.8 Database

This section outlines my database implementation using the Python module sqlite3.

### 3.8.1 DDL

As mentioned in Section 2.3.1, the `migrations` directory contains a collection of Python scripts that edit the game table schema. The files are named with a description of their changes and datetime for organisational purposes.

`create_games_table_19112024.py`

```

1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     """
8     Upgrade function to create games table.
9     """
10    connection = sqlite3.connect(database_path)
11    cursor = connection.cursor()
12
13    cursor.execute('''
14        CREATE TABLE games(
15            id INTEGER PRIMARY KEY,
16            cpu_enabled INTEGER NOT NULL,
17            cpu_depth INTEGER,
18            winner INTEGER,
19            time_enabled INTEGER NOT NULL,
20            time REAL,
21            number_of_ply INTEGER NOT NULL,
22            moves TEXT NOT NULL
23        )
24    ''')
25
26    connection.commit()
27    connection.close()
28
29 def downgrade():
30     """
31     Downgrade function to revert table creation.
32     """
33    connection = sqlite3.connect(database_path)
34    cursor = connection.cursor()
35
36    cursor.execute('''
37        DROP TABLE games
38    ''')
39
40    connection.commit()
41    connection.close()
42

```

```

43 upgrade()
44 # downgrade()

Using the ALTER command allows me to rename table columns.

change_fen_string_column_name_23122024.py

1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     """
8     Upgrade function to rename fen_string column.
9     """
10    connection = sqlite3.connect(database_path)
11    cursor = connection.cursor()
12
13    cursor.execute('''
14        ALTER TABLE games RENAME COLUMN fen_string TO final_fen_string
15    ''')
16
17    connection.commit()
18    connection.close()
19
20 def downgrade():
21     """
22     Downgrade function to revert fen_string column renaming.
23     """
24    connection = sqlite3.connect(database_path)
25    cursor = connection.cursor()
26
27    cursor.execute('''
28        ALTER TABLE games RENAME COLUMN final_fen_string TO fen_string
29    ''')
30
31    connection.commit()
32    connection.close()
33
34 upgrade()
35 # downgrade()

```

### 3.8.2 DML

As described in Section 2.3.1, this file provides functions to help modify the database, with **Aggregate** and **Window** commands used to retrieve the number of rows and sort them to be returned. `database_helpers.py`

```

1 import sqlite3
2 from pathlib import Path
3 from datetime import datetime
4
5 database_path = (Path(__file__).parent / '../database/database.db').resolve()
6
7 def insert_into_games(game_entry):
8     """
9     Inserts a new row into games table.
10
11     Args:

```

```
12     game_entry (GameEntry): GameEntry object containing game information.
13 """
14     connection = sqlite3.connect(database_path, detect_types=sqlite3.
15         PARSE_DECLTYPES)
16     connection.row_factory = sqlite3.Row
17     cursor = connection.cursor()
18
19     # Datetime added for created_dt column
20     game_entry = (*game_entry, datetime.now())
21
22     cursor.execute('''
23         INSERT INTO games (cpu_enabled, cpu_depth, winner, time_enabled, time,
24         number_of_ply, moves, start_fen_string, final_fen_string, created_dt)
25         VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
26     ''', game_entry)
27
28     connection.commit()
29
30     # Return inserted row
31     cursor.execute('''
32         SELECT * FROM games WHERE id = LAST_INSERT_ROWID()
33     ''')
34     inserted_row = cursor.fetchone()
35
36     connection.close()
37
38     return dict(inserted_row)
39
40 def get_all_games():
41 """
42     Get all rows in games table.
43
44     Returns:
45         list[dict]: List of game entries represented as dictionaries.
46 """
47     connection = sqlite3.connect(database_path, detect_types=sqlite3.
48         PARSE_DECLTYPES)
49     connection.row_factory = sqlite3.Row
50     cursor = connection.cursor()
51
52     cursor.execute('''
53         SELECT * FROM games
54     ''')
55     games = cursor.fetchall()
56
57     connection.close()
58
59     return [dict(game) for game in games]
60
61 def delete_all_games():
62 """
63     Delete all rows in games table.
64 """
65     connection = sqlite3.connect(database_path)
66     cursor = connection.cursor()
67
68     cursor.execute('''
69         DELETE FROM games
70     ''')
71
72     connection.commit()
73     connection.close()
```

```

71
72 def delete_game(id):
73     """
74     Deletes specific row in games table using id attribute.
75
76     Args:
77         id (int): Primary key for row.
78     """
79     connection = sqlite3.connect(database_path)
80     cursor = connection.cursor()
81
82     cursor.execute('''
83         DELETE FROM games WHERE id = ?
84     ''', (id,))
85
86     connection.commit()
87     connection.close()
88
89 def get_ordered_games(column, ascend=True, start_row=1, end_row=10):
90     """
91     Get specific number of rows from games table ordered by a specific column(s).
92
93     Args:
94         column (_type_): Column to sort by.
95         ascend (bool, optional): Sort ascending or descending. Defaults to True.
96         start_row (int, optional): First row returned. Defaults to 1.
97         end_row (int, optional): Last row returned. Defaults to 10.
98
99     Raises:
100         ValueError: If ascend argument or column argument are invalid types.
101
102     Returns:
103         list[dict]: List of ordered game entries represented as dictionaries.
104     """
105     if not isinstance(ascend, bool) or not isinstance(column, str):
106         raise ValueError('(database_helpers.get_ordered_games) Invalid input arguments!')
107
108     connection = sqlite3.connect(database_path, detect_types=sqlite3.
109     PARSE_DECLTYPES)
110     connection.row_factory = sqlite3.Row
111     cursor = connection.cursor()
112
113     # Match ascend bool to correct SQL keyword
114     if ascend:
115         ascend_arg = 'ASC'
116     else:
117         ascend_arg = 'DESC'
118
119     # Partition by winner, then order by time and number_of_ply
120     if column == 'winner':
121         cursor.execute(f'''
122             SELECT * FROM
123                 (SELECT ROW_NUMBER() OVER (
124                     PARTITION BY winner
125                     ORDER BY time {ascend_arg}, number_of_ply {ascend_arg}
126                     ) AS row_num, * FROM games)
127             WHERE row_num >= ? AND row_num <= ?
128             ''', (start_row, end_row))
129     else:
130         # Order by time or number_of_ply only
131         cursor.execute(f'''

```

```

131         SELECT * FROM
132             (SELECT ROW_NUMBER() OVER (
133                 ORDER BY {column} {ascend_arg}
134             ) AS row_num, * FROM games)
135             WHERE row_num >= ? AND row_num <= ?
136             !!!, (start_row, end_row))
137
138     games = cursor.fetchall()
139
140     connection.close()
141
142     return [dict(game) for game in games]
143
144 def get_number_of_games():
145     """
146     Returns:
147         int: Number of rows in the games.
148     """
149     connection = sqlite3.connect(database_path)
150     cursor = connection.cursor()
151
152     cursor.execute("""
153         SELECT COUNT(ROWID) FROM games
154     """)
155
156     result = cursor.fetchall()[0][0]
157
158     connection.close()
159
160     return result
161
162 # delete_all_games()

```

## 3.9 Shaders

### 3.9.1 Shader Manager

The `ShaderManager` class is responsible for handling all shader passes, handling the Pygame display, and combining both and drawing the result to the window screen. The class also **inherits** from the `SMPProtocol` class, an **interface** class containing all required `ShaderManager` methods and attributes to aid with syntax highlighting in the fragment shader classes.

Fragment shaders such as `Bloom` are applied by default, and others such as `Ray` are applied during runtime through calling methods on `ShaderManager`, and adding the appropriate fragment shader class to the internal shader pass list.

`shader.py`

```

1 from pathlib import Path
2 from array import array
3 import moderngl
4 from data.shaders.classes import shader_pass_lookup
5 from data.shaders.protocol import SMPProtocol
6 from data.utils.constants import ShaderType
7
8 shader_path = (Path(__file__).parent / '../shaders/').resolve()
9
10 SHADER_PRIORITY = [
11     ShaderType.CRT,
12     ShaderType.SHAKE,

```

```

13     ShaderType.BLOOM,
14     ShaderType.CHROMATIC_ABBREVIATION,
15     ShaderType.RAYS,
16     ShaderType.GRAYSCALE,
17     ShaderType.BASE,
18 ]
19
20 pygame_quad_array = array('f', [
21     -1.0, 1.0, 0.0, 0.0,
22     1.0, 1.0, 1.0, 0.0,
23     -1.0, -1.0, 0.0, 1.0,
24     1.0, -1.0, 1.0, 1.0,
25 ])
26
27 opengl_quad_array = array('f', [
28     -1.0, -1.0, 0.0, 0.0,
29     1.0, -1.0, 1.0, 0.0,
30     -1.0, 1.0, 0.0, 1.0,
31     1.0, 1.0, 1.0, 1.0,
32 ])
33
34 class ShaderManager(SMPProtocol):
35     def __init__(self, ctx: moderngl.Context, screen_size):
36         self._ctx = ctx
37         self._ctx.gc_mode = 'auto'
38
39         self._screen_size = screen_size
40         self._opengl_buffer = self._ctx.buffer(data=opengl_quad_array)
41         self._pygame_buffer = self._ctx.buffer(data=pygame_quad_array)
42         self._shader_list = [ShaderType.BASE]
43
44         self._vert_shaders = {}
45         self._frag_shaders = {}
46         self._programs = {}
47         self._vaos = {}
48         self._textures = {}
49         self._shader_passes = {}
50         self.framebuffers = {}
51
52         self.load_shader(ShaderType.BASE)
53         self.load_shader(ShaderType._CALIBRATE)
54         self.create_framebuffer(ShaderType._CALIBRATE)
55
56     def load_shader(self, shader_type, **kwargs):
57         """
58             Loads a given shader by creating a VAO reading the corresponding .frag
59             file.
60
61             Args:
62                 shader_type (ShaderType): The type of shader to load.
63                 **kwargs: Additional arguments passed when initialising the fragment
64             shader class.
65
66             self._shader_passes[shader_type] = shader_pass_lookup[shader_type](self,
67             **kwargs)
68             self.create_vao(shader_type)
69
70         def clear_shaders(self):
71             """
72                 Clears the shader list, leaving only the base shader.
73
74                 self._shader_list = [ShaderType.BASE]

```

```

72
73     def create_vao(self, shader_type):
74         """
75             Creates a vertex array object (VAO) for the given shader type.
76
77         Args:
78             shader_type (ShaderType): The type of shader.
79         """
80         frag_name = shader_type[1:] if shader_type[0] == '_' else shader_type
81         vert_path = Path(shader_path / 'vertex/base.vert').resolve()
82         frag_path = Path(shader_path / f'fragments/{frag_name}.frag').resolve()
83
84         self._vert_shaders[shader_type] = vert_path.read_text()
85         self._frag_shaders[shader_type] = frag_path.read_text()
86
87         program = self._ctx.program(vertex_shader=self._vert_shaders[shader_type],
88                                     fragment_shader=self._frag_shaders[shader_type])
89         self._programs[shader_type] = program
90
91         if shader_type == ShaderType._CALIBRATE:
92             self._vaos[shader_type] = self._ctx.vertex_array(self._programs[shader_type], [(self._pygame_buffer, '2f 2f', 'vert', 'texCoords')])
93         else:
94             self._vaos[shader_type] = self._ctx.vertex_array(self._programs[shader_type], [(self._opengl_buffer, '2f 2f', 'vert', 'texCoords')])
95
96     def create_framebuffer(self, shader_type, size=None, filter=moderngl.NEAREST):
97         """
98             Creates a framebuffer for the given shader type.
99
100            Args:
101                shader_type (ShaderType): The type of shader.
102                size (tuple[int, int], optional): The size of the framebuffer.
103                    Defaults to screen size.
104                filter (moderngl.Filter, optional): The texture filter. Defaults to
105                    NEAREST.
106            """
107
108            texture_size = size or self._screen_size
109            texture = self._ctx.texture(size=texture_size, components=4)
110            texture.filter = (filter, filter)
111
112            self._textures[shader_type] = texture
113            self.framebuffers[shader_type] = self._ctx.framebuffer(color_attachments=[self._textures[shader_type]])
114
115    def render_to_fbo(self, shader_type, texture, output_fbo=None, program_type=None,
116                     use_image=True, **kwargs):
117        """
118            Applies the shaders and renders the resultant texture to a framebuffer
119            object (FBO).
120
121            Args:
122                shader_type (ShaderType): The type of shader.
123                texture (moderngl.Texture): The texture to render.
124                output_fbo (moderngl.Framebuffer, optional): The output framebuffer.
125                    Defaults to None.
126                program_type (ShaderType, optional): The program type. Defaults to
127                    None.
128                use_image (bool, optional): Whether to use the image uniform. Defaults
129                    to True.
130                **kwargs: Additional uniforms for the fragment shader.
131            """
132

```

```

123         fbo = output_fbo or self._framebuffers[shader_type]
124         program = self._programs[program_type] if program_type else self._programs
125         [shader_type]
126         vao= self._vaos[program_type] if program_type else self._vaos[shader_type]
127
128         fbo.use()
129         texture.use(0)
130
131         if use_image:
132             program['image'] = 0
133         for uniform, value in kwargs.items():
134             program[uniform] = value
135
136         vao.render(mode=moderngl.TRIANGLE_STRIP)
137
138     def apply_shader(self, shader_type, **kwargs):
139         """
140             Applies a shader of the given type and adds it to the list.
141
142         Args:
143             shader_type (ShaderType): The type of shader to apply.
144
145         Raises:
146             ValueError: If the shader is already being applied.
147         """
148         if shader_type in self._shader_list:
149             return
150
151         self.load_shader(shader_type, **kwargs)
152         self._shader_list.append(shader_type)
153
154         # Sort shader list based on the order in SHADER_PRIORITY, so that more
155         # important shaders are applied first
156         self._shader_list.sort(key=lambda shader: -SHADER_PRIORITY.index(shader))
157
158     def remove_shader(self, shader_type):
159         """
160             Removes a shader of the given type from the list.
161
162         Args:
163             shader_type (ShaderType): The type of shader to remove.
164         """
165         if shader_type in self._shader_list:
166             self._shader_list.remove(shader_type)
167
168     def render_output(self):
169         """
170             Renders the final output to the screen.
171
172             # Render to the screen framebuffer
173             self._ctx.screen.use()
174
175             # Take the texture of the last framebuffer to be rendered to, and render
176             # that to the screen framebuffer
177             output_shader_type = self._shader_list[-1]
178             self.get_fbo_texture(output_shader_type).use(0)
179             self._programs[output_shader_type]['image'] = 0
180
181             self._vaos[output_shader_type].render(mode=moderngl.TRIANGLE_STRIP)
182
183     def get_fbo_texture(self, shader_type):
184         """

```

```

182         Gets the texture from the specified shader type's FBO.
183
184     Args:
185         shader_type (ShaderType): The type of shader.
186
187     Returns:
188         moderngl.Texture: The texture from the FBO.
189     """
190     return self.framebuffers[shader_type].color_attachments[0]
191
192 def calibrate_pygame_surface(self, pygame_surface):
193     """
194     Converts the Pygame window surface into an OpenGL texture.
195
196     Args:
197         pygame_surface (pygame.Surface): The finished Pygame surface.
198
199     Returns:
200         moderngl.Texture: The calibrated texture.
201     """
202     texture = self._ctx.texture(pygame_surface.size, 4)
203     texture.filter = (moderngl.NEAREST, moderngl.NEAREST)
204     texture.swizzle = 'BGRA'
205     # Take the Pygame surface's pixel array and draw it to the new texture
206     texture.write(pygame_surface.get_view('1'))
207
208     # ShaderType._CALIBRATE has a VAO containing the pygame_quad_array
209     # coordinates, as Pygame uses different texture coordinates than ModernGL
210     # textures
211     self.render_to_fbo(ShaderType._CALIBRATE, texture)
212     return self.get_fbo_texture(ShaderType._CALIBRATE)
213
214 def draw(self, surface, arguments):
215     """
216     Draws the Pygame surface with shaders applied to the screen.
217
218     Args:
219         surface (pygame.Surface): The final Pygame surface.
220         arguments (dict): A dict of { ShaderType: Args } items, containing
221             keyword arguments for every fragment shader.
222     """
223     self._ctx.viewport = (0, 0, *self._screen_size)
224     texture = self.calibrate_pygame_surface(surface)
225
226     for shader_type in self._shader_list:
227         self._shader_passes[shader_type].apply(texture, **arguments.get(
228             shader_type, {}))
229         texture = self.get_fbo_texture(shader_type)
230
231     self.render_output()
232
233 def __del__(self):
234     """
235     Cleans up ModernGL resources when the ShaderManager object is deleted.
236
237     self.cleanup()
238
239 def cleanup(self):
240     """
241     Cleans up resources used by the ModernGL.
242     Probably unnecessary as the 'auto' garbage collection mode is used.
243     """

```

```

240         self._pygame_buffer.release()
241         self._opengl_buffer.release()
242         for program in self._programs:
243             self._programs[program].release()
244         for texture in self._textures:
245             self._textures[texture].release()
246         for vao in self._vaos:
247             self._vaos[vao].release()
248         for framebuffer in self.framebuffers:
249             self.framebuffers[framebuffer].release()
250
251     def handle_resize(self, new_screen_size):
252         """
253             Handles resizing of the screen.
254
255             Args:
256                 new_screen_size (tuple[int, int]): The new screen size.
257
258             self._screen_size = new_screen_size
259
260             # Recreate all framebuffers to prevent scaling issues
261             for shader_type in self.framebuffers:
262                 filter = self._textures[shader_type].filter[0]
263                 self.create_framebuffer(shader_type, size=self._screen_size, filter=
264                 filter)

```

### 3.9.2 Bloom

The `Bloom` shader effect is a common shader effect giving the illusion of a bright light. It consists of blurred fringes of light extending from the borders of bright areas. This effect can be achieved through obtaining all bright areas of the image, applying a Gaussian blur, and blending the blur additively onto the original image.

My `ShaderManager` class works with this multi-pass shader approach by reading the texture from the last shader's framebuffer for each pass.

#### Extracting bright colours

The `highlight_brightness` fragment shader extracts all colours that are bright enough to exert the bloom effect.

`highlight_brightness.frag`

```

1 # version 330 core
2
3 in vec2 uvs;
4 out vec4 f_colour;
5
6 uniform sampler2D image;
7 uniform float threshold;
8 uniform float intensity;
9
10 void main() {
11     vec4 pixel = texture(image, uvs);
12     // Dot product used to calculate brightness of a pixel from its RGB values
13     // Values taken from https://en.wikipedia.org/wiki/Relative_luminance
14     float brightness = dot(pixel.rgb, vec3(0.2126, 0.7152, 0.0722));
15     float isBright = step(threshold, brightness);
16
17     f_colour = vec4(vec3(pixel.rgb * intensity) * isBright, 1.0);
18 }

```

## Blur

The `Blur` class implements a two-pass **Gaussian blur**. This is preferably over a one-pass blur, as the complexity is  $O(2n)$ , sampling  $n$  pixels twice, as opposed to  $O(n^2)$ . I have implemented this using the ping-pong technique, with the first pass for blurring the image horizontally, and the second pass for blurring vertically, and the resultant textures being passed repeatedly between two framebuffers.

`blur.py`

```

1 from data.shaders.protocol import SMProtocol
2 from data.utils.constants import ShaderType
3
4 BLUR_ITERATIONS = 4
5
6 class _Blur:
7     def __init__(self, shader_manager: SMProtocol):
8         self._shader_manager = shader_manager
9
10    shader_manager.create_framebuffer(ShaderType._BLUR)
11
12    shader_manager.create_framebuffer("blurPing")
13    shader_manager.create_framebuffer("blurPong")
14
15    def apply(self, texture):
16        """
17            Applies Gaussian blur to a given texture.
18
19            Args:
20                texture (moderngl.Texture): Texture to blur.
21        """
22        self._shader_manager.get_fbo_texture("blurPong").write(texture.read())
23
24        for _ in range(BLUR_ITERATIONS):
25            # Apply horizontal blur
26            self._shader_manager.render_to_fbo(
27                ShaderType._BLUR,
28                texture=self._shader_manager.get_fbo_texture("blurPong"),
29                output_fbo=self._shader_manager.framebuffers["blurPing"],
30                passes=5,
31                horizontal=True
32            )
33            # Apply vertical blur
34            self._shader_manager.render_to_fbo(
35                ShaderType._BLUR,
36                texture=self._shader_manager.get_fbo_texture("blurPing"), # Use
37                horizontal blur result as input texture
38                output_fbo=self._shader_manager.framebuffers["blurPong"],
39                passes=5,
40                horizontal=False
41            )
42
43        self._shader_manager.render_to_fbo(ShaderType._BLUR, self._shader_manager.
44        get_fbo_texture("blurPong"))

```

`blur.frag`

```

1 // Modified from https://learnopengl.com/Advanced-Lighting/Bloom
2 #version 330 core
3
4 in vec2 uvs;

```

```

5  out vec4 f_colour;
6
7  uniform sampler2D image;
8  uniform bool horizontal;
9  uniform int passes;
10 uniform float weight[5] = float[] (0.227027, 0.1945946, 0.1216216, 0.054054,
11   0.016216);
12 void main() {
13     vec2 offset = 1.0 / textureSize(image, 0);
14     vec3 result = texture(image, uvs).rgb * weight[0];
15
16     if (horizontal) {
17         for (int i = 1 ; i < passes ; ++i) {
18             result += texture(image, uvs + vec2(offset.x * i, 0.0)).rgb * weight[i];
19         }
20     }
21     else {
22         for (int i = 1 ; i < passes ; ++i) {
23             result += texture(image, uvs + vec2(0.0, offset.y * i)).rgb * weight[i];
24         }
25         result += texture(image, uvs - vec2(0.0, offset.y * i)).rgb * weight[i];
26     }
27 }
28
29 f_colour = vec4(result, 1.0);
30 }
```

## Combining

The `Bloom` class combines the two operations, taking the highlighted areas, blurs them, and adds the RGB values for the final result onto the original texture to simulate bloom.

`bloom.py`

```

1  from data.shaders.classes.highlight_brightness import _HighlightBrightness
2  from data.shaders.classes.highlight_colour import _HighlightColour
3  from data.shaders.protocol import SMProtocol
4  from data.shaders.classes.blur import _Blur
5  from data.utils.constants import ShaderType
6
7  BLOOM_INTENSITY = 0.6
8
9  class Bloom:
10     def __init__(self, shader_manager: SMProtocol):
11         self._shader_manager = shader_manager
12
13         shader_manager.load_shader(ShaderType._BLUR)
14         shader_manager.load_shader(ShaderType._HIGHLIGHT_BRIGHTNESS)
15         shader_manager.load_shader(ShaderType._HIGHLIGHT_COLOUR)
16
17         shader_manager.create_framebuffer(ShaderType.BLOOM)
18         shader_manager.create_framebuffer(ShaderType._BLUR)
19         shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_BRIGHTNESS)
20         shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_COLOUR)
21 
```

```

22     def apply(self, texture, highlight_surface=None, highlight_colours=[], 
23             surface_intensity=BLOOM_INTENSITY, brightness_intensity=BLOOM_INTENSITY, 
24             colour_intensity=BLOOM_INTENSITY):
25         """
26             Applies a bloom effect to a given texture.
27
28             Args:
29                 texture (moderngl.Texture): Texture to apply bloom to.
30                 highlight_surface (pygame.Surface, optional): Surface to use as the
31                 highlights. Defaults to None.
32                 highlight_colours (list[list[int, int, int], ...], optional): Colours
33                 to use as the highlights. Defaults to [].
34                 surface_intensity (_type_, optional): Intensity of bloom applied to
35                 the highlight surface. Defaults to BLOOM_INTENSITY.
36                 brightness_intensity (_type_, optional): Intensity of bloom applied to
37                 the highlight brightness. Defaults to BLOOM_INTENSITY.
38                 colour_intensity (_type_, optional): Intensity of bloom applied to the
39                 highlight colours. Defaults to BLOOM_INTENSITY.
40             """
41
42         if highlight_surface:
43             # Calibrate Pygame surface and apply blur
44             glare_texture = self._shader_manager.calibrate_pygame_surface(
45                 highlight_surface)
46             _Blur(self._shader_manager).apply(glare_texture)
47
48             self._shader_manager.get_fbo_texture(ShaderType._BLUR).use(1)
49             self._shader_manager.render_to_fbo(ShaderType.BLOOM, texture,
50                 blurredImage=1, intensity=surface_intensity)
51
52             # Set bloom-applied texture as the base texture
53             texture = self._shader_manager.get_fbo_texture(ShaderType.BLOOM)
54
55             # Extract bright colours (highlights) from the texture
56             _HighlightBrightness(self._shader_manager).apply(texture, intensity=
57                 brightness_intensity)
58             highlight_texture = self._shader_manager.get_fbo_texture(ShaderType.
59                 _HIGHLIGHT_BRIGHTNESS)
60
61             # Use colour as highlights
62             for colour in highlight_colours:
63                 _HighlightColour(self._shader_manager).apply(texture, old_highlight=
64                     highlight_texture, colour=colour, intensity=colour.intensity)
65                 highlight_texture = self._shader_manager.get_fbo_texture(ShaderType.
66                     _HIGHLIGHT_COLOUR)
67
68             # Apply Gaussian blur to highlights
69             _Blur(self._shader_manager).apply(highlight_texture)
70
71             # Add the pixel values for the highlights onto the base texture
72             self._shader_manager.get_fbo_texture(ShaderType._BLUR).use(1)
73             self._shader_manager.render_to_fbo(ShaderType.BLOOM, texture, blurredImage
74                 =1, intensity=BLOOM_INTENSITY)

```

### 3.9.3 Rays

As described in Section 2.2.5, the `ray` shader is applied whenever the sphinx shoots a laser. It simulates a 2D light source, providing pixel perfect shadows, through the shadow mapping technique outlined in Section 2.2.5. The laser demo seen on the main menu screen is also achieved using the Ray shader, by clamping the angle at which it emits light to a narrower range.

## Occlusion

The occlusion fragment shader processes all pixels with a given colour value as being occluding.  
**occlusion.frag**

```

1 # version 330 core
2
3 in vec2 uvs;
4 out vec4 f_colour;
5
6 uniform sampler2D image;
7 uniform vec3 checkColour;
8
9 void main() {
10     vec4 pixel = texture(image, uvs);
11
12     // If pixel is occluding colour, set pixel to white
13     if (pixel.rgb == checkColour) {
14         f_colour = vec4(1.0, 1.0, 1.0, 1.0);
15     // Else, set pixel to black
16     } else {
17         f_colour = vec4(vec3(0.0), 1.0);
18     }
19 }
```

## Shadowmap

The shadowmap fragment shader takes the occluding texture and creates a 1D shadow map.  
**shadowmap.frag**

```

1 # version 330 core
2
3 #define PI 3.1415926536;
4
5 in vec2 uvs;
6 out vec4 f_colour;
7
8 uniform sampler2D image;
9 uniform float resolution;
10 uniform float THRESHOLD=0.99;
11
12 void main() {
13     float maxDistance = 1.0;
14
15     for (float y = 0.0 ; y < resolution ; y += 1.0) {
16         //rectangular to polar filter
17         float currDistance = y / resolution;
18
19         vec2 norm = vec2(uvs.x, currDistance) * 2.0 - 1.0; // Range from [0, 1] ->
20         [-1, 1]
21         float angle = (1.5 - norm.x) * PI; // Range from [-1, 1] -> [0.5PI, 2.5PI]
22         float radius = (1.0 + norm.y) * 0.5; // Range from [-1, 1] -> [0, 1]
23
24         //coord which we will sample from occlude map
25         vec2 coords = vec2(radius * -sin(angle), radius * -cos(angle)) / 2.0 +
26         0.5;
27
28         // Sample occlusion map
29         vec4 occluding = texture(image, coords);
```

```

29      // If pixel is not occluding (Red channel value below threshold), set
30      maxDistance to current distance
31      // If pixel is occluding, don't change distance
32      // maxDistance therefore is the distance from the center to the nearest
33      // occluding pixel
34      maxDistance = max(maxDistance * step(occluding.r, THRESHOLD), min(
35      maxDistance, currDistance));
36  }
37
38  f_colour = vec4(vec3(maxDistance), 1.0);
39 }
```

## Lightmap

The lightmap shader checks if a pixel is in shadow, blurs the result, and applies the radial light source.

### lightmap.frag

```

1 # version 330 core
2
3 #define PI 3.14159265
4
5 in vec2 uvs;
6 out vec4 f_colour;
7
8 uniform float softShadow;
9 uniform float resolution;
10 uniform float falloff;
11 uniform vec3 lightColour;
12 uniform vec2 angleClamp;
13 uniform sampler2D occlusionMap;
14 uniform sampler2D image;
15
16 vec3 normLightColour = lightColour / 255;
17 vec2 radiansClamp = angleClamp * (PI / 180);
18
19 float sample(vec2 coord, float r) {
20     /*
21     Sample from the 1D distance map.
22
23     Returns:
24     float: 1.0 if sampled radius is greater than the passed radius, 0.0 if not.
25     */
26     return step(r, texture(image, coord).r);
27 }
28
29 void main() {
30     // Cartesian to polar transformation
31     // Range from [0, 1] -> [-1, 1]
32     vec2 norm = uvs.xy * 2.0 - 1.0;
33     float angle = atan(norm.y, norm.x);
34     float r = length(norm);
35
36     // The texture coordinates to sample our 1D lookup texture
37     // Always 0.0 on y-axis, as the texture is 1D
38     float x = (angle + PI) / (2.0 * PI); // Normalise angle to [0, 1]
39     vec2 tc = vec2(x, 0.0);
40
41     // Sample the 1D lookup texture to check if pixel is in light or in shadow
42     // Gives us hard shadows
43     // 1.0 -> in light, 0.0, -> in shadow
```

```

44     float inLight = sample(tc, r);
45     // Clamp angle so that only pixels within the range are in light
46     inLight = inLight * step(angle, radiansClamp.y) * step(radiansClamp.x, angle);
47
48     // Multiply the blur amount by the distance from the center
49     // So that the blurring increases as distance increases
50     float blur = (1.0 / resolution) * smoothstep(0.0, 0.1, r);
51
52     // Use gaussian blur to apply blur effecy
53     float sum = 0.0;
54
55     sum += sample(vec2(tc.x - blur * 4.0, tc.y), r) * 0.05;
56     sum += sample(vec2(tc.x - blur * 3.0, tc.y), r) * 0.09;
57     sum += sample(vec2(tc.x - blur * 2.0, tc.y), r) * 0.12;
58     sum += sample(vec2(tc.x - blur * 1.0, tc.y), r) * 0.15;
59
60     sum += inLight * 0.16;
61
62     sum += sample(vec2(tc.x + blur * 1.0, tc.y), r) * 0.15;
63     sum += sample(vec2(tc.x + blur * 2.0, tc.y), r) * 0.12;
64     sum += sample(vec2(tc.x + blur * 3.0, tc.y), r) * 0.09;
65     sum += sample(vec2(tc.x + blur * 4.0, tc.y), r) * 0.05;
66
67     // Mix with the softShadow uniform to toggle degree of softShadows
68     float finalLight = mix(inLight, sum, softShadow);
69
70     // Multiply the final light value with the distance, to give a radial falloff
71     // Use as the alpha value, with the light colour being the RGB values
72     f_colour = vec4(normLightColour, finalLight * smoothstep(1.0, falloff, r));
73 }

```

## Class

The `Rays` class takes in a texture and array of light information, applies the aforementioned shaders, and blends the final result with the original texture.

`rays.py`

```

1  from data.shaders.classes.lightmap import _Lightmap
2  from data.shaders.classes.blend import _Blend
3  from data.shaders.protocol import SMProtocol
4  from data.shaders.classes.crop import _Crop
5  from data.utils.constants import ShaderType
6
7  class Rays:
8      def __init__(self, shader_manager: SMProtocol, lights):
9          self._shader_manager = shader_manager
10         self._lights = lights
11
12         # Load all necessary shaders
13         shader_manager.load_shader(ShaderType._LIGHTMAP)
14         shader_manager.load_shader(ShaderType._BLEND)
15         shader_manager.load_shader(ShaderType._CROP)
16         shader_manager.create_framebuffer(ShaderType.RAYS)
17
18     def apply(self, texture, occlusion=None, softShadow=0.3):
19         """
20             Applies the light rays effect to a given texture.
21
22         Args:
23             texture (moderngl.Texture): The texture to apply the effect to.

```

```

24         occlusion (pygame.Surface, optional): A Pygame mask surface to use as
25         the occlusion texture. Defaults to None.
26         """
27
28         final_texture = texture
29
29         # Iterate through array containing light information
30         for pos, radius, colour, *args in self._lights:
31             # Topleft of light source square
32             light_topleft = (pos[0] - (radius * texture.size[1] / texture.size[0]))
33             , pos[1] - radius)
34             # Relative size of light compared to texture
35             relative_size = (radius * 2 * texture.size[1] / texture.size[0],
36             radius * 2)
37
38             # Crop texture to light source diameter, and to position light source
39             # at the center
40             _Crop(self._shader_manager).apply(texture, relative_pos=light_topleft,
41             relative_size=relative_size)
42             cropped_texture = self._shader_manager.get_fbo_texture(ShaderType.
43             _CROP)
44
45             if occlusion:
46                 # Calibrate Pygame mask surface and crop it
47                 occlusion_texture = self._shader_manager.calibrate_pygame_surface(
48                 occlusion)
49                 _Crop(self._shader_manager).apply(occlusion_texture, relative_pos=
50                 light_topleft, relative_size=relative_size)
51                 occlusion_texture = self._shader_manager.get_fbo_texture(
52                 ShaderType._CROP)
53             else:
54                 occlusion_texture = None
55
56             # Apply lightmap shader, shadowmap and occlusion are included within
57             # the _Lightmap class
58             _Lightmap(self._shader_manager).apply(cropped_texture, colour,
59             softShadow, occlusion_texture, *args)
60             light_map = self._shader_manager.get_fbo_texture(ShaderType._LIGHTMAP)
61
62             # Blend the final result with the original texture
63             _Blend(self._shader_manager).apply(final_texture, light_map,
64             light_topleft)
65             final_texture = self._shader_manager.get_fbo_texture(ShaderType._BLEND
66         )
67
68         self._shader_manager.render_to_fbo(ShaderType.RAYS, final_texture)

```

# Chapter 4

## Testing

### 4.1 Iterative Testing

I have been playtesting the program throughout the development process to find any bugs and fix them accordingly. However, a few issues have required additional measures.

#### 4.1.1 Minimax

Since minimax is recursive algorithm, debugging it has proven a challenge. I have therefore configured the Python `logging` library to help collect information on the minimax tree for every function call.

`base.py`

```
1  def print_stats(self, score, move):
2      """
3          Prints statistics after traversing tree.
4
5          Args:
6              score (int): Final score obtained after traversal.
7              move (Move): Best move obtained after traversal.
8
9          If self._verbose is False:
10              return
11
12          self._stats['time_taken'] = round(1000 * (time.time() - self._stats['
13              time_taken']), 3)
14          self._stats['ms_per_node'] = round(self._stats['time_taken'] / self._stats
15              ['nodes'], 3)
16
17          # Prints stats across multiple lines
18          if self._verbose is True:
19              logger.info(f'\n\n'
20                          f'{self.__str__()} Search Results:\n'
21                          f'{printer.pformat(self._stats)}\n'
22                          f'Best score: {score}    Best move: {move}\n'
23                          )
24
25          # Prints stats in a compacted format
26          elif self._verbose.lower() == 'compact':
27              logger.info(self._stats)
```

```
26     logger.info(f'Best score: {score}    Best move: {move}')
```

Listing 4.1: BaseCPU Method for logging minimax statistics

### 4.1.2 Migrations

To correct errors made to the `games` table, since recreating it would mean deleting all existing games, I have opted to use migrations to fix bugs by editing the table schema, as shown in Section 3.8.1.

## 4.2 Unit Tests

### 4.2.1 Board Evaluator

To test every aspect of the evaluation function, I have set up some unit tests with custom positions using my editor screen. These positions are designed to test every aspect of the evaluation, along with some obviously imbalanced positions to test the overall accuracy of the evaluation function. All positions are set up to give an advantage to the blue player.

Evaluating	FEN string	Score	Passed
Material	sc9/10/10/4paPa4/5Pa4/10/10/9Sa b	124	✓
Position	sc9/4nanana3/10/10/10/4NaNaNa3/10/9Sa b	66	✓
Mobility	See footnote <sup>1</sup>	196	✓
King Safety	sc4fa3pa/10/10/10/10/10/10/5FaPa2Sa b	3	✓
Combined	See footnote <sup>2</sup>	437	✓

Table 4.1: Board evaluator test results

### 4.2.2 CPU

Similarly, to evaluate the strength of my CPU, I have setup some custom positions that I already know the best continuation of, and run each CPU engine on them to test if they can solve it.

Description	FEN string	Best Move	Passed
Mate in 1	sc9/pafa8/Fa9/10/10/10/10/9Sa b	Rotate J3 clockwise	✓
Mate in 1	sc9/10/10/8faRb/8FaRb/10/9Sa b	Move J3 to J2	✓
Mate in 3	sc9/10/10/8Ra1/7FaRafa/8RaRa/9Ra/9Sa b	Move J2 to I1... <sup>3</sup>	✓
Mate in 3	sc5fcnc2/4Pa4Pc/3pb6/2Pc2ra1pb2/10/pb9/7Pa2/2PdNaFa4Sa b	Move E7 to F7... <sup>4</sup>	✓
Escape Mate	sc9/10/10/10/8faRb/8FaRb/10/9Sa b	Move J3 to J2	✓
Win Material	sc9/10/10/10/8faRb/8FaRb/10/9Sa b	Move J3 to J2	✓

Table 4.2: Transposition Table CPU test results

<sup>1</sup>scpapa7/papapa7/papapa1Pa1Pa1Pa1/10/4Pa1Pa1Pa1/10/4Pa1Pa3/9Sa b

<sup>2</sup>scnclfcncpbb3/pa9/pb1pc1rbpa3Pd/1Pc2Pd4Pc/2Pd1RaRb4/10/7Pa2/2PdNaFaNa3Sa b

<sup>3</sup>2. Move J4 to J5 3. Move J3 to I2

<sup>4</sup>2. Rotate anticlockwise H8 3. Move F7 to G7

### 4.2.3 Shadow Mapping

To test the shadow mapping algorithm, I have set up some occluding objects together with a light source. Since visuals are subjective, me and my client have deemed the following results to be adequate.

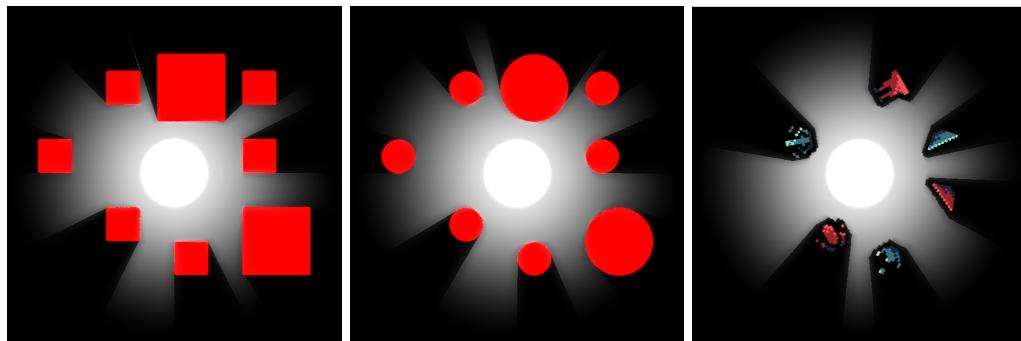


Figure 4.1: Shadow mapping algorithm test (softShadow=0.5, radius=0.5)

## 4.3 Final Tests

### 4.3.1 Objective 1

All laser chess game logic should be properly implemented.

No.	Input	Output	Passed
1	Laser fires on piece	Piece reflects laser	✓
2	Select piece and press rotate clockwise button	Piece rotates clockwise	✓
3	Select piece and adjacent empty square	Piece moves to adjacent square	✓
4	Blue player makes a move	Next move is made by red player	✓
5	Make move	Laser reflects off pieces correctly	✓
6	Position piece with non-reflecting side facing laser	Piece is destroyed	✓
7	Move Pharaoh in front of player	Game displays game over screen	✓
8	Repeat same position three times	Game displays game over screen	✓

### 4.3.2 Objective 2

Save or load game options should be implemented.

No.	Input	Output	Passed
9	Click on copy FEN string button in editor or browser screen	Position formatted as FEN string copied to clipboard	✓
10	Enter browser screen	Program shows list of past games to be scrolled through	✓

No.	Input	Output	Passed
11	Click on previous or next move button in review screen	Board undoes or applies move	✓
12	Enter review screen	Program displays list of past moves, winner and move number	✓

#### 4.3.3 Objective 3

Other board game requirements should be implemented.

No.	Input	Output	Passed
13	Click on draw button	Game ends and shows draw result	✓
14	Click on resign button	Game ends and shows win result for opposite player	✓
15	Enable timer	Timer appears on the left and decrements every second	✓
16	Pause the game	Timer stops decrementing	✓

#### 4.3.4 Objective 4

Game settings and config should be customisable.

No.	Input	Output	Passed
17	Set primary board colour to blue	Alternating board squares appear blue	✓
18	Select CPU player	Every other move is played by the CPU	✓
19	Input timer duration in config screen	Inputted timer duration appears on the left of the game screen	✓
21	Press starting colour button to set starting colour to red	Starting move is played by red player	✓
21	Set up custom board layout in editor screen	Game starts with custom board position	✓

#### 4.3.5 Objective 5

Game UI should improve player experience.

No.	Input	Output	Passed
27	Click the play button on the main menu screen	Program switches to the config screen	✓
28	Click main menu button	Program switches to the menu screen	✓
29	Click help button	Help overlay appears	✓
30	Resize program window	GUI Widgets resize continuously	✓

No.	Input	Output	Passed
31	Drag program window	Program continues running	✓

## 4.4 Videos

# Chapter 5

## Evaluation

### 5.1 Objectives

#### 5.1.1 Objective 1

All laser chess game logic should be properly implemented. Refer to Figure A.4.

Both the play and review screens display a 10x8 laser chess board. Players can move and rotate pieces on their respective turns, with the laser firing and destroying pieces automatically. The game ends when a player draws, resigns, has their pharaoh destroyed or reaches three-fold repetition.

Objective 1 is fulfilled, the game plays successfully. A possible improvement would be an option to play against local hosts. This was not a requested or planned feature, but would be very handy for a two-player game.

#### 5.1.2 Objective 2

Save or load game options should be implemented. Refer to Figures A.6 and A.7.

In the browser screen, users can scroll through all previous games, with the option to delete or review them, or copy their FEN string. Games can be sorted, and pages of games can be scrolled through. The game positions are encoded using a custom FEN string format, and games are saved as rows in a local SQLite database table. In the review screen, users can scroll through all the moves of a previous game. The right side bar displays information such as the winner and move number, and the timer, pieces and move list updates accordingly.

Objective 2 is fulfilled. A possible improvement would be to add the option of loading up a position to resume playing straight from the review screen. This would be a convenient feature, but is not a priority, as users can currently copy the FEN string from the browser screen to the config screen and load the position there as an alternative option.

#### 5.1.3 Objective 3

Other board game requirements should be implemented. Refer to Figure A.4.

In addition to the core game mechanics, other ancillary aspects are added as well. The piece display correctly updates to show all destroyed pieces. Timers can be enabled and disabled, decrement correctly, and end the game when they run out. Buttons also exist for drawing and resigning the game. The move list and status message widgets also provide update accordingly and provide information on the stage of the game.

Objective 3 is fulfilled.

#### 5.1.4 Objective 4

Game settings and config should be customisable. Refer to Figures A.2 and A.3 and A.5.

In the settings screen, the user can change settings to toggle the volume, fullscreen, board colours, particles and shaders. All settings update correctly, and are saved to a local JSON file when the settings screen is exited. The game configurations are changed in the config screen. Here, users can change the game's starting colour, configure timer settings, change to the CPU player and specify the difficulty, select a board preset, or create a custom board layout through the editor screen. The editor screen comes with all the basic operations of placing pieces, rotating pieces, and erasing and dragging them.

Objective 4 is fulfilled.

#### 5.1.5 Objective 5

Game UI should improve player experience. Refer to Figure A.4.

When a player holds down on a piece, the piece can be moved through drag-and-drop. Indicators are also present to show adjacent squares that the piece can be moved to. Audio cues also improve the immersiveness of the game, as well as the laser visuals and particle effects.

Objective 5 is fulfilled. An ambitious plan for greater visuals when the game ends was planned, with a zooming camera and greyscale fade, but was not implemented due to time constraints.

#### 5.1.6 Objective 6

GUI design should be functional and display concise information. Refer to Figure A.1.

I have created most of the custom pixel art and icon graphics to improve the game's visual and attempt to make it more engaging. Most screens also contain both the main menu button and help button. Clicking the menu button switches to the main menu screen, and clicking the help button shows an overlay displaying the functionality of drawn widgets. Shaders are also used to improve the game's visuals, such as by giving the laser a light-emitting effect, but is also used to draw all the backgrounds. The program also resizes seamlessly.

Objective 6 is fulfilled. Some fragment shaders are computationally expensive, and may therefore affect the performance. I have attempted to resolve this by adding the option to disable shaders, however, a better solution would be to optimise the expensive shader code. Working on Windows, I am unable to bundle the program to MacOS or Linux. Although this was not a priority, having access other OS testing environments would've widened access to the game.

## 5.2 Client Feedback

The following interview was conducted after Mr Myslov received and played through the program.

**Q:** Did the final product meet your expectations?

**A:** Yes, it certainly did. The game plays perfectly, I could move, rotate all the pieces as expected, and the laser works well. All the features I requested for were there, and the visuals were also more than I expected and welcome.

---

**Q:** What improvements could be made to the program?

**A:** I found that it was difficult for my laptop to run the program without sacrificing some of the visuals. I also found your FEN string implementation confusing, as there was no guidance on what the characters mean. The text box could also be improved, as it was slightly tedious erasing all the characters manually.

---

**Q:** What other aspects would you like to have been added?

**A:** Being able to somehow easily obtain a digital file for a game would be useful, so I could share it with my students easily. Having a multi-player option instead of people surrounding one laptop would also be a great addition.

My client was satisfied with my final program. His perspective revealed some aspects of the program that I could've better considered and improved upon.

I should've improved upon the intuitiveness of the game by adding more tutorials, to teach new users about the rules or the game, and about my custom FEN string format. Although a tutorial screen is already available in the game screen, an interactive one would perhaps serve the purpose better.

To improve upon the text box and intuitiveness of the widgets, I could implement shortcuts for **Ctrl-A**, or a drag-to-select system.

My client also mentioned a multi-player option, this could be achieved within Python through a peer-to-peer system. This would require a rewrite of the MVC model and how moves are processed, but is a future addition well worth considering.

Regardless, given the limited time frame, I think we are both pleased with the final result.

### 5.3 Conclusion

The final product fulfilled all proposed objectives and more, and has been approved by my client.

Throughout 6 months of development, I felt that I improved on many aspects needed to be a proficient developer. Having the opportunity to create bespoke software for a client was insightful: I learnt about the importance of conducting research and interviews, of obtaining feedback, but also of good communication between client and provider. In terms of coding, working on a large project help improve file management, documentation, and keeping robust backups through Git. It also gave me the opportunity to learn GLSL for my shaders, and LaTeX for my documentation. Having a greater familiarity with Pygame, I also plan to rework my widget system as a library, to be shared for the benefit of all future Pygame users.

Overall, this project was a success.

# Bibliography

- [1] C. Wiki, “Chessprogramming wiki.” <https://www.chessprogramming.org>.
- [2] OfficialGameRules, “Official khet rules.” <https://www.officialgamerules.org/khet>.
- [3] F.-P. Lin, “Good logging practice in python.” <https://fangpenlin.com/posts/2012/08/26/good-logging-practice-in-python/>.
- [4] M. Vanthoor, “Rustic chess.” <https://rustic-chess.org>.

## Assets

- <https://aspecsgaming.itch.io/pixel-art-cursors?download>
- <https://mounirtohami.itch.io/pixel-art-gui-elements>
- <https://leo-red.itch.io/lucid-icon-pack>
- <https://jdsherb1t.itch.io/ultimate-ui-sfx-pack>
- <https://jdwasabi.itch.io/8-bit-16-bit-sound-effects-pack>
- <https://shapeforms.itch.io/shapeforms-audio-free-sfx>
- <https://heltonyan.itch.io/retro-mecha-sfx>
- <https://pixabay.com/sound-effects/game-over-sounds-1-14860/>
- <https://pixabay.com/sound-effects/8-bit-moonlight-sonata-music-loop-20436/>
- <https://pixabay.com/sound-effects/dramatic-synth-echo-43970/>
- <https://jhawk-studios.itch.io/drift-music-pack>
- <https://pixabay.com/music/synthwave-retro-music-260739/>
- <https://pixabay.com/music/upbeat-retro-gaming-271301/>
- <https://pixabay.com/music/video-games-chiptune-medium-boss-218095/>
- <https://www.zapsplat.com/music/game-sound-warm-chime-soft-mallet-start-up-tone/>

## Appendix A

### Screenshots

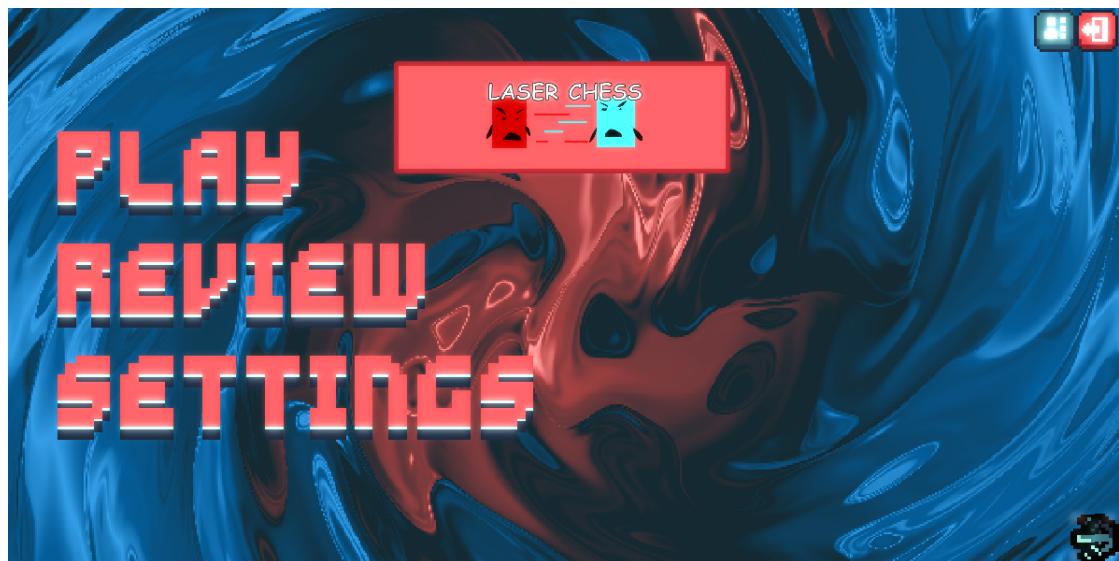


Figure A.1: Main menu screen

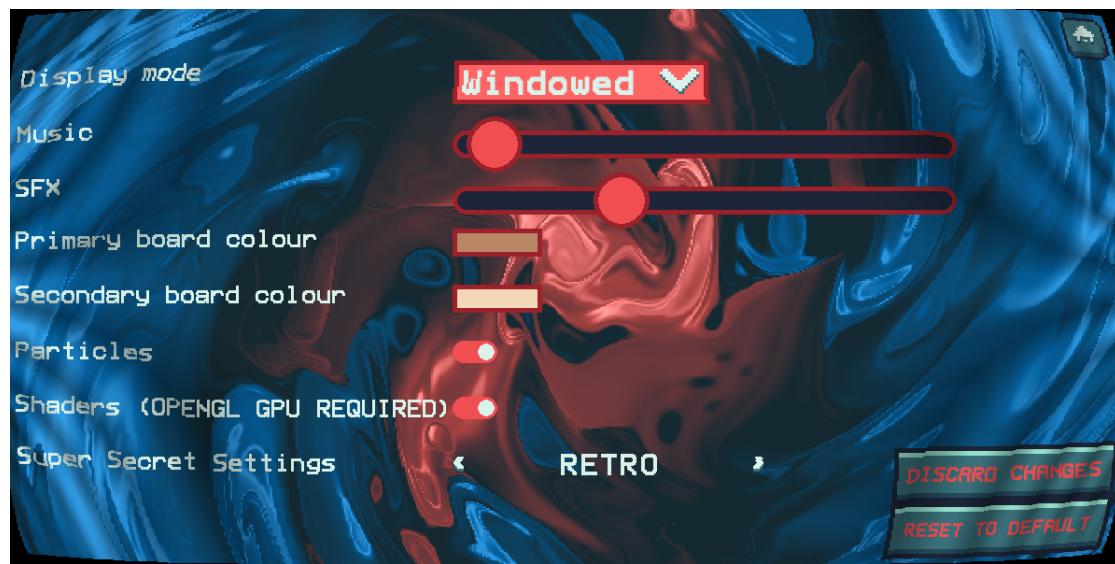


Figure A.2: Settings screen

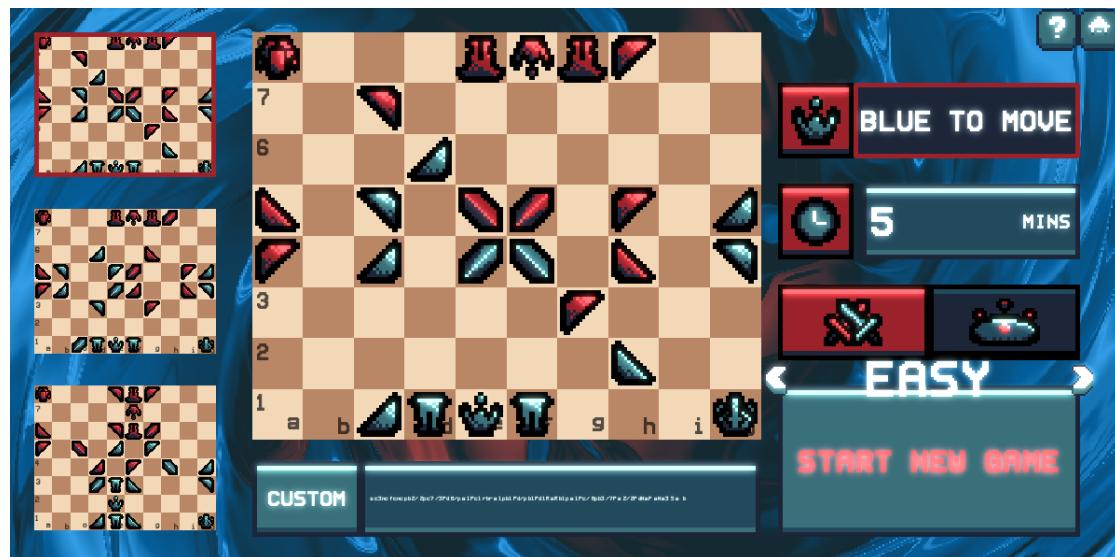


Figure A.3: Config screen

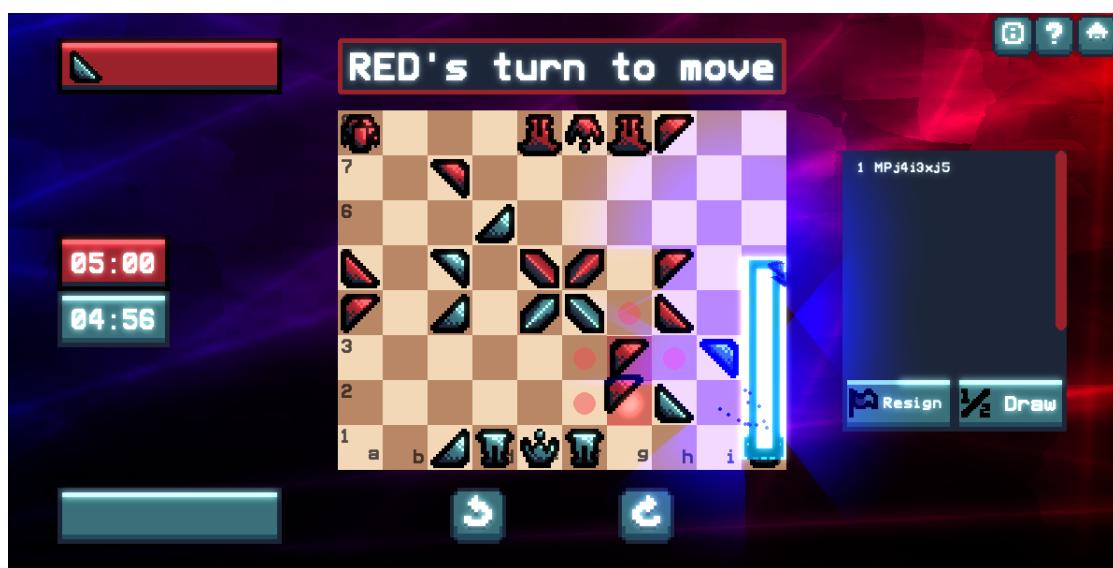


Figure A.4: Game screen



Figure A.5: Editor screen

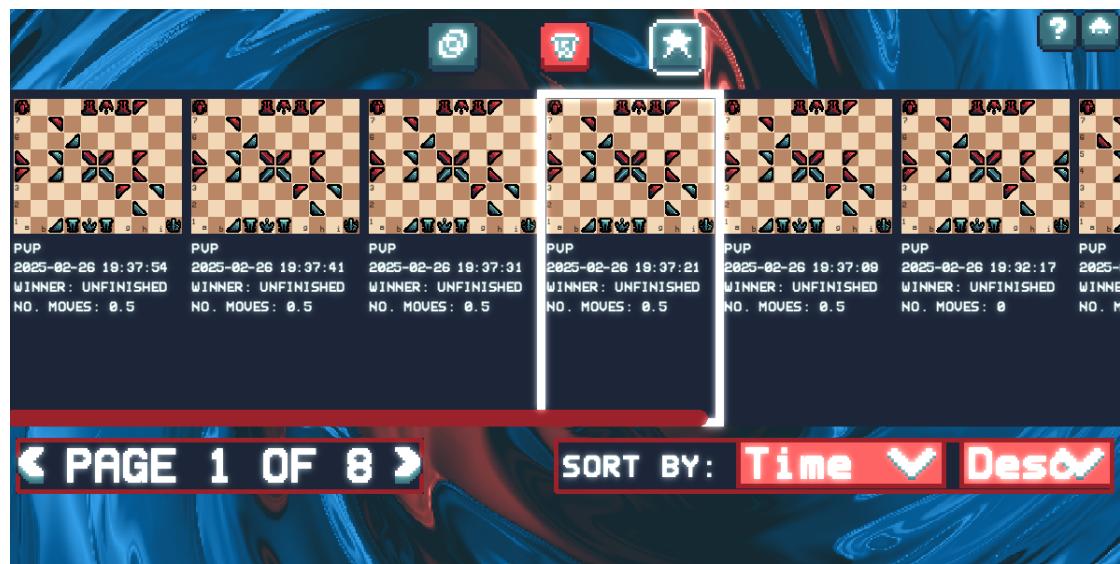


Figure A.6: Browser screen



Figure A.7: Review screen

# Appendix B

## Source Code

### B.1 data

#### B.1.1 control.py

```
1 import pygame
2 from data.components.widget_group import WidgetGroup
3 from data.managers.logs import initialise_logger
4 from data.managers.cursor import CursorManager
5 from data.managers.animation import animation
6 from data.utils.assets import DEFAULT_FONT
7 from data.managers.window import window
8 from data.managers.audio import audio
9 from data.managers.theme import theme
10
11 logger = initialise_logger(__file__)
12
13 FPS = 60
14 SHOW_FPS = False
15 start_ticks = pygame.time.get_ticks()
16
17 class Control:
18     def __init__(self):
19         self.done = False
20         self._clock = pygame.time.Clock()
21
22     def setup_states(self, state_dict, start_state):
23         self.state_dict = state_dict
24         self.state_name = start_state
25
26         self.state = self.state_dict[self.state_name]
27         self.state.startup()
28
29     def flip_state(self):
30         self.state.done = False
31         persist = self.state.cleanup()
32
33         previous, self.state_name = self.state_name, self.state.next
34
35         self.state = self.state_dict[self.state_name]
36         self.state.previous = previous
37         self.state.startup(persist)
38
```

```
39     def update(self):
40         if self.state.quit:
41             self.done = True
42         elif self.state.done:
43             self.flip_state()
44
45         self._clock.tick(FPS)
46         animation.set_delta_time()
47
48         self.state.update()
49
50         if SHOW_FPS:
51             self.draw_fps()
52
53         window.update()
54
55     def main_game_loop(self):
56         while not self.done:
57             self.event_loop()
58             self.update()
59
60     def update_window(self, resize=False):
61         if resize:
62             self.update_native_window_size()
63             window.handle_resize()
64             self.state.handle_resize()
65
66         self.update()
67
68     def draw_fps(self):
69         fps = str(int(self._clock.get_fps()))
70         DEFAULT_FONT.strength = 0.1
71         DEFAULT_FONT.render_to(window.screen, (0, 0), fps, fgcolor=theme['textError'], size=15)
72
73     def update_native_window_size(self):
74         x, y = window.size
75
76         max_window_x = 100000
77         max_window_y = x / 1.4
78         min_window_x = 400
79         min_window_y = min_window_x / 1.4
80
81         if x / y < 1.4:
82             min_window_x = x
83
84         min_window_size = (min_window_x, min_window_y)
85         max_window_size = (max_window_x, max_window_y)
86         window.minimum_size = min_window_size
87         window.maximum_size = max_window_size
88
89     def event_loop(self):
90         for event in pygame.event.get():
91             if event.type == pygame.QUIT:
92                 self.done = True
93
94             if event.type == pygame.MOUSEBUTTONDOWN and event.button != 1: # ONLY
95                 PROCESS LEFT CLICKS
96                 return
97
98             self.state.get_event(event)
```

```

99  class _State:
100     def __init__(self):
101         self.next = None
102         self.previous = None
103         self.done = False
104         self.quit = False
105         self.persist = {}
106
107         self._cursor = CursorManager()
108         self._widget_group = None
109
110     def startup(self, widgets=None, music=None):
111         if widgets:
112             self._widget_group = WidgetGroup(widgets)
113             self._widget_group.handle_resize(window.size)
114
115         if music:
116             audio.play_music(music)
117
118         logger.info(f'starting {self.__class__._name_.lower()}')
119
120     def cleanup(self):
121         logger.info(f'cleaning {self.__class__._name_.lower()}')
122
123     def draw(self):
124         raise NotImplementedError
125
126     def get_event(self, event):
127         raise NotImplementedError
128
129     def handle_resize(self):
130         self._widget_group.handle_resize(window.size)
131
132     def update(self, **kwargs):
133         self.draw()

```

### B.1.2 loading\_screen.py

See Section 3.3.2.

### B.1.3 main.py

See Section 3.3.1.

### B.1.4 setup.py

```

1 import pygame
2
3 pygame.mixer.init()
4 pygame.init()
5
6 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MAJOR_VERSION, 3)
7 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MINOR_VERSION, 3)
8 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_PROFILE_MASK, pygame.
    GL_CONTEXT_PROFILE_CORE)
9 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_FORWARD_COMPATIBLE_FLAG, True)

```

### B.1.5 windows\_setup.py

```

1 import win32gui
2 import win32con
3 import os
4 import ctypes
5 import sys
6
7 def wndProc(oldWndProc, draw_callback, hWnd, message, wParam, lParam):
8     if message == win32con.WM_SIZING or message == win32con.WM_TIMER: # Don't know
9         what WM_TIMER does
10        draw_callback(resize=True)
11        win32gui.RedrawWindow(hWnd, None, None, win32con.RDW_INVALIDATE | win32con.
12 .RDW_ERASE)
13    elif message == win32con.WM_MOVE:
14        draw_callback(resize=False)
15
16    return win32gui.CallWindowProc(oldWndProc, hWnd, message, wParam, lParam)
17
18 def set_win_resize_func(resize_function):
19     oldWndProc = win32gui.SetWindowLong(win32gui.GetForegroundWindow(), win32con.
20 GWL_WNDPROC, lambda *args: wndProc(oldWndProc, resize_function, *args))
21
22 user32 = ctypes.windll.user32
23 user32.SetProcessDPIAware() # To deal with Windows High Text Size / Low Display
   Resolution Settings
24
25 if os.name != 'nt' or sys.getwindowsversion()[0] < 6:
26     raise NotImplementedError("Incompatible OS!")

```

## B.2 data\app\_data

### B.2.1 default\_settings.json

```

1 {
2     "primaryBoardColour": "0xB98766",
3     "secondaryBoardColour": "0xF3D8B8",
4     "laserColourBlue": "0x0000ff",
5     "laserColourRed": "0xffff0000",
6     "displayMode": "windowed",
7     "musicVolume": 0.5,
8     "sfxVolume": 0.5,
9     "particles": true,
10    "opengl": true,
11    "shader": "default"
12 }

```

### B.2.2 logs\_config.json

```

1 {
2     "version": 1,
3     "disable_existing_loggers": false,
4     "formatters": {
5         "simple": {
6             "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
7             "datefmt": "%Y-%m-%d %H:%M:%S"
8         }
9     },
10    "handlers": {
11        "console": {
12            "class": "logging.StreamHandler",

```

```

14         "formatter": "simple",
15         "stream": "ext://sys.stdout"
16     },
17 },
18
19 "root": {
20     "level": "INFO",
21     "handlers": ["console"],
22     "propagate": false
23 }
24

```

### B.2.3 logs\_config\_prod.json

```

1  {
2      "version": 1,
3      "disable_existing_loggers": false,
4      "formatters": {
5          "simple": {
6              "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
7          }
8      },
9
10     "handlers": {
11         "console": {
12             "class": "logging.StreamHandler",
13             "level": "DEBUG",
14             "formatter": "simple",
15             "stream": "ext://sys.stdout"
16         },
17
18         "info_file_handler": {
19             "class": "logging.handlers.RotatingFileHandler",
20             "level": "INFO",
21             "formatter": "simple",
22             "filename": "info.log",
23             "maxBytes": 10485760,
24             "backupCount": 20,
25             "encoding": "utf8"
26         },
27
28         "error_file_handler": {
29             "class": "logging.handlers.RotatingFileHandler",
30             "level": "ERROR",
31             "formatter": "simple",
32             "filename": "errors.log",
33             "maxBytes": 10485760,
34             "backupCount": 20,
35             "encoding": "utf8"
36         }
37     },
38
39     "loggers": {
40         "my_module": {
41             "level": "ERROR",
42             "handlers": ["console"],
43             "propagate": false
44         }
45     },
46
47     "root": {
48         "level": "INFO",

```

```

49         "handlers": ["console", "info_file_handler", "error_file_handler"]
50     }
51 }

```

#### B.2.4 themes.json

```

1  {
2      "colours": {
3          "text": {
4              "primary": "0xdaf2e9",
5              "secondary": "0xf14e52",
6              "error": "0xf14e52"
7          },
8          "fill": {
9              "primary": "0x1c2638",
10             "secondary": "0xf14e52",
11             "tertiary": "0xdaf2e9",
12             "error": "0x9b222b"
13         },
14         "border": {
15             "primary": "0x9b222b",
16             "secondary": ""
17         }
18     },
19     "dimensions": {
20         "borderRadius": 3,
21         "borderWidth": 5,
22         "margin": 10
23     }
24 }

```

#### B.2.5 user\_settings.json

```

1  {
2      "primaryBoardColour": "0xB98766",
3      "secondaryBoardColour": "0xF3D8B8",
4      "laserColourBlue": "0x0000ff",
5      "laserColourRed": "0xffff00",
6      "displayMode": "windowed",
7      "musicVolume": 0.085,
8      "sfxVolume": 0.336,
9      "particles": true,
10     "opengl": true,
11     "shader": "default"
12 }

```

### B.3 data\components

#### B.3.1 circular\_linked\_list.py

See Section 3.4.3.

#### B.3.2 cursor.py

```

1 import pygame
2
3 class Cursor(pygame.sprite.Sprite):
4     def __init__(self):
5         super().__init__()

```

```

6         self.image = pygame.Surface((1, 1))
7         self.image.fill((255, 0, 0))
8         self.rect = self.image.get_rect()
9
10        # def update(self):
11        #     self.rect.center = pygame.mouse.get_pos()
12
13    def get_sprite_collision(self, mouse_pos, square_group):
14        self.rect.center = mouse_pos
15        sprite = pygame.sprite.spritecollideany(self, square_group)
16
17    return sprite

```

### B.3.3 custom\_event.py

See Section 3.4.4.

### B.3.4 game\_entry.py

```

1  from data.states.game.components.move import Move
2  from data.utils.enums import Colour
3
4  class GameEntry:
5      def __init__(self, game_states, final_fen_string):
6          self._game_states = game_states
7          self._final_fen_string = final_fen_string
8
9      def __str__(self):
10         return f'''
11 <GameEntry> :
12     CPU_ENABLED: {self._game_states['CPU_ENABLED']},
13     CPU_DEPTH: {self._game_states['CPU_DEPTH']},
14     WINNER: {self._game_states['WINNER']},
15     TIME_ENABLED: {self._game_states['TIME_ENABLED']},
16     TIME: {self._game_states['TIME']},
17     NUMBER_OF_PLY: {len(self._game_states['MOVES'])},
18     MOVES: {self.convert_moves(self._game_states['MOVES'])}
19     FINAL_FEN_STRING: {self._final_fen_string}
20     START_FEN_STRING: {self._game_states['START_FEN_STRING']}
21 </GameEntry>
22         '''
23
24     def convert_to_row(self):
25         return (self._game_states['CPU_ENABLED'], self._game_states['CPU_DEPTH'],
26         self._game_states['WINNER'], self._game_states['TIME_ENABLED'], self.
27         _game_states['TIME'], len(self._game_states['MOVES']), self.convert_moves(self.
28         _game_states['MOVES']), self._game_states['START_FEN_STRING'], self.
29         _final_fen_string)
30
31     def convert_moves(self, moves):
32         return '|'.join([
33             f'{round(move["time"][Colour.BLUE], 4)};{round(move["time"][Colour.RED],
34             4)};{move["move"]}'
35             for move in moves
36         ])
37
38     @staticmethod
39     def parse_moves(move_str):
40         moves = move_str.split('|')
41         return [

```

```

37
38         'blue_time': move.split(';')[0],
39         'red_time': move.split(';')[1],
40         'move': Move.instance_from_notation(move.split(';')[2]),
41         'unparsed_move': move.split(';')[2],
42     } for move in moves if move != ''
43 ]

```

### B.3.5 widget\_group.py

```

1 import pygame
2 from data.managers.window import window
3
4 class WidgetGroup(pygame.sprite.Group):
5     def __init__(self, widget_dict):
6         super().__init__()
7
8         for value in widget_dict.values():
9             if isinstance(value, list):
10                 for widget in value:
11                     self.add(widget)
12             elif isinstance(value, dict):
13                 for widget in value.values():
14                     self.add(widget)
15             else:
16                 self.add(value)
17
18     def handle_resize(self, new_surface_size):
19         for sprite in self.sprites():
20             sprite.set_surface_size(new_surface_size)
21             sprite.set_image()
22             sprite.set_geometry()
23
24     def process_event(self, event):
25         for sprite in self.sprites():
26             widget_event = sprite.process_event(event)
27
28             if widget_event:
29                 return widget_event
30
31     return None
32
33     def draw(self):
34         sprites = self.sprites()
35         for spr in sprites:
36             surface = spr._surface or window.screen
37             self.spritedict[spr] = surface.blit(spr.image, spr.rect)
38         self.lostsprites = []
39         dirty = self.lostsprites
40
41     return dirty
42
43     def on_widget(self, mouse_pos):
44         test_sprite = pygame.sprite.Sprite()
45         test_sprite.rect = pygame.Rect(*mouse_pos, 1, 1)
46         return pygame.sprite.spritecollideany(test_sprite, self)

```

## B.4 data\database

### B.5 data\database\migrations

#### B.5.1 add\_created\_dt\_column27112024.py

```
1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     connection = sqlite3.connect(database_path)
8     cursor = connection.cursor()
9
10    cursor.execute('''
11        ALTER TABLE games ADD COLUMN created_dt TIMESTAMP NOT NULL
12    ''')
13
14    connection.commit()
15    connection.close()
16
17 def downgrade():
18     connection = sqlite3.connect(database_path)
19     cursor = connection.cursor()
20
21    cursor.execute('''
22        ALTER TABLE games DROP COLUMN created_dt
23    ''')
24
25    connection.commit()
26    connection.close()
27
28 upgrade()
29 # downgrade()
```

#### B.5.2 add\_fen\_string\_column\_22112024.py

```
1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     connection = sqlite3.connect(database_path)
8     cursor = connection.cursor()
9
10    cursor.execute('''
11        ALTER TABLE games ADD COLUMN fen_string TEXT NOT NULL
12    ''')
13
14    connection.commit()
15    connection.close()
16
17 def downgrade():
18     connection = sqlite3.connect(database_path)
19     cursor = connection.cursor()
20
21    cursor.execute(''')
```

```

22         ALTER TABLE games DROP COLUMN fen_string
23     """
24
25     connection.commit()
26     connection.close()
27
28 upgrade()

```

**B.5.3 add\_start\_fen\_string\_column\_23122024.py**

```

1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     connection = sqlite3.connect(database_path)
8     cursor = connection.cursor()
9
10    cursor.execute("""
11        ALTER TABLE games ADD COLUMN start_fen_string TEXT NOT NULL
12    """)
13
14    connection.commit()
15    connection.close()
16
17 def downgrade():
18     connection = sqlite3.connect(database_path)
19     cursor = connection.cursor()
20
21    cursor.execute("""
22        ALTER TABLE games DROP COLUMN start_fen_string
23    """)
24
25    connection.commit()
26    connection.close()
27
28 upgrade()
29 # downgrade()

```

**B.5.4 change\_fen\_string\_column\_name\_23122024.py**

See Section 3.8.1.

**B.5.5 create\_games\_table\_19112024.py**

See Section 3.8.1.

**B.6 data\helpers****B.6.1 asset\_helpers.py**

See Section 3.3.3.

**B.6.2 bitboard\_helpers.py**

```

1 from data.managers.logs import initialise_logger
2 from data.utils.constants import EMPTY_BB
3 from data.utils.enums import Rank, File
4
5 logger = initialise_logger(__name__)
6
7 def print_bitboard(bitboard):
8     if (bitboard >= (2 ** 80)):
9         raise ValueError('Invalid bitboard: too many bits')
10
11    characters = ''
12    for rank in reversed(Rank):
13
14        for file in File:
15            mask = 1 << (rank * 10 + file)
16            if (bitboard & mask) != 0:
17                characters += '1'
18            else:
19                characters += '.'
20
21    characters += '\n\n'
22
23    logger.info('\n' + characters + '\n')
24
25 def is_occupied(bitboard, target_bitboard):
26     return (target_bitboard & bitboard) != EMPTY_BB
27
28 def clear_square(bitboard, target_bitboard):
29     return (~target_bitboard & bitboard)
30
31 def set_square(bitboard, target_bitboard):
32     return (target_bitboard | bitboard)
33
34 def index_to_bitboard(index):
35     return (1 << index)
36
37 def coords_to_bitboard(coords):
38     index = coords[1] * 10 + coords[0]
39     return index_to_bitboard(index)
40
41 def bitboard_to_notation(bitboard):
42     index = bitboard_to_index(bitboard)
43     x = index // 10
44     y = index % 10
45
46     return chr(y + 97) + str(x + 1)
47
48 def notation_to_bitboard(notation):
49     index = (int(notation[1]) - 1) * 10 + int(ord(notation[0])) - 97
50
51     return index_to_bitboard(index)
52
53 def bitboard_to_index(bitboard):
54     return bitboard.bit_length() - 1
55
56 def bitboard_to_coords(bitboard):
57     list_position = bitboard_to_index(bitboard)
58     x = list_position % 10
59     y = list_position // 10
60
61     return x, y
62

```

```

63 def bitboard_to_coords_list(bitboard):
64     list_positions = []
65
66     for square in occupied_squares(bitboard):
67         list_positions.append(bitboard_to_coords(square))
68
69     return list_positions
70
71 def occupied_squares(bitboard):
72     while bitboard:
73         lsb_square = bitboard & -bitboard
74         bitboard = bitboard ^ lsb_square
75
76         yield lsb_square
77
78 def pop_count(bitboard):
79     count = 0
80     while bitboard:
81         count += 1
82         lsb_square = bitboard & -bitboard
83         bitboard = bitboard ^ lsb_square
84
85     return count
86
87 # def pop_count(bitboard):
88 #     count = 0
89 #     while bitboard:
90 #         count += 1
91 #         bitboard &= bitboard - 1
92 #
93 #     return count
94
95 def loop_all_squares():
96     for i in range(80):
97         yield 1 << i
98
99 # Solar
100 def get_LSB_value(bitboard: int):
101     return bitboard & -bitboard
102
103 def pop_count_2(bitboard):
104     count = 0
105     while bitboard > 0:
106         lsb_value = get_LSB_value(bitboard)
107         count += 1
108         bitboard ^= lsb_value
109
110     return count

```

### B.6.3 board\_helpers.py

```

1 import pygame
2 from data.helpers.data_helpers import get_user_settings
3 from data.utils.assets import DEFAULT_FONT
4
5 user_settings = get_user_settings()
6
7 def create_board(board_size, primary_colour, secondary_colour, font=DEFAULT_FONT):
8     square_size = board_size[0] / 10
9     board_surface = pygame.Surface(board_size)
10
11    for i in range(80):

```

```

12         x = i % 10
13         y = i // 10
14
15         if (x + y) % 2 == 0:
16             square_colour = primary_colour
17         else:
18             square_colour = secondary_colour
19
20         square_x = x * square_size
21         square_y = y * square_size
22
23         pygame.draw.rect(board_surface, square_colour, (square_x, square_y,
24 square_size + 1, square_size + 1)) # +1 to fill in black lines
25
26         if y == 7:
27             text_position = (square_x + square_size * 0.7, square_y + square_size
28 * 0.55)
29             text_size = square_size / 3
30             font.render_to(board_surface, text_position, str(chr(x + 1 + 96)),
31 fgcolor=(10, 10, 10, 175), size=text_size)
32             if x == 0:
33                 text_position = (square_x + square_size * 0.1, square_y + square_size
34 * 0.1)
35                 text_size = square_size / 3
36                 font.render_to(board_surface, text_position, str(7-y + 1), fgcolor
37 =(10, 10, 10, 175), size=text_size)
38
39             return board_surface
40
41     def create_square_overlay(square_size, colour):
42         overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
43         overlay.fill(colour)
44
45         return overlay
46
47     def create_circle_overlay(square_size, colour):
48         overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
49         pygame.draw.circle(overlay, colour, (square_size / 2, square_size / 2),
50 square_size / 4)
51
52         return overlay
53
54     def coords_to_screen_pos(coords, board_position, square_size):
55         x = board_position[0] + (coords[0] * square_size)
56         y = board_position[1] + ((7 - coords[1]) * square_size)
57
58         return (x, y)
59
60     def screen_pos_to_coords(mouse_position, board_position, board_size):
61         if (board_position[0] <= mouse_position[0] <= board_position[0] + board_size
62 [0]) and (board_position[1] <= mouse_position[1] <= board_position[1] +
63 board_size[1]):
64             x = (mouse_position[0] - board_position[0]) // (board_size[0] / 10)
65             y = (board_size[1] - (mouse_position[1] - board_position[1])) // (
66 board_size[0] / 10)
67
68             return (int(x), int(y))
69
70         return None

```

#### B.6.4 browser\_helpers.py

```

1 from data.utils.enums import Miscellaneous, Colour
2
3 def get_winner_string(winner):
4     if winner is None:
5         return 'UNFINISHED'
6     elif winner == Miscellaneous.DRAW:
7         return 'DRAW'
8     else:
9         return Colour(winner).name

```

### B.6.5 database\_helpers.py

See Section 3.8.2.

### B.6.6 data\_helpers.py

See Section 3.3.3.

### B.6.7 font\_helpers.py

```

1 def height_to_font_size(font, target_height):
2     test_size = 1
3     while True:
4         glyph_metrics = font.get_metrics('j', size=test_size)
5         descender = font.get_sized_descender(test_size)
6         test_height = abs(glyph_metrics[0][3] - glyph_metrics[0][2]) - descender
7         if test_height > target_height:
8             return test_size - 1
9
10        test_size += 1
11
12 def width_to_font_size(font, target_width):
13     test_size = 1
14     while True:
15         glyph_metrics = font.get_metrics(' ', size=test_size)
16
17         if (glyph_metrics[0][4] * 8) > target_width:
18             return (test_size - 1)
19
20        test_size += 1
21
22 def text_width_to_font_size(text, font, target_width):
23     test_size = 1
24     if len(text) == 0:
25         # print('(text_width_to_font_size) Text must have length greater than 1!')
26         text = " "
27
28     while True:
29         text_rect = font.get_rect(text, size=test_size)
30
31         if text_rect.width > target_width:
32             return (test_size - 1)
33
34         test_size += 1
35
36 def text_height_to_font_size(text, font, target_height):
37     test_size = 1
38
39     if ('(' in text) or (')' in text):

```

```

40     text = text.replace('(', 'j') # Pygame freetype thinks '(' or ')' is
41     taller for some reason
42     text = text.replace(')', 'j')
43
44     if len(text) == 0:
45         # print('(text_height_to_font_size) Text must have length greater than
46         1!')
47         text = "j"
48
49     while True:
50         text_rect = font.get_rect(text, size=test_size)
51
52         if text_rect.height > target_height:
53             return (test_size - 1)
54
55     def get_font_height(font, font_size):
56         glyph_metrics = font.get_metrics('j', size=font_size)
57         descender = font.get_sized_descender(font_size)
58         return abs(glyph_metrics[0][3] - glyph_metrics[0][2]) - descender

```

### B.6.8 input\_helpers.py

```

1 from data.utils.enums import MoveType, Rotation
2
3 def parse_move_type(move_type):
4     if move_type.isalpha() is False:
5         raise ValueError('Invalid move type - move type must be a string!')
6     if move_type.lower() not in MoveType:
7         raise ValueError('Invalid move - type - move type must be m or r!')
8
9     return MoveType(move_type.lower())
10
11 def parse_notation(notation):
12     if (notation[0].isalpha() is False) or (notation[1].isnumeric() is False):
13         raise ValueError('Invalid notation - invalid notation input types!')
14     if not (97 <= ord(notation[0]) <= 106):
15         raise ValueError('Invalid notation - file is out of range!')
16     elif not (0 <= int(notation[1]) <= 10):
17         raise ValueError('Invalid notation - rank is out of range!')
18
19     return notation
20
21 def parse_rotation(rotation):
22     if rotation == '':
23         return None
24     if rotation.isalpha() is False:
25         raise ValueError('Invalid rotation - rotation must be a string!')
26     if rotation.lower() not in Rotation:
27         raise ValueError('Invalid rotation - rotation is invalid!')
28
29     return Rotation(rotation.lower())

```

### B.6.9 load\_helpers.py

```

1 import pygame
2 import pygame.freetype
3 from pathlib import Path
4 from data.helpers.asset_helpers import gif_to_frames, pil_image_to_surface
5

```

```

6 def convert_gfx_alpha(image, colorkey=(0, 0, 0)):
7     # if image.get_alpha():
8         # return image.convert_alpha()
9     # else:
10        # image = image.convert_alpha()
11        # image.set_colorkey(colorkey)
12
13    # return image
14
15 def load_gfx(path, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
16     file_path = Path(path)
17     name, extension = file_path.stem, file_path.suffix
18
19     if extension.lower() in accept:
20         if extension.lower() == '.gif':
21             frames_list = []
22
23             for frame in gif_to_frames(path):
24                 image_surface = pil_image_to_surface(frame)
25                 frames_list.append(image_surface)
26
27             return frames_list
28
29     if extension.lower() == '.svg':
30         low_quality_image = pygame.image.load_sized_svg(path, (200, 200))
31         image = pygame.image.load(path)
32         image = convert_gfx_alpha(image, colorkey)
33
34     return [image, low_quality_image]
35
36     else:
37         image = pygame.image.load(path)
38         return convert_gfx_alpha(image, colorkey)
39
40 def load_all_gfx(directory, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
41     graphics = {}
42
43     for file in Path(directory).rglob('*'):
44         name, extension = file.stem, file.suffix
45         path = Path(directory / file)
46
47         if extension.lower() in accept and 'old' not in name:
48             if name == 'piece_spritesheet':
49                 data = load_spritesheet(
50                     path,
51                     (16, 16),
52                     ['pyramid_1', 'scarab_1', 'anubis_1', 'pharaoh_1', 'sphinx_1',
53                     'pyramid_0', 'scarab_0', 'anubis_0', 'pharaoh_0', 'sphinx_0'],
54                     ['_a', '_b', '_c', '_d'])
55
56                 graphics = graphics | data
57                 continue
58
59                 data = load_gfx(path, colorkey, accept)
60
61                 if isinstance(data, list):
62                     graphics[name] = data[0]
63                     graphics[f'{name}_lq'] = data[1]
64                 else:
65                     graphics[name] = data

```

```

66     return graphics
67
68 def load_spritesheet(path, sprite_size, col_names, row_names):
69     spritesheet = load_gfx(path)
70     col_count = int(spritesheet.width / sprite_size[0])
71     row_count = int(spritesheet.height / sprite_size[1])
72
73     sprite_dict = {}
74
75     for column in range(col_count):
76         for row in range(row_count):
77             surface = pygame.Surface(sprite_size, pygame.SRCALPHA)
78             name = col_names[column] + row_names[row]
79
80             surface.blit(spritesheet, (0, 0), (column * sprite_size[0], row *
81                                         sprite_size[1], *sprite_size))
82             sprite_dict[name] = surface
83
84     return sprite_dict
85
86 def load_all_fonts(directory, accept=( ".ttf", ".otf")):
87     fonts = {}
88
89     for file in Path(directory).rglob('*'):
90         name, extension = file.stem, file.suffix
91         path = Path(directory / file)
92
93         if extension.lower() in accept:
94             font = pygame.freetype.Font(path)
95             fonts[name] = font
96
97     return fonts
98
99 def load_all_sfx(directory, accept=( ".mp3", ".wav", ".ogg")):
100    sound_effects = {}
101
102    for file in Path(directory).rglob('*'):
103        name, extension = file.stem, file.suffix
104        path = Path(directory / file)
105
106        if extension.lower() in accept and 'old' not in name:
107            sound_effects[name] = load_sfx(path)
108
109    return sound_effects
110
111 def load_sfx(path, accept=( ".mp3", ".wav", ".ogg")):
112     file_path = Path(path)
113     name, extension = file_path.stem, file_path.suffix
114
115     if extension.lower() in accept:
116         sfx = pygame.mixer.Sound(path)
117         return sfx
118
119 def load_all_music(directory, accept=( ".mp3", ".wav", ".ogg")):
120     music_paths = {}
121
122     for file in Path(directory).rglob('*'):
123         name, extension = file.stem, file.suffix
124         path = Path(directory / file)
125
126         if extension.lower() in accept:
127             music_paths[name] = path
128

```

```
127     return music_paths
```

### B.6.10 widget\_helpers.py

See Section 3.3.3.

## B.7 data\managers

### B.7.1 animation.py

```
1 import pygame
2 from data.helpers.asset_helpers import scale_and_cache
3
4 FPS = 60
5
6 class AnimationManager:
7     def __init__(self):
8         self._current_ms = 0
9         self._timers = []
10
11    def set_delta_time(self):
12        self._current_ms = pygame.time.get_ticks()
13
14        for timer in self._timers:
15            start_ms, target_ms, callback = timer
16            if self._current_ms - start_ms >= target_ms:
17                callback()
18                self._timers.remove(timer)
19
20    def calculate_frame_index(self, start_index, end_index, fps):
21        ms_per_frame = int(1000 / fps)
22        return start_index + ((self._current_ms // ms_per_frame) % (end_index -
start_index))
23
24    def draw_animation(self, screen, animation, position, size, fps=8):
25        frame_index = self.calculate_frame_index(0, len(animation), fps)
26        scaled_animation = scale_and_cache(animation[frame_index], size)
27        screen.blit(scaled_animation, position)
28
29    def draw_image(self, screen, image, position, size):
30        scaled_background = scale_and_cache(image, size)
31        screen.blit(scaled_background, position)
32
33    def set_timer(self, target_ms, callback):
34        self._timers.append((self._current_ms, target_ms, callback))
35
36 animation = AnimationManager()
```

### B.7.2 audio.py

```
1 import pygame
2 from data.helpers.data_helpers import get_user_settings
3 from data.managers.logs import initialise_logger
4
5 logger = initialise_logger(__name__)
6 user_settings = get_user_settings()
7
8 class AudioManager:
9     def __init__(self, num_channels=16):
```

```
10     pygame.mixer.set_num_channels(num_channels)
11
12     self._music_volume = user_settings['musicVolume']
13     self._sfx_volume = user_settings['sfxVolume']
14
15     self._current_song = None
16     self._current_channels = []
17
18     def set_sfx_volume(self, volume):
19         self._sfx_volume = volume
20
21         for channel in self._current_channels:
22             channel.set_volume(self._sfx_volume)
23
24     def set_music_volume(self, volume):
25         self._music_volume = volume
26
27         pygame.mixer.music.set_volume(self._music_volume)
28
29     def pause_sfx(self):
30         pygame.mixer.pause()
31
32     def unpause_sfx(self):
33         pygame.mixer.unpause()
34
35     def stop_sfx(self, fadeout=0):
36         pygame.mixer.fadeout(fadeout)
37
38     def remove_unused_channels(self):
39         unused_channels = []
40         for channel in self._current_channels:
41             if channel.get_busy() is False:
42                 unused_channels.append(channel)
43
44         return unused_channels
45
46     def play_sfx(self, sfx, loop=False):
47         unused_channels = self.remove_unused_channels()
48
49         if len(unused_channels) == 0:
50             channel = pygame.mixer.find_channel()
51         else:
52             channel = unused_channels.pop(0)
53
54         if channel is None:
55             logger.warning('No available channel for SFX')
56             return
57
58         self._current_channels.append(channel)
59         channel.set_volume(self._sfx_volume)
60
61         if loop:
62             channel.play(sfx, loops=-1)
63         else:
64             channel.play(sfx)
65
66     def play_music(self, music_path):
67         if 'menu' in str(music_path) and 'menu' in str(self._current_song):
68             return
69
70         if music_path == self._current_song:
71             return
```

```

72         pygame.mixer.music.stop()
73         pygame.mixer.music.unload()
74         pygame.mixer.music.load(music_path)
75         pygame.mixer.music.set_volume(self._music_volume)
76         pygame.mixer.music.play(loops=-1)
77
78         self._current_song = music_path
79
80
81 audio = AudioManager()

```

### B.7.3 cursor.py

```

1 import pygame
2 from data.utils.enums import CursorMode
3 from data.utils.assets import GRAPHICS
4
5 class CursorManager:
6     def __init__(self):
7         self._mode = CursorMode.ARROW
8         self.set_mode(CursorMode.ARROW)
9
10    def set_mode(self, mode):
11        pygame.mouse.set_visible(True)
12
13        match mode:
14            case CursorMode.ARROW:
15                pygame.mouse.set_cursor((7, 5), pygame.transform.scale(GRAPHICS['arrow'], (32, 32)))
16            case CursorMode.IBEAM:
17                pygame.mouse.set_cursor((15, 5), pygame.transform.scale(GRAPHICS['ibeam'], (32, 32)))
18            case CursorMode.OPENHAND:
19                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['hand_open'], (32, 32)))
20            case CursorMode.CLOSEDHAND:
21                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['hand_closed'], (32, 32)))
22            case CursorMode.NO:
23                pygame.mouse.set_visible(False)
24
25        self._mode = mode
26
27    def get_mode(self):
28        return self._mode
29
30 cursor = CursorManager()

```

### B.7.4 logs.py

```

1 import logging.config
2 from data.helpers.data_helpers import load_json
3 from pathlib import Path
4 import logging
5
6 config_path = (Path(__file__).parent / '../app_data/logs_config.json').resolve()
7 config = load_json(config_path)
8 logging.config.dictConfig(config)
9
10 def initialise_logger(file_path):
11     return logging.getLogger(Path(file_path).name)

```

### B.7.5 shader.py

See Section 3.9.1.

### B.7.6 theme.py

See Section 3.3.4.

### B.7.7 window.py

```

1 import pygame
2 import moderngl
3 from data.utils.constants import ShaderType, SCREEN_SIZE, SHADER_MAP
4 from data.helpers.data_helpers import get_user_settings
5 from data.helpers.asset_helpers import draw_background
6 from data.managers.shader import ShaderManager
7
8 user_settings = get_user_settings()
9 is_opengl = user_settings['opengl']
10 is_fullscreen = user_settings['displayMode'] == 'fullscreen'
11
12 class WindowManager(pygame.Window):
13     def __init__(self, **kwargs):
14         super().__init__(**kwargs)
15         self._native_screen = self.get_surface() # Initialise convert format
16         self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
17
18     if is_opengl:
19         self._ctx = moderngl.create_context()
20         self._shader_manager = ShaderManager(self._ctx, screen_size=self.size)
21
22         self.shader_arguments = {
23             ShaderType.BASE: {},
24             ShaderType.SHAKE: {},
25             ShaderType.BLOOM: {},
26             ShaderType.GRAYSCALE: {},
27             ShaderType.CRT: {},
28             ShaderType.RAYS: {}
29         }
30
31         if (selected_shader := get_user_settings()['shader']) is not None:
32             for shader_type in SHADER_MAP[selected_shader]:
33                 self.set_effect(shader_type)
34
35         else:
36             from data.utils.assets import GRAPHICS
37             self._background_image = GRAPHICS['temp_background']
38
39     def set_effect(self, effect, **kwargs):
40         if is_opengl:
41             self._shader_manager.apply_shader(effect, **kwargs)
42
43     def set_apply_arguments(self, effect, **kwargs):
44         if is_opengl:
45             self.shader_arguments[effect] = kwargs
46
47     def clear_apply_arguments(self, effect):
48         if is_opengl:
49             self.shader_arguments[effect] = {}
50
51     def clear_effect(self, effect):

```

```

51         if is_opengl:
52             self._shader_manager.remove_shader(effect)
53             self.clear_apply_arguments(effect)
54
55     def clear_all_effects(self, clear_arguments=False):
56         if is_opengl:
57             self._shader_manager.clear_shaders()
58
59         if clear_arguments:
60             for shader_type in self.shader_arguments:
61                 self.shader_arguments[shader_type] = {}
62
63     def draw(self):
64         if is_opengl:
65             self._shader_manager.draw(self.screen, self.shader_arguments)
66         else:
67             self._native_screen.blit(self.screen, (0, 0))
68
69         self.flip()
70
71         if is_opengl:
72             self.screen.fill((0, 0, 0, 0))
73         else:
74             self.screen.fill((0, 0, 0))
75             draw_background(self.screen, self._background_image)
76
77     def update(self):
78         self.draw()
79
80     def handle_resize(self):
81         self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
82         if is_opengl:
83             self._shader_manager.handle_resize(self.size)
84         else:
85             draw_background(self.screen, self._background_image)
86
87 window = WindowManager(size=SCREEN_SIZE, resizable=True, opengl=is_opengl,
    fullscreen_desktop=is_fullscreen)

```

## B.8 data\shaders

### B.8.1 protocol.py

```

1 import pygame
2 import moderngl
3 from typing import Protocol, Optional
4 from data.utils.constants import ShaderType
5
6 class SMProtocol(Protocol):
7     def load_shader(self, shader_type: ShaderType, **kwargs) -> None: ...
8     def clear_shaders(self) -> None: ...
9     def create_vao(self, shader_type: ShaderType) -> None: ...
10    def create_framebuffer(self, shader_type: ShaderType, size: Optional[tuple[int]] = None, filter: Optional[int] = moderngl.NEAREST) -> None: ...
11    def render_to_fbo(self, shader_type: ShaderType, texture: moderngl.Texture, output_fbo: Optional[moderngl.Framebuffer] = None, program_type: Optional[ShaderType] = None, use_image: Optional[bool] = True, **kwargs) -> None: ...
12    def apply_shader(self, shader_type: ShaderType, **kwargs) -> None: ...
13    def remove_shader(self, shader_type: ShaderType) -> None: ...
14    def render_output(self, texture: moderngl.Texture) -> None: ...
15    def get_fbo_texture(self, shader_type: ShaderType) -> moderngl.Texture: ...

```

```

16     def calibrate_pygame_surface(self, pygame_surface: pygame.Surface) -> moderngl
17         .Texture: ...
18     def draw(self, surface: pygame.Surface, arguments: dict) -> None: ...
19     def __del__(self) -> None: ...
20     def cleanup(self) -> None: ...
21     def handle_resize(self, new_screen_size: tuple[int]) -> None: ...
22
23     _ctx: moderngl.Context
24     _screen_size: tuple[int]
25     _opengl_buffer: moderngl.Buffer
26     _pygame_buffer: moderngl.Buffer
27     _shader_stack: list[ShaderType]
28
29     _vert_shaders: dict
30     _frag_shaders: dict
31     _programs: dict
32     _vaos: dict
33     _textures: dict
34     _shader_passes: dict
35     framebuffers: dict

```

## B.9 data\shaders\classes

### B.9.1 base.py

```

1 import pygame
2 from data.utils.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
4
5 class Base:
6     def __init__(self, shader_manager: SMProtocol):
7         self._shader_manager = shader_manager
8
9         self._shader_manager.create_framebuffer(ShaderType.BASE)
10        self._shader_manager.create_vao(ShaderType.BACKGROUND_WAVES)
11        self._shader_manager.create_vao(ShaderType.BACKGROUND_BALATRO)
12        self._shader_manager.create_vao(ShaderType.BACKGROUND_LASERS)
13        self._shader_manager.create_vao(ShaderType.BACKGROUND_GRADIENT)
14        self._shader_manager.create_vao(ShaderType.BACKGROUND_NONE)
15
16    def apply(self, texture, background_type=None):
17        base_texture = self._shader_manager.get_fbo_texture(ShaderType.BASE)
18
19        match background_type:
20            case ShaderType.BACKGROUND_WAVES:
21                self._shader_manager.render_to_fbo(
22                    ShaderType.BASE,
23                    texture=base_texture,
24                    program_type=ShaderType.BACKGROUND_WAVES,
25                    use_image=False,
26                    time=pygame.time.get_ticks() / 1000
27                )
28            case ShaderType.BACKGROUND_BALATRO:
29                self._shader_manager.render_to_fbo(
30                    ShaderType.BASE,
31                    texture=base_texture,
32                    program_type=ShaderType.BACKGROUND_BALATRO,
33                    use_image=False,
34                    time=pygame.time.get_ticks() / 1000,
35                    screenSize=base_texture.size
36                )

```

```

37         case ShaderType.BACKGROUND_LASERS:
38             self._shader_manager.render_to_fbo(
39                 ShaderType.BASE,
40                 texture=base_texture,
41                 program_type=ShaderType.BACKGROUND_LASERS,
42                 use_image=False,
43                 time=pygame.time.get_ticks() / 1000,
44                 screenSize=base_texture.size
45             )
46         case ShaderType.BACKGROUND_GRADIENT:
47             self._shader_manager.render_to_fbo(
48                 ShaderType.BASE,
49                 texture=base_texture,
50                 program_type=ShaderType.BACKGROUND_GRADIENT,
51                 use_image=False,
52                 time=pygame.time.get_ticks() / 1000,
53                 screenSize=base_texture.size
54             )
55         case None:
56             self._shader_manager.render_to_fbo(
57                 ShaderType.BASE,
58                 texture=base_texture,
59                 program_type=ShaderType.BACKGROUND_NONE,
60                 use_image=False,
61             )
62         case _:
63             raise ValueError('(shader.py) Unknown background type:', background_type)
64
65         self._shader_manager.get_fbo_texture(ShaderType.BASE).use(1)
66         self._shader_manager.render_to_fbo(ShaderType.BASE, texture, background=1)

```

### B.9.2 blend.py

```

1 import moderngl
2 from data.utils.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
4
5 class _Blend:
6     def __init__(self, shader_manager: SMProtocol):
7         self._shader_manager = shader_manager
8
9         self._shader_manager.create_framebuffer(ShaderType._BLEND)
10
11    def apply(self, texture, texture_2, texture_2_pos):
12        self._shader_manager._ctx.blend_func = (moderngl.SRC_ALPHA, moderngl.ONE)
13
14        relative_size = (texture_2.size[0] / texture.size[0], texture_2.size[1] / texture.size[1])
15        opengl_pos = (texture_2_pos[0], 1 - texture_2_pos[1] - relative_size[1])
16
17        texture_2.use(1)
18        self._shader_manager.render_to_fbo(ShaderType._BLEND, texture, image2=1,
19        image2Pos=opengl_pos, relativeSize=relative_size)
20        self._shader_manager._ctx.blend_func = moderngl.DEFAULT_BLENDING

```

### B.9.3 bloom.py

See Section 3.9.2.

### B.9.4 blur.py

See Section 3.9.2.

### B.9.5 chromatic\_abbreviation.py

```

1 import pygame
2 from data.utils.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
4
5 CHROMATIC_ABBREVIATION_INTENSITY = 2.0
6
7 class ChromaticAbbreviation:
8     def __init__(self, shader_manager: SMProtocol):
9         self._shader_manager = shader_manager
10
11     self._shader_manager.create_framebuffer(ShaderType.CHROMATIC_ABBREVIATION)
12
13     def apply(self, texture):
14         mouse_pos = (pygame.mouse.get_pos()[0] / texture.size[0], pygame.mouse.
15         get_pos()[1] / texture.size[1])
16         self._shader_manager.render_to_fbo(ShaderType.CHROMATIC_ABBREVIATION,
17         texture, mouseFocusPoint=mouse_pos, enabled=pygame.mouse.get_pressed()[0],
18         intensity=CHROMATIC_ABBREVIATION_INTENSITY)

```

### B.9.6 crop.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3
4 class _Crop:
5     def __init__(self, shader_manager: SMProtocol):
6         self._shader_manager = shader_manager
7
8     def apply(self, texture, relative_pos, relative_size):
9         opengl_pos = (relative_pos[0], 1 - relative_pos[1] - relative_size[1])
10        pixel_size = (int(relative_size[0] * texture.size[0]), int(relative_size
11        [1] * texture.size[1]))
12
13        self._shader_manager.create_framebuffer(ShaderType._CROP, size=pixel_size)
14
15        self._shader_manager.render_to_fbo(ShaderType._CROP, texture, relativePos=
16        opengl_pos, relativeSize=relative_size)

```

### B.9.7 crt.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3
4 class CRT:
5     def __init__(self, shader_manager: SMProtocol):
6         self._shader_manager = shader_manager
7
8         shader_manager.create_framebuffer(ShaderType.CRT)
9
10    def apply(self, texture):
11        self._shader_manager.render_to_fbo(ShaderType.CRT, texture)

```

### B.9.8 grayscale.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3
4 class Grayscale:
5     def __init__(self, shader_manager: SMProtocol):
6         self._shader_manager = shader_manager
7
8     shader_manager.create_framebuffer(ShaderType.GRAYSCALE)
9
10    def apply(self, texture):
11        self._shader_manager.render_to_fbo(ShaderType.GRAYSCALE, texture)
12
13

```

### B.9.9 highlight\_brightness.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3
4 HIGHLIGHT_THRESHOLD = 0.9
5
6 class _HighlightBrightness:
7     def __init__(self, shader_manager: SMProtocol):
8         self._shader_manager = shader_manager
9
10    shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_BRIGHTNESS)
11
12    def apply(self, texture, intensity):
13        self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_BRIGHTNESS,
14                                         texture, threshold=HIGHLIGHT_THRESHOLD, intensity=intensity)

```

### B.9.10 highlight\_colour.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3
4 class _HighlightColour:
5     def __init__(self, shader_manager: SMProtocol):
6         self._shader_manager = shader_manager
7
8     shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_COLOUR)
9
10    def apply(self, texture, old_highlight, colour, intensity):
11        old_highlight.use(1)
12        self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_COLOUR, texture,
13                                         highlight=1, colour=colour, threshold=0.1, intensity=intensity)

```

### B.9.11 lightmap.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3 from data.shaders.classes.shadowmap import _Shadowmap
4
5 LIGHT_RESOLUTION = 256
6
7 class _Lightmap:
8     def __init__(self, shader_manager: SMProtocol):
9         self._shader_manager = shader_manager
10
11    shader_manager.load_shader(ShaderType._SHADOWMAP)
12
13    def apply(self, texture, colour, softShadow, occlusion=None, falloff=0.0,
14              clamp=(-180, 180)):

```

```

14     self._shader_manager.create_framebuffer(ShaderType._LIGHTMAP, size=texture
15     .size)
16     self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)
17
18     _Shadowmap(self._shader_manager).apply(texture, occlusion)
19     shadow_map = self._shader_manager.get_fbo_texture(ShaderType._SHADOWMAP)
20
21     self._shader_manager.render_to_fbo(ShaderType._LIGHTMAP, shadow_map,
22     resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
23     clamp, softShadow=softShadow)
24
25     self._shader_manager._ctx.disable(self._shader_manager._ctx.BLEND)

```

### B.9.12 occlusion.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3
4 class _Occlusion:
5     def __init__(self, shader_manager: SMProtocol):
6         self._shader_manager = shader_manager
7
8     def apply(self, texture, occlusion_colour=(255, 0, 0)):
9         self._shader_manager.create_framebuffer(ShaderType._OCCLUSION, size=
10         texture.size)
11         self._shader_manager.render_to_fbo(ShaderType._OCCLUSION, texture,
12         checkColour=tuple(num / 255 for num in occlusion_colour))

```

### B.9.13 rays.py

See Section 3.9.3.

### B.9.14 shadowmap.py

```

1 import moderngl
2 from data.utils.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
4 from data.shaders.classes.occlusion import _Occlusion
5
6 LIGHT_RESOLUTION = 256
7
8 class _Shadowmap:
9     def __init__(self, shader_manager: SMProtocol):
10         self._shader_manager = shader_manager
11
12         shader_manager.load_shader(ShaderType._OCCLUSION)
13
14     def apply(self, texture, occlusion_texture=None):
15         self._shader_manager.create_framebuffer(ShaderType._SHADOWMAP, size=(
16         texture.size[0], 1), filter=moderngl.LINEAR)
17
18         if occlusion_texture is None:
19             _Occlusion(self._shader_manager).apply(texture)
20             occlusion_texture = self._shader_manager.get_fbo_texture(ShaderType.
21             _OCCLUSION)
22
23         self._shader_manager.render_to_fbo(ShaderType._SHADOWMAP,
24         occlusion_texture, resolution=LIGHT_RESOLUTION)

```

### B.9.15 shake.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3 from random import randint
4
5 SHAKE_INTENSITY = 3
6
7 class Shake:
8     def __init__(self, shader_manager: SMProtocol):
9         self._shader_manager = shader_manager
10
11     self._shader_manager.create_framebuffer(ShaderType.SHAKE)
12
13     def apply(self, texture, intensity=SHAKE_INTENSITY):
14         displacement = (randint(-intensity, intensity) / 1000, randint(-intensity,
15         intensity) / 1000)
16         self._shader_manager.render_to_fbo(ShaderType.SHAKE, texture, displacement
17         =displacement)

```

### B.9.16 \_\_init\_\_.py

```

1 from data.shaders.classes.chromatic_abbreviation import ChromaticAbbreviation
2 from data.shaders.classes.highlight_brightness import _HighlightBrightness
3 from data.shaders.classes.highlight_colour import _HighlightColour
4 from data.shaders.classes.shadowmap import _Shadowmap
5 from data.shaders.classes.occlusion import _Occlusion
6 from data.shaders.classes.grayscale import Grayscale
7 from data.shaders.classes.lightmap import _Lightmap
8 from data.shaders.classes.blend import _Blend
9 from data.shaders.classes.shake import Shake
10 from data.shaders.classes.bloom import Bloom
11 from data.shaders.classes.blur import _Blur
12 from data.shaders.classes.crop import _Crop
13 from data.shaders.classes.rays import Rays
14 from data.shaders.classes.base import Base
15 from data.shaders.classes.crt import CRT
16 from data.utils.constants import ShaderType
17
18 shader_pass_lookup = {
19     ShaderType.CHROMATIC_ABBREVIATION: ChromaticAbbreviation,
20     ShaderType.GRAYSCALE: Grayscale,
21     ShaderType.SHAKE: Shake,
22     ShaderType.BLOOM: Bloom,
23     ShaderType.BASE: Base,
24     ShaderType.RAYS: Rays,
25     ShaderType.CRT: CRT,
26
27     ShaderType._HIGHLIGHT_BRIGHTNESS: _HighlightBrightness,
28     ShaderType._HIGHLIGHT_COLOUR: _HighlightColour,
29     ShaderType._CALIBRATE: lambda *args: None,
30     ShaderType._OCCLUSION: _Occlusion,
31     ShaderType._SHADOWMAP: _Shadowmap,
32     ShaderType._LIGHTMAP: _Lightmap,
33     ShaderType._BLEND: _Blend,
34     ShaderType._BLUR: _Blur,
35     ShaderType._CROP: _Crop,
36 }

```

## B.10 data\shaders\fragments

### B.10.1 background\_balatro.frag

```

1 # version 330 core
2
3 // Original by localthunk (https://www.playbalatro.com)
4
5 // Configuration (modify these values to change the effect)
6 #define SPIN_ROTATION -2.0
7 #define SPIN_SPEED 7.0
8 #define OFFSET vec2(0.0)
9 #define COLOUR_2 vec4(0.871, 0.267, 0.231, 1.0)
10 #define COLOUR_1 vec4(0.0, 0.42, 0.706, 1.0)
11 #define COLOUR_3 vec4(0.086, 0.137, 0.145, 1.0)
12 #define CONTRAST 3.5
13 #define LIGHTHING 0.4
14 #define SPIN_AMOUNT 0.25
15 #define PIXEL_FILTER 745.0
16 #define SPIN_EASE 1.0
17 #define PI 3.14159265359
18 #define IS_ROTATE false
19
20 uniform float time;
21 uniform vec2 screenSize;
22
23 in vec2 uvs;
24 out vec4 f_colour;
25
26 vec4 effect(vec2 screenSize, vec2 screen_coords) {
27     float pixel_size = length(screenSize.xy) / PIXEL_FILTER;
28     vec2 uv = (floor(screen_coords.xy*(1./pixel_size)))*pixel_size - 0.5*screenSize
29     .xy)/length(screenSize.xy) - OFFSET;
29     float uv_len = length(uv);
30
31     float speed = (SPIN_ROTATION*SPIN_EASE*0.2);
32     if(IS_ROTATE){
33         speed = time * speed;
34     }
35     speed += 302.2;
36     float new_pixel_angle = atan(uv.y, uv.x) + speed - SPIN_EASE*20.*(
37     SPIN_AMOUNT*uv_len + (1. - 1.*SPIN_AMOUNT));
37     vec2 mid = (screenSize.xy/length(screenSize.xy))/2.;
38     uv = (vec2((uv_len * cos(new_pixel_angle) + mid.x), (uv_len * sin(
39     new_pixel_angle) + mid.y)) - mid);
39
40     uv *= 30.;
41     speed = time*(SPIN_SPEED);
42     vec2 uv2 = vec2(uv.x+uv.y);
43
44     for(int i=0; i < 5; i++) {
45         uv2 += sin(max(uv.x, uv.y)) + uv;
46         uv += 0.5*vec2(cos(5.1123314 + 0.353*uv2.y + speed*0.131121), sin(uv2.x -
47         0.113*speed));
47         uv -= 1.0*cos(uv.x + uv.y) - 1.0*sin(uv.x*0.711 - uv.y);
48     }
49
50     float contrast_mod = (0.25*CONTRAST + 0.5*SPIN_AMOUNT + 1.2);
51     float paint_res = min(2., max(0., length(uv)*(0.035)*contrast_mod));
52     float c1p = max(0., 1. - contrast_mod*abs(1.-paint_res));
53     float c2p = max(0., 1. - contrast_mod*abs(paint_res));
54     float c3p = 1. - min(1., c1p + c2p);
55     float light = (LIGHTHING - 0.2)*max(c1p*5. - 4., 0.) + LIGHTHING*max(c2p*5. -
56     4., 0.);
56     return (0.3/CONTRAST)*COLOUR_1 + (1. - 0.3/CONTRAST)*(COLOUR_1*c1p + COLOUR_2*
c2p + vec4(c3p*COLOUR_3.rgb, c3p*COLOUR_1.a)) + light;

```

```

57 }
58
59 void main() {
60     f_colour = effect(screenSize.xy, uvs* screenSize.xy);
61 }
```

### B.10.2 background\_gradient.frag

```

1 // Modified from https://www.shadertoy.com/view/wdyczG
2
3 #version 330 core
4
5 uniform float time;
6 uniform vec2 screenSize;
7
8 in vec2 uvs;
9 out vec4 f_colour;
10
11 #define S(a,b,t) smoothstep(a,b,t)
12
13 mat2 Rot(float a)
14 {
15     float s = sin(a);
16     float c = cos(a);
17     return mat2(c, -s, s, c);
18 }
19
20 // Created by inigo quilez - iq/2014
21 // License Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported
22 // License.
23 vec2 hash( vec2 p )
24 {
25     p = vec2( dot(p,vec2(2127.1,81.17)), dot(p,vec2(1269.5,283.37)) );
26     return fract(sin(p)*43758.5453);
27 }
28
29 float noise( in vec2 p )
30 {
31     vec2 i = floor( p );
32     vec2 f = fract( p );
33
34     vec2 u = f*f*(3.0-2.0*f);
35
36     float n = mix( mix( dot( -1.0+2.0*hash( i + vec2(0.0,0.0) ), f - vec2(0.0,0.0)
37         ),
38                     dot( -1.0+2.0*hash( i + vec2(1.0,0.0) ), f - vec2(1.0,0.0)
39                     ),
40                     mix( dot( -1.0+2.0*hash( i + vec2(0.0,1.0) ), f - vec2(0.0,1.0)
41                     ),
42                     dot( -1.0+2.0*hash( i + vec2(1.0,1.0) ), f - vec2(1.0,1.0)
43                     ),
44                     0.5 + 0.5*n,
45     );
46
47     void main() {
48         float ratio = screenSize.x / screenSize.y;
49
50         vec2 tuv = uvs;
51         tuv -= .5;
52
53         // rotate with Noise
54         float degree = noise(vec2(time*.1, tuv.x*tuv.y));
55     }
56 }
```

```

50
51     tuv.y *= 1./ratio;
52     tuv *= Rot(radians((degree-.5)*720.+180.));
53     tuv.y *= ratio;
54
55     // Wave warp with sin
56     float frequency = 5.;
57     float amplitude = 30.;
58     float speed = time * 2.;
59     tuv.x += sin(tuv.y*frequency+speed)/amplitude;
60     tuv.y += sin(tuv.x*frequency*1.5+speed)/(amplitude*.5);
61
62     // draw the image
63     vec3 colorYellow = vec3(.957, .804, .623);
64     vec3 colorDeepBlue = vec3(.192, .384, .933);
65     vec3 layer1 = mix(colorYellow, colorDeepBlue, S(-.3, .2, (tuv*Rot(radians(-5.))).x));
66
67     vec3 colorRed = vec3(.910, .510, .8);
68     vec3 colorBlue = vec3(0.350, .71, .953);
69     vec3 layer2 = mix(colorRed, colorBlue, S(-.3, .2, (tuv*Rot(radians(-5.))).x));
70
71     vec3 finalComp = mix(layer1, layer2, S(.5, -.3, tuv.y));
72
73     vec3 col = finalComp;
74
75     f_colour = vec4(col,1.0);
76 }

```

### B.10.3 background\_lasers.frag

```

1 // Modified from https://www.shadertoy.com/view/7tBSR1
2 // rand [0,1] https://www.shadertoy.com/view/4djSRW
3
4 #version 330 core
5
6 uniform float time;
7 uniform vec2 screenSize;
8
9 in vec2 uvs;
10 out vec4 f_colour;
11
12 float rand(vec2 p) {
13     p *= 500.0;
14     vec3 p3 = fract(vec3(p.xyx) * .1031);
15     p3 += dot(p3, p3.yzx + 33.33);
16     return fract((p3.x + p3.y) * p3.z);
17 }
18
19 // value noise
20 float noise(vec2 p) {
21     vec2 f = smoothstep(0.0, 1.0, fract(p));
22     vec2 i = floor(p);
23     float a = rand(i);
24     float b = rand(i+vec2(1.0,0.0));
25     float c = rand(i+vec2(0.0,1.0));
26     float d = rand(i+vec2(1.0,1.0));
27     return mix(mix(a, b, f.x), mix(c, d, f.x), f.y);
28 }
29
30 // fractal noise
31 float fbm(vec2 p) {

```

```

32     float a = 0.5;
33     float r = 0.0;
34     for (int i = 0; i < 8; i++) {
35         r += a*noise(p);
36         a *= 0.5;
37         p *= 2.0;
38     }
39     return r;
40 }
41
42 // lasers originating from a central point
43 float laser(vec2 p, int num) {
44     float r = atan(p.x, p.y);
45     float sn = sin(r*float(num)+time);
46     float lsr = 0.5+0.5*sn;
47     lsr = lsr*lzr*lzr*lzr*lzr;
48     float glow = pow(clamp(sn, 0.0, 1.0),100.0);
49     return lsr+glow;
50 }
51
52 // mix of fractal noises to simulate fog
53 float clouds(vec2 uv) {
54     vec2 t = vec2(0,time);
55     float c1 = fbm(fbm(uv*3.0)*0.75+uv*3.0+t/3.0);
56     float c2 = fbm(fbm(uv*2.0)*0.5+uv*7.0+t/3.0);
57     float c3 = fbm(fbm(uv*10.0-t)*0.75+uv*5.0+t/6.0);
58     float r = mix(c1, c2, c3*c3);
59     return r*r;
60 }
61
62 void main() {
63     vec2 hs = screenSize.xy/screenSize.y*0.5;
64     vec2 uvc = uvs-hs;
65     float l = (1.0 + 3.0*noise(vec2(15.0-time)))
66     * laser(vec2(uvs.x+0.5, uvs.y*(0.5 + 10.0*noise(vec2(time/5.0)))) + 0.1),
67     15);
68     l += fbm(vec2(2.0*time))
69     * laser(vec2(hs.x-uvc.x-0.2, uvs.y+0.1), 25);
70     l += noise(vec2(time-73.0))
71     * laser(vec2(uvc.x, 1.0-uvs.y+0.5), 30);
72     float c = clouds(uvs);
73     vec4 col = vec4(uvs.x, 0.0, 1-uvs.x, 1.0)*(uvs.y*l+uvs.y*uvs.y)*c;
74
75     f_colour = pow(col, vec4(0.75));
76 }

```

#### B.10.4 background\_none.frag

```

1 # version 330 core
2
3 in vec2 uvs;
4 out vec4 f_colour;
5
6 void main() {
7     f_colour = vec4(vec3(0.0 + uvs.x * 0.001), 1.0);
8 }

```

#### B.10.5 background\_waves.frag

```

1 // Modified from https://godotshaders.com/shader/discrete-ocean/
2

```

```

3 # version 330 core
4
5 uniform float wave_amp=1.0;
6 uniform float wave_size=4.0;
7 uniform float wave_time_mul=0.2;
8
9 uniform int total_phases=20;
10
11 uniform vec4 bottom_color=vec4(0.608, 0.133, 0.167, 1.0);
12 uniform vec4 top_color=vec4(0.110, 0.149, 0.220, 1.0);
13
14 // uniform vec4 bottom_color=vec4(0.38, 0.04, 0.71, 1.0);
15 // uniform vec4 top_color=vec4(0.15, 0.02, 0.49, 1.0);
16
17 uniform float time;
18
19 in vec2 uvs;
20 out vec4 f_colour;
21
22 #define PI 3.14159
23
24 float rand (float n) {
25     return fract(sin(n) * 43758.5453123);
26 }
27 float noise (float p){
28     float fl = floor(p);
29     float fc = fract(p);
30     return mix(rand(fl), rand(fl + 1.0), fc);
31 }
32 float fmod(float x, float y) {
33     return x - floor(x / y) * y;
34 }
35 vec4 lerp(vec4 a, vec4 b, float w) {
36     return a + w * (b - a);
37 }
38
39 void main() {
40     float t = float(total_phases);
41     float effective_wave_amp = min(wave_amp, 0.5 / t);
42     float d = fmod(uvs.y, 1.0 / t);
43     float i = floor(uvs.y * t);
44     float vi = floor(uvs.y * t + t * effective_wave_amp);
45     float s = effective_wave_amp * sin((uvs.x + time * max(1.0 / t, noise(vi)) *
46         wave_time_mul * vi / t) * 2.0 * PI * wave_size);
47
48     if (d < s) i--;
49     if (d > s + 1.0 / t) i++;
50     i = clamp(i, 0.0, t - 1.0);
51
52     f_colour = lerp(top_color, bottom_color, i / (t - 1.0));
53 }

```

### B.10.6 base.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4 uniform sampler2D background;
5
6 in vec2 uvs;
7 out vec4 f_colour;
8

```

```

9 void main() {
10     vec4 colour = texture(image, uvs);
11
12     if (colour.a == 1.0) {
13         f_colour = colour;
14     } else {
15         f_colour = texture(background, uvs);
16     }
17 }
```

### B.10.7 blend.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4 uniform sampler2D image2;
5 uniform vec2 relativeSize;
6 uniform vec2 image2Pos;
7
8 in vec2 uvs;
9 out vec4 f_colour;
10
11 // void main() {
12 //     f_colour = vec4(texture(image, uvs).rgba);
13 // }
14
15 void main() {
16     vec3 colour = texture(image, uvs).rgb;
17
18     vec2 image2Coords = vec2((uvs.x - image2Pos.x) / relativeSize.x, (uvs.y -
19     image2Pos.y) / relativeSize.y);
20
21     float withinBounds = step(image2Pos.x, uvs.x) * step(uvs.x, (image2Pos.x +
22     relativeSize.x)) * step(image2Pos.y, uvs.y) * step(uvs.y, (image2Pos.y +
23     relativeSize.y));
24
25     f_colour = vec4(colour + (texture(image2, image2Coords).rgb * withinBounds),
26     1.0);
27
28     // if (image2Pos.x < uvs.x &&
29     //     uvs.x < (image2Pos.x + relativeSize.x) &&
30     //     image2Pos.y < uvs.y &&
31     //     uvs.y < (image2Pos.y + relativeSize.y)) {
32
33         //     vec2 image2Coords = vec2((uvs.x - image2Pos.x) / relativeSize.x, (uvs.y -
34         //     image2Pos.y) / relativeSize.y);
35         //     colour += texture(image2, image2Coords).rgb;
36     // }
37
38     // f_colour = vec4(colour, 1.0);
39 }
```

### B.10.8 bloom.frag

```

1 #version 330 core
2
3 in vec2 uvs;
4 out vec4 f_colour;
5
6 uniform sampler2D image;
7 uniform sampler2D blurredImage;
```

```

8 uniform float intensity;
9
10 void main() {
11     vec3 baseColour = texture(image, uvs).rgb;
12     vec3 bloomColor = texture(blurredImage, uvs).rgb;
13
14     baseColour += bloomColor * intensity;
15     f_colour = vec4(baseColour, 1.0);
16 }

```

### B.10.9 bloom\_old.frag

```

1 #version 330 core
2
3 in vec2 uvs;
4 out vec4 f_colour;
5
6 uniform sampler2D image;
7 uniform float bloom_spread = 0.1;
8 uniform float bloom_intensity = 0.5;
9
10 void main() {
11     ivec2 size = textureSize(image, 0);
12
13     float uv_x = uvs.x * size.x;
14     float uv_y = uvs.y * size.y;
15
16     vec4 sum = vec4(0.0);
17
18     for (int n = 0; n < 9; ++n) {
19         uv_y = (uvs.y * size.y) + (bloom_spread * float(n - 4));
20         vec4 h_sum = vec4(0.0);
21         h_sum += texelFetch(image, ivec2(uv_x - (4.0 * bloom_spread), uv_y), 0);
22         h_sum += texelFetch(image, ivec2(uv_x - (3.0 * bloom_spread), uv_y), 0);
23         h_sum += texelFetch(image, ivec2(uv_x - (2.0 * bloom_spread), uv_y), 0);
24         h_sum += texelFetch(image, ivec2(uv_x - bloom_spread, uv_y), 0);
25         h_sum += texelFetch(image, ivec2(uv_x, uv_y), 0);
26         h_sum += texelFetch(image, ivec2(uv_x + bloom_spread, uv_y), 0);
27         h_sum += texelFetch(image, ivec2(uv_x + (2.0 * bloom_spread), uv_y), 0);
28         h_sum += texelFetch(image, ivec2(uv_x + (3.0 * bloom_spread), uv_y), 0);
29         h_sum += texelFetch(image, ivec2(uv_x + (4.0 * bloom_spread), uv_y), 0);
30         sum += h_sum / 9.0;
31     }
32
33     f_colour = texture(image, uvs) + ((sum / 9.0) * bloom_intensity);
34 }

```

### B.10.10 blur.frag

See Section 3.9.2.

### B.10.11 box\_blur.frag

```

1 # version 330 core
2
3 uniform sampler2D image;
4
5 uniform int size=1;
6 uniform int separation=1;
7

```

```

8  in  vec2 uvs;
9  out vec4 f_colour;
10
11 vec2 textureSize = textureSize(image, 0);
12
13 void main() {
14     if (size <= 0) {
15         return;
16     }
17
18     float count = 0.0;
19
20     for (int i = -size ; i <= size ; ++i) {
21         for (int j = -size ; j <= size ; ++j) {
22             f_colour += texture(image, uvs + (vec2(i, j) * separation) /
23             textureSize).rgba;
24             count += 1.0;
25         }
26     }
27
28     f_colour.rgb /= count;
29 }
```

### B.10.12 calibrate.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4
5 in vec2 uvs;
6 out vec4 f_colour;
7
8 void main() {
9     f_colour = vec4(texture(image, uvs).rgba);
10 }
```

### B.10.13 chromatic\_abbreviation.frag

```

1 #version 330 core
2
3 in vec2 uvs;
4 out vec4 f_colour;
5
6 uniform sampler2D image;
7
8 uniform bool enabled;
9 uniform vec2 mouseFocusPoint;
10 uniform float intensity;
11
12 void main() {
13     if (!enabled) {
14         f_colour = texture(image, uvs);
15         return;
16     }
17
18     float redOffset = 0.009 * intensity;
19     float greenOffset = 0.006 * intensity;
20     float blueOffset = -0.006 * intensity;
21
22     vec2 texSize = textureSize(image, 0).xy;
```

```

23     vec2 direction = uvs - mouseFocusPoint;
24
25     f_colour = texture(image, uvs);
26
27     f_colour.r = texture(image, uvs + (direction * vec2(redOffset))).r;
28     f_colour.g = texture(image, uvs + (direction * vec2(greenOffset))).g;
29     f_colour.b = texture(image, uvs + (direction * vec2(blueOffset))).b;
30 }

```

### B.10.14 crop.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4 uniform vec2 relativeSize;
5 uniform vec2 relativePos;
6
7 in vec2 uvs;
8 out vec4 f_colour;
9
10 void main() {
11     vec2 sampleCoords = relativeSize.xy * uvs.xy + relativePos.xy;
12
13     float withinBounds = step(0.0, sampleCoords.x) * step(sampleCoords.x, 1.0) *
14         step(0.0, sampleCoords.y) * step(sampleCoords.y, 1.0);
15
16     vec3 colour = texture(image, sampleCoords).rgb * withinBounds;
17     colour.r += (1 - withinBounds);
18
19     f_colour = vec4(colour, 1.0);
}

```

### B.10.15 crt.frag

```

1 #version 330 core
2
3 precision mediump float;
4 uniform sampler2D image;
5
6 in vec2 uvs;
7 out vec4 f_colour;
8 uniform int mode = 1;
9
10 void main() {
11     if (mode == 0){
12         f_colour = vec4(texture(image, uvs).rgb, 1.0);
13     }
14     else {
15         float flatness = 1.0;
16
17         if (mode == 1) flatness = 5.0;
18         else if(mode == 2) flatness = 10.0;
19
20         vec2 center = vec2(0.5, 0.5);
21         vec2 off_center = uvs - center;
22
23         off_center *= 1.0 + 0.8 * pow(abs(off_center.yx), vec2(flatness));
24         // 1.0 -> 1.5 make distance to screen
25         // vec 2 -> screen flatness
26
27         vec2 uvs_2 = center+off_center;
}

```

```

28
29     if (uvs_2.x > 1.0 || uvs_2.x < 0.0 || uvs_2.y > 1.0 || uvs_2.y < 0.0) {
30         f_colour=vec4(0.0, 0.0, 0.0, 1.0);
31
32     } else {
33         f_colour = vec4(texture(image, uvs_2).rgb, 1.0);
34         float fv = fract(uvs_2.y * float(textureSize(image,0).y));
35         fv = min(1.0, 0.8+0.5*min(fv, 1.0-fv));
36         f_colour.rgb *= fv;
37     }
38 }
39 }
```

### B.10.16 flashlight.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4 uniform vec2 center;
5
6 in vec2 uvs;
7 out vec4 f_colour;
8
9 vec2 resolution = textureSize(image, 0);
10 float radius = 100.0; // radius in pixel
11
12 float getDistance(vec2 pixelCoord, vec2 playerCoord) {
13     return distance(pixelCoord*resolution, playerCoord);
14 }
15
16 void main() {
17     float distance = getDistance(uvs, center);
18     float a = 0;
19     float b = 1;
20
21     // if (distance < radius)
22     float factor = 1.0 / (pow((distance / 100), 2) + 1);
23     float isLit = step(distance, 10000);
24
25     f_colour = vec4(texture(image, uvs).rgb + factor * isLit, 1.0);
26
27     // if (distance < 10000) {
28     //     float factor = 1.0 / (pow((distance / 100), 2) + 1);
29     //     f_colour = vec4(texture(image, uvs).rgb + factor, 1.0);
30     // }
31     // else {
32     //     f_colour = vec4(texture(image, uvs).rgb, 1.0);
33     // }
34 }
```

### B.10.17 grayscale.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4
5 in vec2 uvs;
6 out vec4 f_colour;
7
8 void main() {
9     f_colour = vec4(texture(image, uvs).rgb, 1.0);
```

```

10     float gray = dot(f_colour.rgb, vec3(0.299, 0.587, 0.114));
11     f_colour.rgb = vec3(gray, gray, gray);
12 }
```

**B.10.18 highlight\_brightness.frag**

See Section 3.9.2.

**B.10.19 highlight\_colour.frag**

```

1 # version 330 core
2
3 uniform sampler2D image;
4 uniform sampler2D highlight;
5
6 uniform vec3 colour;
7 uniform float threshold;
8 uniform float intensity;
9
10 in vec2 uvs;
11 out vec4 f_colour;
12
13 vec3 normColour = colour / 255;
14
15 void main() {
16     vec4 pixel = texture(image, uvs);
17     float isClose = step(abs(pixel.r - normColour.r), threshold) * step(abs(pixel.g - normColour.g), threshold) * step(abs(pixel.b - normColour.b), threshold);
18
19     if (isClose == 1.0) {
20         f_colour = vec4(vec3(pixel.rgb * intensity), 1.0);
21     } else {
22         f_colour = vec4(texture(highlight, uvs).rgb, 1.0);
23     }
24 }
```

**B.10.20 lightmap.frag**

See Section 3.9.3.

**B.10.21 occlusion.frag**

See Section 3.9.3.

**B.10.22 rays.frag**

```

1 #version 330 core
2
3 uniform sampler2D image;
4
5 in vec2 uvs;
6 out vec4 f_colour;
7
8 void main() {
9     f_colour = vec4(texture(image, uvs).rgb, 1.0);
10 }
```

### B.10.23 shadowmap.frag

See Section 3.9.3.

### B.10.24 shake.frag

```

1 #version 330 core
2
3 uniform sampler2D image;
4 uniform vec2 displacement;
5
6 in vec2 uvs;
7 out vec4 f_colour;
8
9 void main() {
10     f_colour = vec4(texture(image, uvs + displacement).rgb, 1.0);
11 }
```

## B.11 data\shaders\vertex

### B.11.1 base.vert

```

1 #version 330 core
2
3 in vec2 vert;
4 in vec2 texCoords;
5 out vec2 uvs;
6
7 void main() {
8     uvs = texCoords;
9     gl_Position = vec4(vert, 0.0, 1.0);
10 }
```

## B.12 data\states

### B.13 data\states\browser

#### B.13.1 browser.py

```

1 import pygame
2 import pyperclip
3 from data.helpers.database_helpers import delete_game, get_ordered_games
4 from data.states.browser.widget_dict import BROWSER_WIDGETS
5 from data.utils.event_types import BrowserEventType
6 from data.managers.logs import initialise_logger
7 from data.utils.constants import GAMES_PER_PAGE
8 from data.managers.window import window
9 from data.utils.enums import ShaderType
10 from data.utils.assets import MUSIC
11 from data.control import _State
12 from random import randint
13
14 logger = initialise_logger(__name__)
15
16 class Browser(_State):
17     def __init__(self):
18         super().__init__()
```

```

19
20     self._selected_index = None
21     self._filter_column = 'number_of_ply'
22     self._filter_ascend = False
23     self._games_list = []
24     self._page_number = 1
25
26     def cleanup(self):
27         super().cleanup()
28
29         if self._selected_index is not None:
30             return self._games_list[self._selected_index]
31
32     return None
33
34     def startup(self, persist=None):
35         self.refresh_games_list() # BEFORE RESIZE TO FILL WIDGET BEFORE RESIZING
36         super().startup(BROWSER_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
37
38         self._filter_column = 'number_of_ply'
39         self._filter_ascend = False
40
41         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
42                                     BACKGROUND_BALATRO)
43
44         BROWSER_WIDGETS['help'].kill()
45         BROWSER_WIDGETS['browser_strip'].kill()
46
47         self.draw()
48
49     def refresh_games_list(self):
50         column_map = {
51             'moves': 'number_of_ply',
52             'winner': 'winner',
53             'time': 'created_dt'
54         }
55
56         ascend_map = {
57             'asc': True,
58             'desc': False
59         }
60
61         filter_column = BROWSER_WIDGETS['filter_column_dropdown'].
62         get_selected_word()
63         filter_ascend = BROWSER_WIDGETS['filter_ascend_dropdown'].
64         get_selected_word()
65
66         self._selected_index = None
67
68         start_row = (self._page_number - 1) * GAMES_PER_PAGE + 1
69         end_row = (self._page_number) * GAMES_PER_PAGE
70         self._games_list = get_ordered_games(column_map[filter_column], ascend_map
71 [filter_ascend], start_row=start_row, end_row=end_row)
72
73         BROWSER_WIDGETS['browser_strip'].initialise_games_list(self._games_list)
74         BROWSER_WIDGETS['browser_strip'].set_surface_size(window.size)
75         BROWSER_WIDGETS['scroll_area'].set_image()
76
77     def get_event(self, event):
78         widget_event = self._widget_group.process_event(event)
79
80         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:

```

```

77         BROWSER_WIDGETS['help'].kill()
78
79     if widget_event is None:
80         return
81
82     match widget_event.type:
83         case BrowserEventType.MENU_CLICK:
84             self.next = 'menu'
85             self.done = True
86
87         case BrowserEventType.BROWSER_STRIP_CLICK:
88             self._selected_index = widget_event.selected_index
89
90         case BrowserEventType.COPY_CLICK:
91             if self._selected_index is None:
92                 return
93             logger.info('COPYING TO CLIPBOARD:', self._games_list[self._selected_index]['fen_string'])
94             pyperclip.copy(self._games_list[self._selected_index]['fen_string'])
95
96         case BrowserEventType.DELETE_CLICK:
97             if self._selected_index is None:
98                 return
99             delete_game(self._games_list[self._selected_index]['id'])
100            self.refresh_games_list()
101
102         case BrowserEventType.REVIEW_CLICK:
103             if self._selected_index is None:
104                 return
105
106             self.next = 'review'
107             self.done = True
108
109         case BrowserEventType.FILTER_COLUMN_CLICK:
110             selected_word = BROWSER_WIDGETS['filter_column_dropdown'].get_selected_word()
111
112             if selected_word is None:
113                 return
114
115             self.refresh_games_list()
116
117         case BrowserEventType.FILTER_ASCEND_CLICK:
118             selected_word = BROWSER_WIDGETS['filter_ascend_dropdown'].get_selected_word()
119
120             if selected_word is None:
121                 return
122
123             self.refresh_games_list()
124
125         case BrowserEventType.PAGE_CLICK:
126             self._page_number = widget_event.data
127
128             self.refresh_games_list()
129
130         case BrowserEventType.HELP_CLICK:
131             self._widget_group.add(BROWSER_WIDGETS['help'])
132             self._widget_group.handle_resize(window.size)
133
134     def draw(self):

```

```

135         self._widget_group.draw()

B.13.2 widget_dict.py

1  from data.helpers.database_helpers import get_number_of_games
2  from data.components.custom_event import CustomEvent
3  from data.utils.event_types import BrowserEventType
4  from data.utils.constants import GAMES_PER_PAGE
5  from data.utils.assets import GRAPHICS
6  from data.widgets import *
7
8  BROWSER_HEIGHT = 0.6
9
10 browser_strip = BrowserStrip(
11     relative_position=(0.0, 0.0),
12     relative_height=BROWSER_HEIGHT,
13     games_list=[]
14 )
15
16 number_of_pages = get_number_of_games() // GAMES_PER_PAGE + 1
17
18 carousel_widgets = {
19     i: Text(
20         relative_position=(0, 0),
21         relative_size=(0.3, 0.1),
22         text=f"PAGE {i} OF {number_of_pages}",
23         fill_colour=(0, 0, 0, 0),
24         fit_vertical=False,
25         border_width=0,
26     )
27     for i in range(1, number_of_pages + 1)
28 }
29
30 sort_by_container = Rectangle(
31     relative_size=(0.5, 0.1),
32     relative_position=(0.01, 0.77),
33     anchor_x='right',
34     visible=True
35 )
36
37 buttons_container = Rectangle(
38     relative_position=(0, 0.025),
39     relative_size=(0.5, 0.1),
40     scale_mode='height',
41     anchor_x='center'
42 )
43
44 top_right_container = Rectangle(
45     relative_position=(0, 0),
46     relative_size=(0.15, 0.075),
47     fixed_position=(5, 5),
48     anchor_x='right',
49     scale_mode='height'
50 )
51
52 BROWSER_WIDGETS = {
53     'help':
54     Icon(
55         relative_position=(0, 0),
56         relative_size=(1.02, 1.02),
57         icon=GRAPHICS['browser_help'],
58         anchor_x='center',

```

```

59         anchor_y='center',
60         border_width=0,
61         fill_colour=(0, 0, 0, 0)
62     ),
63     'default': [
64         buttons_container,
65         sort_by_container,
66         top_right_container,
67         ReactiveIconButton(
68             parent=top_right_container,
69             relative_position=(0, 0),
70             relative_size=(1, 1),
71             anchor_x='right',
72             scale_mode='height',
73             base_icon=GRAPHICS['home_base'],
74             hover_icon=GRAPHICS['home_hover'],
75             press_icon=GRAPHICS['home_press'],
76             event=CustomEvent(BrowserEventType.MENU_CLICK)
77         ),
78         ReactiveIconButton(
79             parent=top_right_container,
80             relative_position=(0, 0),
81             relative_size=(1, 1),
82             scale_mode='height',
83             base_icon=GRAPHICS['help_base'],
84             hover_icon=GRAPHICS['help_hover'],
85             press_icon=GRAPHICS['help_press'],
86             event=CustomEvent(BrowserEventType.HELP_CLICK)
87         ),
88         ReactiveIconButton(
89             parent=buttons_container,
90             relative_position=(0, 0),
91             relative_size=(1, 1),
92             scale_mode='height',
93             base_icon=GRAPHICS['copy_base'],
94             hover_icon=GRAPHICS['copy_hover'],
95             press_icon=GRAPHICS['copy_press'],
96             event=CustomEvent(BrowserEventType.COPY_CLICK),
97         ),
98         ReactiveIconButton(
99             parent=buttons_container,
100            relative_position=(0, 0),
101            relative_size=(1, 1),
102            scale_mode='height',
103            anchor_x='center',
104            base_icon=GRAPHICS['delete_base'],
105            hover_icon=GRAPHICS['delete_hover'],
106            press_icon=GRAPHICS['delete_press'],
107            event=CustomEvent(BrowserEventType.DELETE_CLICK),
108        ),
109        ReactiveIconButton(
110            parent=buttons_container,
111            relative_position=(0, 0),
112            relative_size=(1, 1),
113            scale_mode='height',
114            anchor_x='right',
115            base_icon=GRAPHICS['review_base'],
116            hover_icon=GRAPHICS['review_hover'],
117            press_icon=GRAPHICS['review_press'],
118            event=CustomEvent(BrowserEventType.REVIEW_CLICK),
119        ),
120        Text(

```

```

121         parent=sort_by_container,
122         relative_position=(0, 0),
123         relative_size=(0.3, 1),
124         fit_vertical=False,
125         text='SORT BY:',
126         border_width=0,
127         fill_colour=(0, 0, 0, 0)
128     )
129 ],
130 'browser_strip':
131     browser_strip,
132 'scroll_area':
133 ScrollArea(
134     relative_position=(0.0, 0.15),
135     relative_size=(1, BROWSER_HEIGHT),
136     vertical=False,
137     widget=browser_strip
138 ),
139 'filter_column_dropdown':
140 Dropdown(
141     parent=sort_by_container,
142     relative_position=(0.3, 0),
143     relative_height=0.75,
144     anchor_x='right',
145     word_list=['time', 'moves', 'winner'],
146     fill_colour=(255, 100, 100),
147     event=CustomEvent(BrowserEventType.FILTER_COLUMN_CLICK)
148 ),
149 'filter_ascend_dropdown':
150 Dropdown(
151     parent=sort_by_container,
152     relative_position=(0, 0),
153     relative_height=0.75,
154     anchor_x='right',
155     word_list=['desc', 'asc'],
156     fill_colour=(255, 100, 100),
157     event=CustomEvent(BrowserEventType.FILTER_ASCEND_CLICK)
158 ),
159 'page_carousel':
160 Carousel(
161     relative_position=(0.01, 0.77),
162     margin=5,
163     widgets_dict=carousel_widgets,
164     event=CustomEvent(BrowserEventType.PAGE_CLICK),
165   )
166 }

```

## B.14 data\states\config

### B.14.1 config.py

```

1 import pygame
2 from data.states.config.default_config import default_config
3 from data.states.config.widget_dict import CONFIG_WIDGETS
4 from data.utils.event_types import ConfigEventType
5 from data.managers.logs import initialise_logger
6 from data.managers.animation import animation
7 from data.utils.constants import ShaderType
8 from data.utils.assets import MUSIC, SFX
9 from data.managers.window import window
10 from data.managers.audio import audio

```

```

11 from data.managers.theme import theme
12 from data.utils.enums import Colour
13 from data.control import _State
14 from random import randint
15
16 logger = initialise_logger(__name__)
17
18 class Config(_State):
19     def __init__(self):
20         super().__init__()
21
22         self._config = None
23         self._valid_fen = True
24         self._selected_preset = None
25
26     def cleanup(self):
27         super().cleanup()
28
29         window.clear_apply_arguments(ShaderType.BLOOM)
30
31     return self._config
32
33     def startup(self, persist=None):
34         super().startup(CONFIG_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
35         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.9)
36
37         CONFIG_WIDGETS['invalid_fen_string'].kill()
38         CONFIG_WIDGETS['help'].kill()
39
40         self._config = default_config
41
42         if persist:
43             self._config['FEN_STRING'] = persist
44
45         self.set_fen_string(self._config['FEN_STRING'])
46         self.toggle_pvc(self._config['CPU_ENABLED'])
47         self.set_active_colour(self._config['COLOUR'])
48
49         CONFIG_WIDGETS['cpu_depth_carousel'].set_to_key(self._config['CPU_DEPTH'])
50         if self._config['CPU_ENABLED']:
51             self.create_depth_picker()
52         else:
53             self.remove_depth_picker()
54
55         self.draw()
56
57     def create_depth_picker(self):
58         # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.8))
59         # CONFIG_WIDGETS['start_button'].set_image()
60         CONFIG_WIDGETS['cpu_depth_carousel'].set_surface_size(window.size)
61         CONFIG_WIDGETS['cpu_depth_carousel'].set_image()
62         CONFIG_WIDGETS['cpu_depth_carousel'].set_geometry()
63         self._widget_group.add(CONFIG_WIDGETS['cpu_depth_carousel'])
64
65     def remove_depth_picker(self):
66         # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.7))
67         # CONFIG_WIDGETS['start_button'].set_image()
68
69         CONFIG_WIDGETS['cpu_depth_carousel'].kill()
70
71     def toggle_pvc(self, pvc_enabled):

```

```

72         if pvc_enabled:
73             CONFIG_WIDGETS['pvc_button'].set_locked(True)
74             CONFIG_WIDGETS['pvp_button'].set_locked(False)
75         else:
76             CONFIG_WIDGETS['pvp_button'].set_locked(True)
77             CONFIG_WIDGETS['pvc_button'].set_locked(False)
78
79         self._config['CPU_ENABLED'] = pvc_enabled
80
81     if self._config['CPU_ENABLED']:
82         self.create_depth_picker()
83     else:
84         self.remove_depth_picker()
85
86     def set_fen_string(self, new_fen_string):
87         CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
88         self._config['FEN_STRING'] = new_fen_string
89
90         self.set_preset_overlay(new_fen_string)
91
92     try:
93         CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
94         CONFIG_WIDGETS['invalid_fen_string'].kill()
95
96         if new_fen_string[-1].lower() == 'r':
97             self.set_active_colour(Colour.RED)
98         else:
99             self.set_active_colour(Colour.BLUE)
100
101         self._valid_fen = True
102     except:
103         CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
104         self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])
105
106         window.set_effect(ShaderType.SHAKE)
107         animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE))
108
109         audio.play_sfx(SFX['error_1'])
110         audio.play_sfx(SFX['error_2'])
111
112         self._valid_fen = False
113
114     def get_event(self, event):
115         widget_event = self._widget_group.process_event(event)
116
117         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
118             CONFIG_WIDGETS['help'].kill()
119
120         if widget_event is None:
121             return
122
123         match widget_event.type:
124             case ConfigEventType.GAME_CLICK:
125                 if self._valid_fen:
126                     self.next = 'game'
127                     self.done = True
128
129             case ConfigEventType.MENU_CLICK:
130                 self.next = 'menu'
131                 self.done = True
132

```

```

133         case ConfigEventType.TIME_CLICK:
134             self._config['TIME_ENABLED'] = not(widget_event.data)
135             CONFIG_WIDGETS['timer_button'].set_next_icon()
136
137         case ConfigEventType.PVP_CLICK:
138             self.toggle_pvc(False)
139
140         case ConfigEventType.PVC_CLICK:
141             self.toggle_pvc(True)
142
143         case ConfigEventType.FEN_STRING_TYPE:
144             self.set_fen_string(widget_event.text)
145
146         case ConfigEventType.TIME_TYPE:
147             if widget_event.text == '':
148                 self._config['TIME'] = 5
149             else:
150                 self._config['TIME'] = float(widget_event.text)
151
152         case ConfigEventType.CPU_DEPTH_CLICK:
153             self._config['CPU_DEPTH'] = int(widget_event.data)
154
155         case ConfigEventType.PRESET_CLICK:
156             self.set_fen_string(widget_event.fen_string)
157
158         case ConfigEventType.SETUP_CLICK:
159             self.next = 'editor'
160             self.done = True
161
162         case ConfigEventType.COLOUR_CLICK:
163             self.set_active_colour(widget_event.data.get_flipped_colour())
164
165         case ConfigEventType.HELP_CLICK:
166             self._widget_group.add(CONFIG_WIDGETS['help'])
167             self._widget_group.handle_resize(window.size)
168
169     def set_preset_overlay(self, fen_string):
170         fen_string_widget_map = {
171             'sc3ncfcnccb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
172             PdNaFaNa3Sa b': 'preset_1',
173             'sc3ncfcnccra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3/10/2
174             RaNaFaNa3Sa b': 'preset_2',
175             'sc3pcnccb3/5fc4/pa3pcnccra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3RaNaPa3Pc/4Fa5
176             /3PdNaPa3Sa b': 'preset_3'
177         }
178
179         if fen_string in fen_string_widget_map:
180             self._selected_preset = CONFIG_WIDGETS[fen_string_widget_map[fen_string]]
181         else:
182             self._selected_preset = None
183
184     def set_active_colour(self, colour):
185         if self._config['COLOUR'] != colour:
186             CONFIG_WIDGETS['to_move_button'].set_next_icon()
187
188         self._config['COLOUR'] = colour
189
190         if colour == Colour.BLUE:
191             CONFIG_WIDGETS['to_move_text'].set_text('BLUE TO MOVE')
192         elif colour == Colour.RED:
193             CONFIG_WIDGETS['to_move_text'].set_text('RED TO MOVE')

```

```

191     if self._valid_fen:
192         self._config['FEN_STRING'] = self._config['FEN_STRING'][:-1] + colour.
193         name[0].lower()
194         CONFIG_WIDGETS['fen_string_input'].set_text(self._config['FEN_STRING']
195     ])
196
197     def draw(self):
198         self._widget_group.draw()
199
200         if self._selected_preset:
201             pygame.draw.rect(window.screen, theme['borderPrimary'], (*self.
202             _selected_preset.position, *self._selected_preset.size), width=int(theme['
203             borderWidth']))
204
205     def update(self, **kwargs):
206         self._widget_group.update()
207         super().update(**kwargs)

```

### B.14.2 default\_config.py

```

1 from data.utils.enums import Colour
2
3 default_config = {
4     'FEN_STRING': 'sc3ncfcn cpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
5     Pa2/2PdNaFaNa3Sa b',
6     'COLOUR': Colour.BLUE,
7     'TIME_ENABLED': True,
8     'CPU_ENABLED': False,
9     'CPU_DEPTH': 2,
10    'TIME': 5,
11 }

```

### B.14.3 widget\_dict.py

```

1 from data.widgets import *
2 from data.states.config.default_config import default_config
3 from data.helpers.asset_helpers import get_highlighted_icon
4 from data.components.custom_event import CustomEvent
5 from data.utils.event_types import ConfigEventType
6 from data.utils.assets import GRAPHICS
7 from data.managers.theme import theme
8 from data.utils.enums import Colour
9
10 def float_validator(num_string):
11     try:
12         float(num_string)
13         return True
14     except:
15         return False
16
17 if default_config['CPU_ENABLED']:
18     pvp_icons = {False: GRAPHICS['swords'], True: GRAPHICS['swords']}
19     pvc_icons = {True: GRAPHICS['robot'], False: GRAPHICS['robot']}
20     pvc_locked = True
21     pvp_locked = False
22 else:
23     pvp_icons = {True: GRAPHICS['swords'], False: GRAPHICS['swords']}
24     pvc_icons = {False: GRAPHICS['robot'], True: GRAPHICS['robot']}
25     pvc_locked = False
26     pvp_locked = True

```

```

27
28 if default_config['TIME_ENABLED']:
29     time_enabled_icons = {True: GRAPHICS['timer'], False: get_highlighted_icon(
30         GRAPHICS['timer'])}
31 else:
32     time_enabled_icons = {False: get_highlighted_icon(GRAPHICS['timer']), True:
33         GRAPHICS['timer']}
34
35 if default_config['COLOUR'] == Colour.BLUE:
36     colour_icons = {Colour.BLUE: GRAPHICS['pharaoh_0_a'], Colour.RED: GRAPHICS['
37         pharaoh_1_a']}
38 else:
39     colour_icons = {Colour.RED: GRAPHICS['pharaoh_1_a'], Colour.BLUE: GRAPHICS['
40         pharaoh_0_a']}
41
42 preview_container = Rectangle(
43     relative_position=(-0.15, 0),
44     relative_size=(0.65, 0.9),
45     anchor_x='center',
46     anchor_y='center',
47 )
48
49 config_container = Rectangle(
50     relative_position=(0.325, 0),
51     relative_size=(0.3, 0.9),
52     anchor_x='center',
53     anchor_y='center',
54 )
55
56 to_move_container = Rectangle(
57     parent=config_container,
58     relative_size=(0.9, 0.15),
59     relative_position=(0, 0.1),
60     anchor_x='center'
61 )
62
63 board_thumbnail = BoardThumbnail(
64     parent=preview_container,
65     relative_position=(0, 0),
66     relative_width=0.7,
67     scale_mode='width',
68     anchor_x='right',
69 )
70
71 top_right_container = Rectangle(
72     relative_position=(0, 0),
73     relative_size=(0.15, 0.075),
74     fixed_position=(5, 5),
75     anchor_x='right',
76     scale_mode='height'
77 )
78
79 CONFIG_WIDGETS = {
80     'help':
81     Icon(
82         relative_position=(0, 0),
83         relative_size=(1.02, 1.02),
84         icon=GRAPHICS['config_help'],
85         anchor_x='center',
86         anchor_y='center',
87         border_width=0,
88         fill_colour=(0, 0, 0, 0)
89 }

```

```

85     ),
86     'default': [
87         preview_container,
88         config_container,
89         to_move_container,
90         top_right_container,
91         ReactiveIconButton(
92             parent=top_right_container,
93             relative_position=(0, 0),
94             relative_size=(1, 1),
95             anchor_x='right',
96             scale_mode='height',
97             base_icon=GRAPHICS['home_base'],
98             hover_icon=GRAPHICS['home_hover'],
99             press_icon=GRAPHICS['home_press'],
100            event=CustomEvent(ConfigEventType.MENU_CLICK)
101        ),
102        ReactiveIconButton(
103            parent=top_right_container,
104            relative_position=(0, 0),
105            relative_size=(1, 1),
106            scale_mode='height',
107            base_icon=GRAPHICS['help_base'],
108            hover_icon=GRAPHICS['help_hover'],
109            press_icon=GRAPHICS['help_press'],
110            event=CustomEvent(ConfigEventType.HELP_CLICK)
111        ),
112        TextInput(
113            parent=config_container,
114            relative_position=(0.3, 0.3),
115            relative_size=(0.65, 0.15),
116            fit_vertical=True,
117            placeholder='TIME CONTROL (DEFAULT 5)',
118            default=str(default_config['TIME']),
119            border_width=5,
120            margin=20,
121            validator=float_validator,
122            event=CustomEvent(ConfigEventType.TIME_TYPE)
123        ),
124        Text(
125            parent=config_container,
126            fit_vertical=False,
127            relative_position=(0.75, 0.3),
128            relative_size=(0.2, 0.15),
129            text='MINS',
130            border_width=0,
131            fill_colour=(0, 0, 0, 0)
132        ),
133        TextButton(
134            parent=preview_container,
135            relative_position=(0.3, 0),
136            relative_size=(0.15, 0.15),
137            text='CUSTOM',
138            anchor_y='bottom',
139            fit_vertical=False,
140            margin=10,
141            event=CustomEvent(ConfigEventType.SETUP_CLICK)
142        )
143    ],
144    'board_thumbnail':
145        board_thumbnail,
146    'fen_string_input':

```

```

147     TextInput(
148         parent=preview_container,
149         relative_position=(0, 0),
150         relative_size=(0.55, 0.15),
151         fit_vertical=False,
152         placeholder='ENTER FEN STRING',
153         default='sc3ncfcn cpb2/2pc7/3Pd7/pa1Pc1r b1pb1Pd/pb1Pd1RaB1pa1Pc/6pb3/7
Pa2/2PdNaFaNa3Sa b',
154         border_width=5,
155         anchor_y='bottom',
156         anchor_x='right',
157         margin=20,
158         event=CustomEvent(ConfigEventType.FEN_STRING_TYPE)
159     ),
160     'start_button':
161     TextButton(
162         parent=config_container,
163         relative_position=(0, 0),
164         relative_size=(0.9, 0.3),
165         anchor_y='bottom',
166         anchor_x='center',
167         text='START NEW GAME',
168         strength=0.1,
169         text_colour=theme['textSecondary'],
170         margin=20,
171         fit_vertical=False,
172         event=CustomEvent(ConfigEventType.GAME_CLICK)
173     ),
174     'timer_button':
175     MultipleIconButton(
176         parent=config_container,
177         scale_mode='height',
178         relative_position=(0.05, 0.3),
179         relative_size=(0.15, 0.15),
180         margin=10,
181         border_width=5,
182         border_radius=5,
183         icons_dict=time_enabled_icons,
184         event=CustomEvent(ConfigEventType.TIME_CLICK)
185     ),
186     'pvp_button':
187     MultipleIconButton(
188         parent=config_container,
189         relative_position=(-0.225, 0.5),
190         relative_size=(0.45, 0.15),
191         margin=15,
192         anchor_x='center',
193         icons_dict=pvp_icons,
194         stretch=False,
195         event=CustomEvent(ConfigEventType.PVP_CLICK)
196     ),
197     'pvc_button':
198     MultipleIconButton(
199         parent=config_container,
200         relative_position=(0.225, 0.5),
201         relative_size=(0.45, 0.15),
202         anchor_x='center',
203         margin=15,
204         icons_dict=pvc_icons,
205         stretch=False,
206         event=CustomEvent(ConfigEventType.PVC_CLICK)
207     ),

```

```

208     'invalid_fen_string':
209     Text(
210         parent=board_thumbnail,
211         relative_position=(0, 0),
212         relative_size=(0.9, 0.1),
213         fit_vertical=False,
214         anchor_x='center',
215         anchor_y='center',
216         text='INVALID FEN STRING!',
217         margin=10,
218         fill_colour=theme['fillError'],
219         text_colour=theme['textError'],
220     ),
221     'preset_1':
222     BoardThumbnailButton(
223         parent=preview_container,
224         relative_width=0.25,
225         relative_position=(0, 0),
226         scale_mode='width',
227         fen_string="sc3ncfcnspb2/2pc7/3Pd6/pa1Pcirbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3
/7Pa2/2PdNaFaNa3Sa b",
228         event=CustomEvent(ConfigEventType.PRESET_CLICK)
229     ),
230     'preset_2':
231     BoardThumbnailButton(
232         parent=preview_container,
233         relative_width=0.25,
234         relative_position=(0, 0.35),
235         scale_mode='width',
236         fen_string="sc3ncfcnra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3
/10/2RaNaFaNa3Sa b",
237         event=CustomEvent(ConfigEventType.PRESET_CLICK)
238     ),
239     'preset_3':
240     BoardThumbnailButton(
241         parent=preview_container,
242         relative_width=0.25,
243         relative_position=(0, 0.7),
244         scale_mode='width',
245         fen_string="sc3pcnspb3/5fc4/pa3pcnra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3
RaNaPa3Pc/4Fa5/3PdNaPa3Sa b",
246         event=CustomEvent(ConfigEventType.PRESET_CLICK)
247     ),
248     'to_move_button':
249     MultipleIconButton(
250         parent=to_move_container,
251         scale_mode='height',
252         relative_position=(0, 0),
253         relative_size=(1, 1),
254         icons_dict=colour_icons,
255         anchor_x='left',
256         event=CustomEvent(ConfigEventType.COLOUR_CLICK)
257     ),
258     'to_move_text':
259     Text(
260         parent=to_move_container,
261         relative_position=(0, 0),
262         relative_size=(0.75, 1),
263         fit_vertical=False,
264         text='TO MOVE',
265         anchor_x='right'
266     ),

```

```

267     'cpu_depth_carousel':
268     Carousel(
269         parent=config_container,
270         relative_position=(0, 0.65),
271         event=CustomEvent(ConfigEventType.CPU_DEPTH_CLICK),
272         anchor_x='center',
273         border_width=0,
274         fill_colour=(0, 0, 0, 0),
275         widgets_dict={
276             2: Text(
277                 parent=config_container,
278                 relative_position=(0, 0),
279                 relative_size=(0.8, 0.075),
280                 text="EASY",
281                 margin=0,
282                 border_width=0,
283                 fill_colour=(0, 0, 0, 0)
284             ),
285             3: Text(
286                 parent=config_container,
287                 relative_position=(0, 0),
288                 relative_size=(0.8, 0.075),
289                 text="MEDIUM",
290                 margin=0,
291                 border_width=0,
292                 fill_colour=(0, 0, 0, 0)
293             ),
294             4: Text(
295                 parent=config_container,
296                 relative_position=(0, 0),
297                 relative_size=(0.8, 0.075),
298                 text="HARD",
299                 margin=0,
300                 border_width=0,
301                 fill_colour=(0, 0, 0, 0)
302             ),
303         }
304     )
305 }
```

## B.15 data\states\editor

### B.15.1 editor.py

```

1 import pygame
2 import pyperclip
3 from data.states.game.components.bitboard_collection import BitboardCollection
4 from data.utils.enums import Colour, RotationDirection, Piece, Rotation
5 from data.states.game.components.fen_parser import encode_fen_string
6 from data.states.game.components.overlay_draw import OverlayDraw
7 from data.states.game.components.piece_group import PieceGroup
8 from data.helpers.bitboard_helpers import coords_to_bitboard
9 from data.helpers.board_helpers import screen_pos_to_coords
10 from data.states.game.components.father import DragAndDrop
11 from data.states.editor.widget_dict import EDITOR_WIDGETS
12 from data.utils.event_types import EditorEventType
13 from data.managers.logs import initialise_logger
14 from data.managers.window import window
15 from data.control import _State
16
17 logger = initialise_logger(__name__)
```

```

18
19 class Editor(_State):
20     def __init__(self):
21         super().__init__()
22
23         self._bitboards = None
24         self._piece_group = None
25         self._selected_coords = None
26         self._selected_tool = None
27         self._selected_tool.colour = None
28         self._initial_fen_string = None
29         self._starting.colour = None
30
31         self._drag_and_drop = None
32         self._overlay_draw = None
33
34     def cleanup(self):
35         super().cleanup()
36
37         self.deselect_tool()
38
39         return encode_fen_string(self._bitboards)
40
41     def startup(self, persist):
42         super().startup(EDITOR_WIDGETS)
43         EDITOR_WIDGETS['help'].kill()
44
45         self._drag_and_drop = DragAndDrop(EDITOR_WIDGETS['chessboard'].position,
46                                         EDITOR_WIDGETS['chessboard'].size)
47         self._overlay_draw = OverlayDraw(EDITOR_WIDGETS['chessboard'].position,
48                                         EDITOR_WIDGETS['chessboard'].size)
49         self._bitboards = BitboardCollection(persist['FEN_STRING'])
50         self._piece_group = PieceGroup()
51
52         self._selected_coords = None
53         self._selected_tool = None
54         self._selected_tool.colour = None
55         self._initial_fen_string = persist['FEN_STRING']
56         self._starting.colour = Colour.BLUE
57
58         self.refresh_pieces()
59         self.set_starting.colour(Colour.BLUE if persist['FEN_STRING'][-1].lower()
60 == 'b' else Colour.RED)
61         self.draw()
62
63     @property
64     def selected_coords(self):
65         return self._selected_coords
66
67     @selected_coords.setter
68     def selected_coords(self, new_coords):
69         self._overlay_draw.set_selected_coords(new_coords)
70         self._selected_coords = new_coords
71
72     def get_event(self, event):
73         widget_event = self._widget_group.process_event(event)
74
75         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
76             EDITOR_WIDGETS['help'].kill()
77
78         if event.type == pygame.MOUSEBUTTONDOWN:
79             clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['

```

```

    chessboard'].position, EDITOR_WIDGETS['chessboard'].size)

77     if clicked_coords:
78         self.selected_coords = clicked_coords
79
80     if self._selected_tool is None:
81         return
82
83     if self._selected_tool == 'MOVE':
84         self.set_dragged_piece(clicked_coords)
85
86     elif self._selected_tool == 'ERASE':
87         self.remove_piece()
88     else:
89         self.set_piece(self._selected_tool, self._selected_tool.colour
90 , Rotation.UP)
91
92     return
93
94     if event.type == pygame.MOUSEBUTTONUP:
95         clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
96
97     if self._drag_and_drop.dragged_sprite:
98         self.remove_dragged_piece(clicked_coords)
99     return
100
101     if widget_event is None:
102         if event.type == pygame.MOUSEBUTTONDOWN and self._widget_group.
on_widget(event.pos) is False:
103             self.selected_coords = None
104
105         return
106
107     match widget_event.type:
108         case None:
109             return
110
111         case EditorEventType.MENU_CLICK:
112             self.next = 'menu'
113             self.done = True
114
115         case EditorEventType.PICK_PIECE_CLICK:
116             if widget_event.piece == self._selected_tool and widget_event.
active.colour == self._selected_tool.colour:
117                 self.deselect_tool()
118             else:
119                 self.select_tool(widget_event.piece, widget_event.
active.colour)
120
121         case EditorEventType.ROTATE_PIECE_CLICK:
122             self.rotate_piece(widget_event.rotation_direction)
123
124         case EditorEventType.EMPTY_CLICK:
125             self._bitboards = BitboardCollection(fen_string='sc9
/10/10/10/10/10/10/9Sa b')
126             self.refresh_pieces()
127
128         case EditorEventType.RESET_CLICK:
129             self.reset_board()
130
131         case EditorEventType.COPY_CLICK:

```

```

132         logger.info(f'COPYING TO CLIPBOARD: {encode_fen_string(self._bitboards)})')
133         pyperclip.copy(encode_fen_string(self._bitboards))
134
135     case EditorEventType.BLUE_START_CLICK:
136         self.set_starting_colour(Colour.BLUE)
137
138     case EditorEventType.RED_START_CLICK:
139         self.set_starting_colour(Colour.RED)
140
141     case EditorEventType.START_CLICK:
142         self.next = 'config'
143         self.done = True
144
145     case EditorEventType.CONFIG_CLICK:
146         self.reset_board()
147         self.next = 'config'
148         self.done = True
149
150     case EditorEventType.ERASE_CLICK:
151         if self._selected_tool == 'ERASE':
152             self.deselect_tool()
153         else:
154             self.select_tool('ERASE', None)
155
156     case EditorEventType.MOVE_CLICK:
157         if self._selected_tool == 'MOVE':
158             self.deselect_tool()
159         else:
160             self.select_tool('MOVE', None)
161
162     case EditorEventType.HELP_CLICK:
163         self._widget_group.add(EDITOR_WIDGETS['help'])
164         self._widget_group.handle_resize(window.size)
165
166 def reset_board(self):
167     self._bitboards = BitboardCollection(self._initial_fen_string)
168     self.refresh_pieces()
169
170 def refresh_pieces(self):
171     self._piece_group.initialise_pieces(self._bitboards.convert_to_piece_list(),
172                                         EDITOR_WIDGETS['chessboard'].position,
173                                         EDITOR_WIDGETS['chessboard'].size)
174
175     def set_starting_colour(self, new_colour):
176         if new_colour == Colour.BLUE:
177             EDITOR_WIDGETS['blue_start_button'].set_locked(True)
178             EDITOR_WIDGETS['red_start_button'].set_locked(False)
179         elif new_colour == Colour.RED:
180             EDITOR_WIDGETS['blue_start_button'].set_locked(False)
181             EDITOR_WIDGETS['red_start_button'].set_locked(True)
182
183         if new_colour != self._starting_colour:
184             EDITOR_WIDGETS['blue_start_button'].set_next_icon()
185             EDITOR_WIDGETS['red_start_button'].set_next_icon()
186
187         self._starting_colour = new_colour
188         self._bitboards.active_colour = new_colour
189
190     def set_dragged_piece(self, coords):
191         bitboard_under_mouse = coords_to_bitboard(coords)
192         dragged_piece = self._bitboards.get_piece_on(bitboard_under_mouse, Colour.BLUE) or self._bitboards.get_piece_on(bitboard_under_mouse, Colour.RED)

```

```

191
192     if dragged_piece is None:
193         return
194
195     dragged_colour = self._bitboards.get_colour_on(bitboard_under_mouse)
196     dragged_rotation = self._bitboards.get_rotation_on(bitboard_under_mouse)
197
198     self._drag_and_drop.set_dragged_piece(dragged_piece, dragged_colour,
199                                             dragged_rotation)
200     self._overlay_draw.set_hover_limit(False)
201
202     def remove_dragged_piece(self, coords):
203         piece, colour, rotation = self._drag_and_drop.get_dragged_info()
204
205         if coords and coords != self._selected_coords and piece != Piece.SPHINX:
206             self.remove_piece()
207             self.selected_coords = coords
208             self.set_piece(piece, colour, rotation)
209             self.selected_coords = None
210
211             self._drag_and_drop.remove_dragged_piece()
212             self._overlay_draw.set_hover_limit(True)
213
214     def set_piece(self, piece, colour, rotation):
215         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
216         selected_coords == (9, 0):
217             return
218
219             self.remove_piece()
220
221             selected_bitboard = coords_to_bitboard(self.selected_coords)
222             self._bitboards.set_square(selected_bitboard, piece, colour)
223             self._bitboards.set_rotation(selected_bitboard, rotation)
224
225             self.refresh_pieces()
226
227     def remove_piece(self):
228         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
229         selected_coords == (9, 0):
230             return
231
232             selected_bitboard = coords_to_bitboard(self.selected_coords)
233             self._bitboards.clear_square(selected_bitboard, Colour.BLUE)
234             self._bitboards.clear_square(selected_bitboard, Colour.RED)
235             self._bitboards.clear_rotation(selected_bitboard)
236
237             self.refresh_pieces()
238
239     def rotate_piece(self, rotation_direction):
240         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
241         selected_coords == (9, 0):
242             return
243
244             selected_bitboard = coords_to_bitboard(self.selected_coords)
245
246             if self._bitboards.get_piece_on(selected_bitboard, Colour.BLUE) is None
247             and self._bitboards.get_piece_on(selected_bitboard, Colour.RED) is None:
248                 return
249
250             current_rotation = self._bitboards.get_rotation_on(selected_bitboard)
251
252             if rotation_direction == RotationDirection.CLOCKWISE:

```

```

248         self._bitboards.update_rotation(selected_bitboard, selected_bitboard,
249                                         current_rotation.get_clockwise())
250         elif rotation_direction == RotationDirection.ANTICLOCKWISE:
251             self._bitboards.update_rotation(selected_bitboard, selected_bitboard,
252                                         current_rotation.get_anticlockwise())
253
254     self.refresh_pieces()
255
256     def select_tool(self, piece, colour):
257         dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: 'red_piece_buttons' }
258
259         self.deselect_tool()
260
261         if piece == 'ERASE':
262             EDITOR_WIDGETS['erase_button'].set_locked(True)
263             EDITOR_WIDGETS['erase_button'].set_next_icon()
264         elif piece == 'MOVE':
265             EDITOR_WIDGETS['move_button'].set_locked(True)
266             EDITOR_WIDGETS['move_button'].set_next_icon()
267         else:
268             EDITOR_WIDGETS[dict_name_map[colour]][piece].set_locked(True)
269             EDITOR_WIDGETS[dict_name_map[colour]][piece].set_next_icon()
270
271         self._selected_tool = piece
272         self._selected_tool_colour = colour
273
274     def deselect_tool(self):
275         dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: 'red_piece_buttons' }
276
277         if self._selected_tool:
278             if self._selected_tool == 'ERASE':
279                 EDITOR_WIDGETS['erase_button'].set_locked(False)
280                 EDITOR_WIDGETS['erase_button'].set_next_icon()
281             elif self._selected_tool == 'MOVE':
282                 EDITOR_WIDGETS['move_button'].set_locked(False)
283                 EDITOR_WIDGETS['move_button'].set_next_icon()
284             else:
285                 EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self._selected_tool].set_locked(False)
286                 EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self._selected_tool].set_next_icon()
287
288         self._selected_tool = None
289         self._selected_tool_colour = None
290
291     def handle_resize(self):
292         super().handle_resize()
293         self._piece_group.handle_resize(EDITOR_WIDGETS['chessboard'].position,
294                                         EDITOR_WIDGETS['chessboard'].size)
295         self._drag_and_drop.handle_resize(EDITOR_WIDGETS['chessboard'].position,
296                                         EDITOR_WIDGETS['chessboard'].size)
297         self._overlay_draw.handle_resize(EDITOR_WIDGETS['chessboard'].position,
298                                         EDITOR_WIDGETS['chessboard'].size)
299
300     def draw(self):
301         self._widget_group.draw()
302         self._overlay_draw.draw(window.screen)
303         self._piece_group.draw(window.screen)
304         self._drag_and_drop.draw(window.screen)

```

### B.15.2 widget\_dict.py

```
1 from data.utils.enums import Piece, Colour, RotationDirection
2 from data.helpers.asset_helpers import get_highlighted_icon
3 from data.components.custom_event import CustomEvent
4 from data.utils.constants import BLUE_BUTTON_COLOURS
5 from data.utils.event_types import EditorEventType
6 from data.utils.assets import GRAPHICS
7 from data.widgets import *
8
9 blue_pieces_container = Rectangle(
10     relative_position=(0.25, 0),
11     relative_size=(0.13, 0.65),
12     scale_mode='height',
13     anchor_y='center',
14     anchor_x='center'
15 )
16
17 red_pieces_container = Rectangle(
18     relative_position=(-0.25, 0),
19     relative_size=(0.13, 0.65),
20     scale_mode='height',
21     anchor_y='center',
22     anchor_x='center'
23 )
24
25 bottom_actions_container = Rectangle(
26     relative_position=(0, 0.05),
27     relative_size=(0.4, 0.1),
28     anchor_x='center',
29     anchor_y='bottom'
30 )
31
32 top_actions_container = Rectangle(
33     relative_position=(0, 0.05),
34     relative_size=(0.3, 0.1),
35     anchor_x='center',
36     scale_mode='height'
37 )
38
39 top_right_container = Rectangle(
40     relative_position=(0, 0),
41     relative_size=(0.15, 0.075),
42     fixed_position=(5, 5),
43     anchor_x='right',
44     scale_mode='height'
45 )
46
47 EDITOR_WIDGETS = {
48     'help':
49         Icon(
50             relative_position=(0, 0),
51             relative_size=(1.02, 1.02),
52             icon=GRAPHICS['editor_help'],
53             anchor_x='center',
54             anchor_y='center',
55             border_width=0,
56             fill_colour=(0, 0, 0, 0)
57         ),
58     'default': [
59         red_pieces_container,
```

```

60         blue_pieces_container,
61         bottom_actions_container,
62         top_actions_container,
63         top_right_container,
64         ReactiveIconButton(
65             parent=top_right_container,
66             relative_position=(0, 0),
67             relative_size=(1, 1),
68             anchor_x='right',
69             scale_mode='height',
70             base_icon=GRAPHICS['home_base'],
71             hover_icon=GRAPHICS['home_hover'],
72             press_icon=GRAPHICS['home_press'],
73             event=CustomEvent(EditorEventType.MENU_CLICK)
74         ),
75         ReactiveIconButton(
76             parent=top_right_container,
77             relative_position=(0, 0),
78             relative_size=(1, 1),
79             scale_mode='height',
80             base_icon=GRAPHICS['help_base'],
81             hover_icon=GRAPHICS['help_hover'],
82             press_icon=GRAPHICS['help_press'],
83             event=CustomEvent(EditorEventType.HELP_CLICK)
84         ),
85         ReactiveIconButton(
86             parent=bottom_actions_container,
87             relative_position=(0.06, 0),
88             relative_size=(1, 1),
89             anchor_x='center',
90             scale_mode='height',
91             base_icon=GRAPHICS['clockwise_arrow_base'],
92             hover_icon=GRAPHICS['clockwise_arrow_hover'],
93             press_icon=GRAPHICS['clockwise_arrow_press'],
94             event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
95             rotation_direction=RotationDirection.CLOCKWISE)
96         ),
97         ReactiveIconButton(
98             parent=bottom_actions_container,
99             relative_position=(-0.06, 0),
100            relative_size=(1, 1),
101            anchor_x='center',
102            scale_mode='height',
103            base_icon=GRAPHICS['anticlockwise_arrow_base'],
104            hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
105            press_icon=GRAPHICS['anticlockwise_arrow_press'],
106            event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
107            rotation_direction=RotationDirection.ANTICLOCKWISE)
108        ),
109        ReactiveIconButton(
110            parent=top_actions_container,
111            relative_position=(0, 0),
112            relative_size=(1, 1),
113            scale_mode='height',
114            anchor_x='right',
115            base_icon=GRAPHICS['copy_base'],
116            hover_icon=GRAPHICS['copy_hover'],
117            press_icon=GRAPHICS['copy_press'],
118            event=CustomEvent(EditorEventType.COPY_CLICK),
119            parent=top_actions_container,

```

```

120         relative_position=(0, 0),
121         relative_size=(1, 1),
122         scale_mode='height',
123         base_icon=GRAPHICS['delete_base'],
124         hover_icon=GRAPHICS['delete_hover'],
125         press_icon=GRAPHICS['delete_press'],
126         event=CustomEvent(EditorEventType.EMPTY_CLICK),
127     ),
128     ReactiveIconButton(
129         parent=top_actions_container,
130         relative_position=(0, 0),
131         relative_size=(1, 1),
132         scale_mode='height',
133         anchor_x='center',
134         base_icon=GRAPHICS['discard_arrow_base'],
135         hover_icon=GRAPHICS['discard_arrow_hover'],
136         press_icon=GRAPHICS['discard_arrow_press'],
137         event=CustomEvent(EditorEventType.RESET_CLICK),
138     ),
139     ReactiveIconButton(
140         relative_position=(0, 0),
141         fixed_position=(10, 0),
142         relative_size=(0.1, 0.1),
143         anchor_x='right',
144         anchor_y='center',
145         scale_mode='height',
146         base_icon=GRAPHICS['play_arrow_base'],
147         hover_icon=GRAPHICS['play_arrow_hover'],
148         press_icon=GRAPHICS['play_arrow_press'],
149         event=CustomEvent(EditorEventType.START_CLICK),
150     ),
151     ReactiveIconButton(
152         relative_position=(0, 0),
153         fixed_position=(10, 0),
154         relative_size=(0.1, 0.1),
155         anchor_y='center',
156         scale_mode='height',
157         base_icon=GRAPHICS['return_arrow_base'],
158         hover_icon=GRAPHICS['return_arrow_hover'],
159         press_icon=GRAPHICS['return_arrow_press'],
160         event=CustomEvent(EditorEventType.CONFIG_CLICK),
161     )
162 ],
163 'blue_piece_buttons': {},
164 'red_piece_buttons': {},
165 'erase_button':
166     MultipleIconButton(
167         parent=red_pieces_container,
168         relative_position=(0, 0),
169         relative_size=(0.2, 0.2),
170         scale_mode='height',
171         margin=10,
172         icons_dict={True: GRAPHICS['eraser'], False: get_highlighted_icon(GRAPHICS['eraser'])},
173         event=CustomEvent(EditorEventType.ERASE_CLICK),
174     ),
175     'move_button':
176     MultipleIconButton(
177         parent=blue_pieces_container,
178         relative_position=(0, 0),
179         relative_size=(0.2, 0.2),
180         scale_mode='height',

```

```

181     box_colours=BLUE_BUTTON_COLOURS,
182     icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
183 ['finger'])}),
184     event=CustomEvent(EditorEventType.MOVE_CLICK),
185 ),
186     'chessboard':
187     Chessboard(
188         relative_position=(0, 0),
189         relative_width=0.4,
190         scale_mode='width',
191         anchor_x='center',
192         anchor_y='center'
193     ),
194     'blue_start_button':
195     MultipleIconButton(
196         parent=bottom_actions_container,
197         relative_position=(0, 0),
198         relative_size=(1, 1),
199         scale_mode='height',
200         anchor_x='right',
201         box_colours=BLUE_BUTTON_COLOURS,
202         icons_dict={False: get_highlighted_icon(GRAPHICS['pharaoh_0_a']), True:
203 GRAPHICS['pharaoh_0_a']},
204         event=CustomEvent(EditorEventType.BLUE_START_CLICK)
205     ),
206     'red_start_button':
207     MultipleIconButton(
208         parent=bottom_actions_container,
209         relative_position=(0, 0),
210         relative_size=(1, 1),
211         scale_mode='height',
212         icons_dict={True: GRAPHICS['pharaoh_1_a'], False: get_highlighted_icon(
213 GRAPHICS['pharaoh_1_a'])},
214         event=CustomEvent(EditorEventType.RED_START_CLICK)
215     )
216 }
217
218 for index, piece in enumerate([piece for piece in Piece if piece != Piece.SPHINX]):
219     :
220     blue_icon = GRAPHICS[f'{piece.name.lower()}_0_a']
221     dimmed_blue_icon = get_highlighted_icon(blue_icon)
222
223     EDITOR_WIDGETS['blue_piece_buttons'][piece] = MultipleIconButton(
224         parent=blue_pieces_container,
225         relative_position=(0, (index + 1) / 5),
226         relative_size=(0.2, 0.2),
227         scale_mode='height',
228         box_colours=BLUE_BUTTON_COLOURS,
229         icons_dict={True: blue_icon, False: dimmed_blue_icon},
230         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
231 active_colour=Colour.BLUE)
232     )
233
234     red_icon = GRAPHICS[f'{piece.name.lower()}_1_a']
235
236     dimmed_red_icon = get_highlighted_icon(red_icon)
237
238     EDITOR_WIDGETS['red_piece_buttons'][piece] = MultipleIconButton(
239         parent=red_pieces_container,
240         relative_position=(0, (index + 1) / 5),
241         relative_size=(0.2, 0.2),
242         scale_mode='height',
243

```

```

238         icons_dict={True: red_icon, False: dimmed_red_icon},
239         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
240             active_colour=Colour.RED)

```

## B.16 data\states\game

### B.16.1 game.py

```

1 import pygame
2 from functools import partial
3 from data.states.game.mvc.game_controller import GameController
4 from data.helpers.database_helpers import insert_into_games
5 from data.states.game.mvc.game_model import GameModel
6 from data.states.game.mvc.pause_view import PauseView
7 from data.states.game.mvc.game_view import GameView
8 from data.states.game.mvc.win_view import WinView
9 from data.components.game_entry import GameEntry
10 from data.managers.logs import initialise_logger
11 from data.managers.window import window
12 from data.managers.audio import audio
13 from data.utils.constants import ShaderType
14 from data.utils.assets import MUSIC, SFX
15 from data.control import _State
16
17 logger = initialise_logger(__name__)
18
19 class Game(_State):
20     def __init__(self):
21         super().__init__()
22
23     def cleanup(self):
24         super().cleanup()
25
26         window.clear_apply_arguments(ShaderType.BLOOM)
27         window.clear_effect(ShaderType.RAYS)
28
29         game_entry = GameEntry(self.model.states, final_fen_string=self.model.
30             get_fen_string())
31         inserted_game = insert_into_games(game_entry.convert_to_row())
32
33         return inserted_game
34
35     def switch_to_menu(self):
36         self.next = 'menu'
37         self.done = True
38
39     def switch_to_review(self):
40         self.next = 'review'
41         self.done = True
42
43     def startup(self, persist):
44         music = MUSIC[['cpu_easy', 'cpu_medium', 'cpu_hard'][persist['CPU_DEPTH']
45 - 2]] if persist['CPU_ENABLED'] else MUSIC['pvp']
46         super().startup(music=music)
47
48         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
49             BACKGROUND_LASERS)
50         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
51             Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
52         binded_startup = partial(self.startup, persist)

```

```

49
50         self.model = GameModel(persist)
51         self.view = GameView(self.model)
52         self.pause_view = PauseView(self.model)
53         self.win_view = WinView(self.model)
54         self.controller = GameController(self.model, self.view, self.win_view,
55                                         self.pause_view, self.switch_to_menu, self.switch_to_review, binded_startup)
56
57         self.view.draw()
58
59         audio.play_sfx(SFX['game_start_1'])
60         audio.play_sfx(SFX['game_start_2'])
61
62     def get_event(self, event):
63         self.controller.handle_event(event)
64
65     def handle_resize(self):
66         self.view.handle_resize()
67         self.win_view.handle_resize()
68         self.pause_view.handle_resize()
69
70     def draw(self):
71         self.view.draw()
72         self.win_view.draw()
73         self.pause_view.draw()
74
75     def update(self):
76         self.controller.check_cpu()
77         super().update()

```

### B.16.2 widget\_dict.py

```

1 from data.widgets import *
2 from data.utils.enums import RotationDirection, Colour
3 from data.components.custom_event import CustomEvent
4 from data.utils.event_types import GameEventType
5 from data.utils.assets import GRAPHICS
6
7 right_container = Rectangle(
8     relative_position=(0.05, 0),
9     relative_size=(0.2, 0.5),
10    anchor_y='center',
11    anchor_x='right',
12 )
13
14 rotate_container = Rectangle(
15     relative_position=(0, 0.05),
16     relative_size=(0.2, 0.1),
17     anchor_x='center',
18     anchor_y='bottom',
19 )
20
21 move_list = MoveList(
22     parent=right_container,
23     relative_position=(0, 0),
24     relative_width=1,
25     minimum_height=300,
26     move_list=[]
27 )
28
29 resign_button = TextButton(
30     parent=right_container,

```

```

31     relative_position=(0, 0),
32     relative_size=(0.5, 0.2),
33     fit_vertical=False,
34     anchor_y='bottom',
35     text="    Resign",
36     margin=5,
37     event=CustomEvent(GameEventType.RESIGN_CLICK)
38 )
39
40 draw_button = TextButton(
41     parent=right_container,
42     relative_position=(0, 0),
43     relative_size=(0.5, 0.2),
44     fit_vertical=False,
45     anchor_x='right',
46     anchor_y='bottom',
47     text="    Draw",
48     margin=5,
49     event=CustomEvent(GameEventType.DRAW_CLICK)
50 )
51
52 top_right_container = Rectangle(
53     relative_position=(0, 0),
54     relative_size=(0.225, 0.075),
55     fixed_position=(5, 5),
56     anchor_x='right',
57     scale_mode='height'
58 )
59
60 GAME_WIDGETS = {
61     'help':
62     Icon(
63         relative_position=(0, 0),
64         relative_size=(1.02, 1.02),
65         icon=GRAPHICS['game_help'],
66         anchor_x='center',
67         anchor_y='center',
68         border_width=0,
69         fill_colour=(0, 0, 0, 0)
70     ),
71     'tutorial':
72     Icon(
73         relative_position=(0, 0),
74         relative_size=(0.9, 0.9),
75         icon=GRAPHICS['game_tutorial'],
76         anchor_x='center',
77         anchor_y='center',
78     ),
79     'default': [
80         right_container,
81         rotate_container,
82         top_right_container,
83         ReactiveIconButton(
84             parent=top_right_container,
85             relative_position=(0, 0),
86             relative_size=(1, 1),
87             anchor_x='right',
88             scale_mode='height',
89             base_icon=GRAPHICS['home_base'],
90             hover_icon=GRAPHICS['home_hover'],
91             press_icon=GRAPHICS['home_press'],
92             event=CustomEvent(GameEventType.MENU_CLICK)

```

```

93     ),
94     ReactiveIconButton(
95         parent=top_right_container,
96         relative_position=(0, 0),
97         relative_size=(1, 1),
98         scale_mode='height',
99         base_icon=GRAPHICS['tutorial_base'],
100        hover_icon=GRAPHICS['tutorial_hover'],
101        press_icon=GRAPHICS['tutorial_press'],
102        event=CustomEvent(GameEventType.TUTORIAL_CLICK)
103    ),
104    ReactiveIconButton(
105        parent=top_right_container,
106        relative_position=(0.33, 0),
107        relative_size=(1, 1),
108        scale_mode='height',
109        base_icon=GRAPHICS['help_base'],
110        hover_icon=GRAPHICS['help_hover'],
111        press_icon=GRAPHICS['help_press'],
112        event=CustomEvent(GameEventType.HELP_CLICK)
113    ),
114    ReactiveIconButton(
115        parent=rotate_container,
116        relative_position=(0, 0),
117        relative_size=(1, 1),
118        scale_mode='height',
119        anchor_x='right',
120        base_icon=GRAPHICS['clockwise_arrow_base'],
121        hover_icon=GRAPHICS['clockwise_arrow_hover'],
122        press_icon=GRAPHICS['clockwise_arrow_press'],
123        event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
RotationDirection.CLOCKWISE)
124    ),
125    ReactiveIconButton(
126        parent=rotate_container,
127        relative_position=(0, 0),
128        relative_size=(1, 1),
129        scale_mode='height',
130        base_icon=GRAPHICS['anticlockwise_arrow_base'],
131        hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
132        press_icon=GRAPHICS['anticlockwise_arrow_press'],
133        event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
RotationDirection.ANTICLOCKWISE)
134    ),
135    resign_button,
136    draw_button,
137    Icon(
138        parent=resign_button,
139        relative_position=(0, 0),
140        relative_size=(0.75, 0.75),
141        fill_colour=(0, 0, 0, 0),
142        scale_mode='height',
143        anchor_y='center',
144        border_radius=0,
145        border_width=0,
146        margin=5,
147        icon=GRAPHICS['resign']
148    ),
149    Icon(
150        parent=draw_button,
151        relative_position=(0, 0),
152        relative_size=(0.75, 0.75),

```

```

153         fill_colour=(0, 0, 0, 0),
154         scale_mode='height',
155         anchor_y='center',
156         border_radius=0,
157         border_width=0,
158         margin=5,
159         icon=GRAPHICS['draw']
160     ),
161 ],
162 'scroll_area': # REMEMBER SCROLL AREA AFTER CONTAINER FOR RESIZING
163 ScrollArea(
164     parent=right_container,
165     relative_position=(0, 0),
166     relative_size=(1, 0.8),
167     vertical=True,
168     widget=move_list
169 ),
170 'move_list':
171     move_list,
172 'blue_timer':
173 Timer(
174     relative_position=(0.05, 0.05),
175     anchor_y='center',
176     relative_size=(0.1, 0.1),
177     active_colour=Colour.BLUE,
178     event=CustomEvent(GameEventType.TIMER_END),
179 ),
180 'red_timer':
181 Timer(
182     relative_position=(0.05, -0.05),
183     anchor_y='center',
184     relative_size=(0.1, 0.1),
185     active_colour=Colour.RED,
186     event=CustomEvent(GameEventType.TIMER_END),
187 ),
188 'status_text':
189 Text(
190     relative_position=(0, 0.05),
191     relative_size=(0.4, 0.1),
192     anchor_x='center',
193     fit_vertical=False,
194     margin=10,
195     text="g",
196     minimum_width=400
197 ),
198 'chessboard':
199 Chessboard(
200     relative_position=(0, 0),
201     anchor_x='center',
202     anchor_y='center',
203     scale_mode='width',
204     relative_width=0.4
205 ),
206 'blue_piece_display':
207 PieceDisplay(
208     relative_position=(0.05, 0.05),
209     relative_size=(0.2, 0.1),
210     anchor_y='bottom',
211     active_colour=Colour.BLUE
212 ),
213 'red_piece_display':
214 PieceDisplay(

```

```

215         relative_position=(0.05, 0.05),
216         relative_size=(0.2, 0.1),
217         active_colour=Colour.RED
218     )
219 }
220
221 PAUSE_WIDGETS = {
222     'default': [
223         TextButton(
224             relative_position=(0, -0.125),
225             relative_size=(0.3, 0.2),
226             anchor_x='center',
227             anchor_y='center',
228             text='GO TO MENU',
229             fit_vertical=False,
230             event=CustomEvent(GameEventType.MENU_CLICK)
231         ),
232         TextButton(
233             relative_position=(0, 0.125),
234             relative_size=(0.3, 0.2),
235             anchor_x='center',
236             anchor_y='center',
237             text='RESUME GAME',
238             fit_vertical=False,
239             event=CustomEvent(GameEventType.PAUSE_CLICK)
240         )
241     ]
242 }
243
244 win_container = Rectangle(
245     relative_position=(0, 0),
246     relative_size=(0.4, 0.8),
247     scale_mode='height',
248     anchor_x='center',
249     anchor_y='center',
250     fill_colour=(128, 128, 128, 200),
251     visible=True
252 )
253
254 WIN_WIDGETS = {
255     'default': [
256         win_container,
257         TextButton(
258             parent=win_container,
259             relative_position=(0, 0.5),
260             relative_size=(0.8, 0.15),
261             text='GO TO MENU',
262             anchor_x='center',
263             fit_vertical=False,
264             event=CustomEvent(GameEventType.MENU_CLICK)
265         ),
266         TextButton(
267             parent=win_container,
268             relative_position=(0, 0.65),
269             relative_size=(0.8, 0.15),
270             text='REVIEW GAME',
271             anchor_x='center',
272             fit_vertical=False,
273             event=CustomEvent(GameEventType.REVIEW_CLICK)
274         ),
275         TextButton(
276             parent=win_container,

```

```

277         relative_position=(0, 0.8),
278         relative_size=(0.8, 0.15),
279         text='NEW GAME',
280         anchor_x='center',
281         fit_vertical=False,
282         event=CustomEvent(GameEventType.GAME_CLICK)
283     ),
284 ],
285 'blue_won':
286 Icon(
287     parent=win_container,
288     relative_position=(0, 0.05),
289     relative_size=(0.8, 0.3),
290     anchor_x='center',
291     border_width=0,
292     margin=0,
293     icon=GRAPHICS['blue_won'],
294     fill_colour=(0, 0, 0, 0),
295 ),
296 'red_won':
297 Icon(
298     parent=win_container,
299     relative_position=(0, 0.05),
300     relative_size=(0.8, 0.3),
301     anchor_x='center',
302     border_width=0,
303     margin=0,
304     icon=GRAPHICS['red_won'],
305     fill_colour=(0, 0, 0, 0),
306     fit_icon=True,
307 ),
308 'draw_won':
309 Icon(
310     parent=win_container,
311     relative_position=(0, 0.05),
312     relative_size=(0.8, 0.3),
313     anchor_x='center',
314     border_width=0,
315     margin=0,
316     icon=GRAPHICS['draw_won'],
317     fill_colour=(0, 0, 0, 0),
318 ),
319 'by_checkmate':
320 Icon(
321     parent=win_container,
322     relative_position=(0, 0.375),
323     relative_size=(0.8, 0.1),
324     anchor_x='center',
325     border_width=0,
326     margin=0,
327     icon=GRAPHICS['by_checkmate'],
328     fill_colour=(0, 0, 0, 0),
329 ),
330 'by_resignation':
331 Icon(
332     parent=win_container,
333     relative_position=(0, 0.375),
334     relative_size=(0.8, 0.1),
335     anchor_x='center',
336     border_width=0,
337     margin=0,
338     icon=GRAPHICS['by_resignation'],

```

```

339         fill_colour=(0, 0, 0, 0),
340     ),
341     'by_draw':
342     Icon(
343         parent=win_container,
344         relative_position=(0, 0.375),
345         relative_size=(0.8, 0.1),
346         anchor_x='center',
347         border_width=0,
348         margin=0,
349         icon=GRAPHICS['by_draw'],
350         fill_colour=(0, 0, 0, 0),
351     ),
352     'by_timeout':
353     Icon(
354         parent=win_container,
355         relative_position=(0, 0.375),
356         relative_size=(0.8, 0.1),
357         anchor_x='center',
358         border_width=0,
359         margin=0,
360         icon=GRAPHICS['by_timeout'],
361         fill_colour=(0, 0, 0, 0),
362     )
363 }
```

## B.17 data\states\game\components

### B.17.1 bitboard\_collection.py

See Section 3.5.5.

### B.17.2 board.py

See Section 3.5.4.

### B.17.3 capture\_draw.py

```

1 from data.states.game.components.particles_draw import ParticlesDraw
2 from data.helpers.board_helpers import coords_to_screen_pos
3 from data.managers.animation import animation
4 from data.utils.constants import ShaderType
5 from data.managers.window import window
6 from data.utils.enums import Colour
7
8 class CaptureDraw:
9     def __init__(self, board_position, board_size):
10         self._board_position = board_position
11         self._square_size = board_size[0] / 10
12         self._particles_draw = ParticlesDraw()
13
14     def add_capture(self, piece, colour, rotation, piece_coords, sphinx_coords,
15                     active_colour, particles=True, shake=True):
16         if particles:
17             self._particles_draw.add_captured_piece(
18                 piece,
19                 colour,
20                 rotation,
```

```

20         coords_to_screen_pos(piece_coords, self._board_position, self.
21             _square_size),
22             self._square_size
23     )
24     self._particles_draw.add_sparks(
25         3,
26         (255, 0, 0) if active_colour == Colour.RED else (0, 0, 255),
27         coords_to_screen_pos(sphinx_coords, self._board_position, self.
28             _square_size)
29     )
30
31     if shake:
32         window.set_effect(ShaderType.SHAKE)
33         animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE))
34     )
35
36     def draw(self, screen):
37         self._particles_draw.draw(screen)
38
39     def update(self):
40         self._particles_draw.update()
41
42     def handle_resize(self, board_position, board_size):
43         self._board_position = board_position
44         self._square_size = board_size[0] / 10

```

#### B.17.4 father.py

```

1 import pygame
2 from data.states.game.components.piece_sprite import PieceSprite
3 from data.utils.enums import CursorMode
4 from data.managers.cursor import cursor
5
6 DRAG_THRESHOLD = 500
7
8 class DragAndDrop:
9     def __init__(self, board_position, board_size, change_cursor=True):
10         self._board_position = board_position
11         self._board_size = board_size
12         self._change_cursor = change_cursor
13         self._ticks_since_drag = 0
14
15         self.dragged_sprite = None
16
17     def set_dragged_piece(self, piece, colour, rotation):
18         sprite = PieceSprite(piece=piece, colour=colour, rotation=rotation)
19         sprite.set_geometry((0, 0), self._board_size[0] / 10)
20         sprite.set_image()
21
22         self.dragged_sprite = sprite
23         self._ticks_since_drag = pygame.time.get_ticks()
24
25         if self._change_cursor:
26             cursor.set_mode(CursorMode.CLOSEDHAND)
27
28     def remove_dragged_piece(self):
29         self.dragged_sprite = None
30         time_dragged = pygame.time.get_ticks() - self._ticks_since_drag
31         self._ticks_since_drag = 0
32
33         if self._change_cursor:
34             cursor.set_mode(CursorMode.OPENHAND)

```

```

35         return time_dragged > DRAG_THRESHOLD
36
37     def get_dragged_info(self):
38         return self.dragged_sprite.type, self.dragged_sprite.colour, self.
39             dragged_sprite.rotation
40
41     def draw(self, screen):
42         if self.dragged_sprite is None:
43             return
44
45         self.dragged_sprite.rect.center = pygame.mouse.get_pos()
46         screen.blit(self.dragged_sprite.image, self.dragged_sprite.rect.topleft)
47
48     def handle_resize(self, board_position, board_size):
49         if self.dragged_sprite:
50             self.dragged_sprite.set_geometry(board_position, board_size[0] / 10)
51
52         self._board_position = board_position
53         self._board_size = board_size

```

### B.17.5 fen\_parser.py

```

1  from data.helpers.bitboard_helpers import occupied_squares, bitboard_to_index
2  from data.utils.enums import Colour, RotationIndex, Rotation, Piece
3  from data.utils.constants import EMPTY_BB
4
5  def parse_fen_string(fen_string):
6      #sc3ncfcnccb2/2pc7/3Pd6/pa1Pcirbra1pb1Pd/pb1Pd1RaRbipa1Pc/6pb3/7Pa2/2
7      #PdNaFaNa3Sa b
8      piece_bitboards = {char: EMPTY_BB for char in Piece}, {char: EMPTY_BB for
9          char in Piece}]
10     rotation_bitboards = [EMPTY_BB, EMPTY_BB]
11     combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
12     combined_all_bitboard = 0
13     part_1, part_2 = fen_string.split(' ')
14
15     rank = 7
16     file = 0
17
18     piece_count = {char.lower(): 0 for char in Piece} | {char.upper(): 0 for char
19          in Piece}
20
21     for index, character in enumerate(part_1):
22         square = rank * 10 + file
23
24         if character.lower() in Piece:
25             piece_count[character] += 1
26             if character.isupper():
27                 piece_bitboards[Colour.BLUE][character.lower()] |= 1 << square
28
29             else:
30                 piece_bitboards[Colour.RED][character.lower()] |= 1 << square
31
32         rotation = part_1[index + 1]
33         match rotation:
34             case Rotation.UP:
35                 pass
36             case Rotation.RIGHT:
37                 rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
38             case Rotation.DOWN:
39                 rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square

```

```

37         case Rotation.LEFT:
38             rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
39             rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
40         case _:
41             raise ValueError('Invalid FEN String - piece character not
42 followed by rotational character')
43
44     file += 1
45     elif character in '0123456789':
46         if character == '1' and fen_string[index + 1] == '0':
47             file += 10
48             continue
49
50         file += int(character)
51     elif character == '/':
52         rank = rank - 1
53         file = 0
54     elif character in Rotation:
55         continue
56     else:
57         raise ValueError('Invalid FEN String - invalid character found:', character)
58
59     if piece_count['s'] != 1 or piece_count['S'] != 1:
60         raise ValueError('Invalid FEN string - invalid number of Sphinx pieces')
61     # COMMENTED OUT AS NO PHARAOH PIECES IS OKAY IF PARSING FEN STRING FOR
62     FINISHED GAME BOARD THUMBNAIL
63     elif piece_count['f'] > 1 or piece_count['F'] > 1:
64         raise ValueError('Invalid FEN string - invalid number of Pharaoh pieces')
65
66     if part_2 == 'b':
67         colour = Colour.BLUE
68     elif part_2 == 'r':
69         colour = Colour.RED
70     else:
71         raise ValueError('Invalid FEN string - invalid active colour')
72
73     for piece in Piece:
74         combined_colour_bitboards[Colour.BLUE] |= piece_bitboards[Colour.BLUE][piece]
75         combined_colour_bitboards[Colour.RED] |= piece_bitboards[Colour.RED][piece]
76
77     combined_all_bitboard = combined_colour_bitboards[Colour.BLUE] |
78     combined_colour_bitboards[Colour.RED]
79     return (piece_bitboards, combined_colour_bitboards, combined_all_bitboard,
80             rotation_bitboards, colour)
81
82 def encode_fen_string(bitboard_collection):
83     blue_bitboards = bitboard_collection.piece_bitboards[Colour.BLUE]
84     red_bitboards = bitboard_collection.piece_bitboards[Colour.RED]
85
86     fen_string_list = [''] * 80
87
88     for piece, bitboard in blue_bitboards.items():
89         for individual_bitboard in occupied_squares(bitboard):
90             index = bitboard_to_index(individual_bitboard)
91             rotation = bitboard_collection.get_rotation_on(individual_bitboard)
92             fen_string_list[index] = piece.upper() + rotation
93
94     for piece, bitboard in red_bitboards.items():
95         for individual_bitboard in occupied_squares(bitboard):
96

```

```

92         index = bitboard_to_index(individual_bitboard)
93         rotation = bitboard_collection.get_rotation_on(individual_bitboard)
94         fen_string_list[index] = piece.lower() + rotation
95
96         fen_string = ''
97         row_string = ''
98         empty_count = 0
99         for index, square in enumerate(fen_string_list):
100             if square == '':
101                 empty_count += 1
102             else:
103                 if empty_count > 0:
104                     row_string += str(empty_count)
105                     empty_count = 0
106
107                 row_string += square
108
109             if index % 10 == 9:
110                 if empty_count > 0:
111                     fen_string = '/' + row_string + str(empty_count) + fen_string
112                 else:
113                     fen_string = '/' + row_string + fen_string
114
115                 row_string = ''
116                 empty_count = 0
117
118         fen_string = fen_string[1:]
119
120         if bitboard_collection.active_colour == Colour.BLUE:
121             colour = 'b'
122         else:
123             colour = 'r'
124
125     return fen_string + ' ' + colour

```

### B.17.6 laser.py

```

1  from data.utils.constants import A_FILE_MASK, J_FILE_MASK, ONE_RANK_MASK,
2      EIGHT_RANK_MASK, EMPTY_BB
3  from data.helpers import bitboard_helpers as bb_helpers
4  from data.utils.enums import Piece, Colour, Rotation
5
6  class Laser:
7      def __init__(self, bitboards):
8          self._bitboards = bitboards
9          self.hit_square_bitboard, self.piece_hit, self.laser_path, self.
10             path_bitboard, self.pieces_on_trajectory = self.calculate_trajectory()
11
12          if (self.hit_square_bitboard != EMPTY_BB):
13              self.piece_rotation = self._bitboards.get_rotation_on(self.
14                  hit_square_bitboard)
15              self.piece_colour = self._bitboards.get_colour_on(self.
16                  hit_square_bitboard)
17
18      def calculate_trajectory(self):
19          current_square = self._bitboards.get_piece_bitboard(Piece.SPHINX, self.
20              _bitboards.active_colour)
21          previous_direction = self._bitboards.get_rotation_on(current_square)
22          trajectory_bitboard = 0b0
23          trajectory_list = []
24          square_animation_states = []
25          pieces_on_trajectory = []

```

```

21
22     while current_square:
23         current_piece = self._bitboards.get_piece_on(current_square, Colour.
24             BLUE) or self._bitboards.get_piece_on(current_square, Colour.RED)
25         current_rotation = self._bitboards.get_rotation_on(current_square)
26
27         next_square, direction, piece_hit = self.calculate_next_square(
28             current_square, current_piece, current_rotation, previous_direction)
29
30         trajectory_bitboard |= current_square
31         trajectory_list.append(bb_helpers.bitboard_to_coords(current_square))
32         square_animation_states.append(direction)
33
34         if previous_direction != direction:
35             pieces_on_trajectory.append(current_square)
36
37         if next_square == EMPTY_BB:
38             hit_square_bitboard = 0b0
39
40         if piece_hit:
41             hit_square_bitboard = current_square
42
43     return hit_square_bitboard, piece_hit, list(zip(trajectory_list,
44         square_animation_states)), trajectory_bitboard, pieces_on_trajectory
45
46     current_square = next_square
47     previous_direction = direction
48
49     def calculate_next_square(self, square, piece, rotation, previous_direction):
50         match piece:
51             case Piece.SPHINX:
52                 if previous_direction != rotation:
53                     return EMPTY_BB, previous_direction, None
54
55             next_square = self.next_square_bitboard(square, rotation)
56             return next_square, previous_direction, Piece.SPHINX
57
58             case Piece.PYRAMID:
59                 if previous_direction in [rotation, rotation.get_clockwise()]:
60                     return EMPTY_BB, previous_direction, Piece.PYRAMID
61
62                 if previous_direction == rotation.get_anticlockwise():
63                     new_direction = previous_direction.get_clockwise()
64                 else:
65                     new_direction = previous_direction.get_anticlockwise()
66
67             next_square = self.next_square_bitboard(square, new_direction)
68
69             return next_square, new_direction, None
70
71             case Piece.ANUBIS:
72                 if previous_direction == rotation.get_clockwise().get_clockwise():
73                     return EMPTY_BB, previous_direction, None
74
75             return EMPTY_BB, previous_direction, Piece.ANUBIS
76
77             case Piece.SCARAB:
78                 if previous_direction in [rotation.get_clockwise(), rotation.
79                     get_anticlockwise()]:
80                     new_direction = previous_direction.get_anticlockwise()
81                 else:
82                     new_direction = previous_direction.get_clockwise()

```

```

79             next_square = self.next_square_bitboard(square, new_direction)
80
81         return next_square, new_direction, None
82
83     case Piece.PHAROAH:
84         return EMPTY_BB, previous_direction, Piece.PHAROAH
85
86     case None:
87         next_square = self.next_square_bitboard(square, previous_direction)
88     )
89
90     return next_square, previous_direction, None
91
92     def next_square_bitboard(self, src_bitboard, previous_direction):
93         match previous_direction:
94             case Rotation.UP:
95                 masked_src_bitboard = src_bitboard & EIGHT_RANK_MASK
96                 return masked_src_bitboard << 10
97             case Rotation.RIGHT:
98                 masked_src_bitboard = src_bitboard & J_FILE_MASK
99                 return masked_src_bitboard << 1
100            case Rotation.DOWN:
101                masked_src_bitboard = src_bitboard & ONE_RANK_MASK
102                return masked_src_bitboard >> 10
103            case Rotation.LEFT:
104                masked_src_bitboard = src_bitboard & A_FILE_MASK
105                return masked_src_bitboard >> 1

```

### B.17.7 laser\_draw.py

See Section 3.4.1.

### B.17.8 move.py

```

1 import re
2 from data.helpers.bitboard_helpers import notation_to_bitboard, coords_to_bitboard
3     , bitboard_to_coords, bitboard_to_notation
4 from data.utils.enums import MoveType, Colour, RotationDirection
5 from data.managers.logs import initialise_logger
6
7 logger = initialise_logger(__name__)
8
9 class Move():
10     def __init__(self, move_type, src, dest=None, rotation_direction=None):
11         self.move_type = move_type
12         self.src = src
13         self.dest = dest
14         self.rotation_direction = rotation_direction
15
16     def to_notation(self, colour, piece, hit_square_bitboard):
17         hit_square = ''
18         if colour == Colour.BLUE:
19             piece = piece.upper()
20
21         if hit_square_bitboard:
22             hit_square = 'x' + bitboard_to_notation(hit_square_bitboard)
23
24         if self.move_type == MoveType.MOVE:
25             return 'M' + piece + bitboard_to_notation(self.src) +
26             bitboard_to_notation(self.dest) + hit_square

```

```
25         else:
26             return 'R' + piece + bitboard_to_notation(self.src) + self.
27 rotation_direction + hit_square
28
29     def __str__(self):
30         rotate_text = ''
31         coords_1 = '(' + chr(bitboard_to_coords(self.src)[0] + 65) + ',' + str(
32             bitboard_to_coords(self.src)[1] + 1) + ')'
33
34         if self.move_type == MoveType.ROTATE:
35             rotate_text = ' ' + self.rotation_direction.name
36             return f'{self.move_type.name}{rotate_text}: ON {coords_1}'
37
38         elif self.move_type == MoveType.MOVE:
39             coords_2 = '(' + chr(bitboard_to_coords(self.dest)[0] + 65) + ',' + str(
40                 bitboard_to_coords(self.dest)[1] + 1) + ')'
41             return f'{self.move_type.name}{rotate_text}: FROM {coords_1} TO {coords_2}'
42
43         # (Rotation: {self.rotation_direction})
44
45     @classmethod
46     def instance_from_notation(move_cls, notation):
47         try:
48             notation = notation.split('x')[0]
49             move_type = notation[0].lower()
50
51             moves = notation[2:]
52             letters = re.findall(r'[A-Za-z]+', moves)
53             numbers = re.findall(r'\d+', moves)
54
55             if move_type == MoveType.MOVE:
56                 src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
57                 dest_bitboard = notation_to_bitboard(letters[1] + numbers[1])
58
59                 return move_cls(move_type, src_bitboard, dest_bitboard)
60
61             elif move_type == MoveType.ROTATE:
62                 src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
63                 rotation_direction = RotationDirection(letters[1])
64
65                 return move_cls(move_type, src_bitboard, src_bitboard,
66 rotation_direction)
67             else:
68                 raise ValueError('(Move.instance_from_notation) Invalid move type')
69
70         except Exception as error:
71             logger.info('(Move.instance_from_notation) Error occured while parsing',
72 ':', error)
73             raise error
74
75     @classmethod
76     def instance_from_input(move_cls, move_type, src, dest=None, rotation=None):
77         try:
78             if move_type == MoveType.MOVE:
79                 src_bitboard = notation_to_bitboard(src)
80                 dest_bitboard = notation_to_bitboard(dest)
81
82             elif move_type == MoveType.ROTATE:
83                 src_bitboard = notation_to_bitboard(src)
84                 dest_bitboard = src_bitboard
```

```

80
81         return move_cls(move_type, src_bitboard, dest_bitboard, rotation)
82     except Exception as error:
83         logger.info('Error (Move.instance_from):', error)
84         raise error
85
86     @classmethod
87     def instance_from_coords(move_cls, move_type, src_coords, dest_coords=None,
88     rotation_direction=None):
89         try:
90             src_bitboard = coords_to_bitboard(src_coords)
91             dest_bitboard = coords_to_bitboard(dest_coords)
92
93             return move_cls(move_type, src_bitboard, dest_bitboard,
94             rotation_direction)
95         except Exception as error:
96             logger.info('Error (Move.instance_from_coords):', error)
97             raise error
98
99     @classmethod
100    def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
101        None, rotation_direction=None):
102        try:
103            return move_cls(move_type, src_bitboard, dest_bitboard,
104            rotation_direction)
105        except Exception as error:
106            logger.info('Error (Move.instance_from_bitboards):', error)
107            raise error

```

### B.17.9 overlay\_draw.py

```

1 import pygame
2 from data.utils.constants import OVERLAY_COLOUR_LIGHT, OVERLAY_COLOUR_DARK
3 from data.helpers.board_helpers import coords_to_screen_pos, screen_pos_to_coords,
4                                         create_square_overlay, create_circle_overlay
5
6 class OverlayDraw:
7     def __init__(self, board_position, board_size, limit_hover=True):
8         self._board_position = board_position
9         self._board_size = board_size
10
11         self._hovered_coords = None
12         self._selected_coords = None
13         self._available_coords = None
14
15         self._limit_hover = limit_hover
16
17         self._selected_overlay = None
18         self._hovered_overlay = None
19         self._available_overlay = None
20
21         self.initialise_overlay_surfaces()
22
23     @property
24     def square_size(self):
25         return self._board_size[0] / 10
26
27     def initialise_overlay_surfaces(self):
28         self._selected_overlay = create_square_overlay(self.square_size,
OVERLAY_COLOUR_DARK)
         self._hovered_overlay = create_square_overlay(self.square_size,
OVERLAY_COLOUR_LIGHT)

```

```

29         self._available_overlay = create_circle_overlay(self.square_size,
OVERLAY_COLOUR_LIGHT)
30
31     def set_hovered_coords(self, mouse_pos):
32         self._hovered_coords = screen_pos_to_coords(mouse_pos, self.
_board_position, self._board_size)
33
34     def set_selected_coords(self, coords):
35         self._selected_coords = coords
36
37     def set_available_coords(self, coords_list):
38         self._available_coords = coords_list
39
40     def set_hover_limit(self, new_limit):
41         self._limit_hover = new_limit
42
43     def draw(self, screen):
44         self.set_hovered_coords(pygame.mouse.get_pos())
45
46         if self._selected_coords:
47             screen.blit(self._selected_overlay, coords_to_screen_pos(self.
_selected_coords, self._board_position, self.square_size))
48
49         if self._available_coords:
50             for coords in self._available_coords:
51                 screen.blit(self._available_overlay, coords_to_screen_pos(coords,
self._board_position, self.square_size))
52
53         if self._hovered_coords:
54             if self._hovered_coords is None:
55                 return
56
57             if self._limit_hover and ((self._available_coords is None) or (self.
_hovered_coords not in self._available_coords)):
58                 return
59
60             screen.blit(self._hovered_overlay, coords_to_screen_pos(self.
_hovered_coords, self._board_position, self.square_size))
61
62     def handle_resize(self, board_position, board_size):
63         self._board_position = board_position
64         self._board_size = board_size
65
66         self.initialise_overlay_surfaces()

```

### B.17.10 particles\_draw.py

See Section 3.4.2.

### B.17.11 piece\_group.py

```

1 import pygame
2 from data.states.game.components.piece_sprite import PieceSprite
3 from data.utils.enums import Colour, Piece
4
5 class PieceGroup(pygame.sprite.Group):
6     def __init__(self):
7         super().__init__()
8
9     def initialise_pieces(self, piece_list, board_position, board_size):

```

```

10         self.empty()
11
12     for index, piece_and_rotation in enumerate(piece_list):
13         x = index % 10
14         y = index // 10
15
16         if piece_and_rotation:
17             if piece_and_rotation[0].isupper():
18                 colour = Colour.BLUE
19             else:
20                 colour = Colour.RED
21
22         piece = PieceSprite(piece=Piece(piece_and_rotation[0].lower()),
23                             colour=colour, rotation=piece_and_rotation[1])
24         piece.set_coords((x, y))
25         piece.set_geometry(board_position, board_size[0] / 10)
26         piece.set_image()
27         self.add(piece)
28
29     def set_geometry(self, board_position, board_size):
30         for sprite in self.sprites():
31             sprite.set_geometry(board_position, board_size[0] / 10)
32
33     def handle_resize(self, board_position, board_size):
34         self.set_geometry(board_position, board_size)
35
36         for sprite in self.sprites():
37             sprite.set_image()
38
39     def remove_piece(self, coords):
40         for sprite in self.sprites():
41             if sprite.coords == coords:
42                 sprite.kill()

```

### B.17.12 piece\_sprite.py

```

1 import pygame
2 from data.helpers.board_helpers import coords_to_screen_pos
3 from data.helpers.asset_helpers import scale_and_cache
4 from data.utils.assets import GRAPHICS
5 from data.utils.enums import Piece
6
7 class PieceSprite(pygame.sprite.Sprite):
8     def __init__(self, piece, colour, rotation):
9         super().__init__()
10        self.colour = colour
11        self.rotation = rotation
12
13        self.type = piece
14        self.coords = None
15        self.size = None
16
17    @property
18    def image_name(self):
19        return Piece(self.type).name.lower() + '_' + str(self.colour) + '_' + self.rotation
20
21    def set_image(self):
22        self.image = scale_and_cache(GRAPHICS[self.image_name], (self.size, self.size))
23
24    def set_geometry(self, new_position, square_size):

```

```

25         self.size = square_size
26         self.rect = pygame.FRect((0, 0, square_size, square_size))
27
28     if self.coords:
29         self.rect.topleft = coords_to_screen_pos(self.coords, new_position,
30         square_size)
31     else:
32         self.rect.topleft = new_position
33
34     def set_coords(self, new_coords):
35         self.coords = new_coords

```

### B.17.13 psqt.py

```

1  from data.utils.enums import Piece
2
3 FLIP = [
4      70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
5      60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
6      50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
7      40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
8      6, 31, 32, 33, 34, 35, 36, 37, 38, 39,
9      4, 21, 22, 23, 24, 25, 26, 27, 28, 29,
10     2, 11, 12, 13, 14, 3, 16, 17, 18, 19,
11     0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
12 ]
13
14 PSQT = {
15     Piece.PYRAMID: [
16         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
17         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
18         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
19         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
20         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
21         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
22         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
23         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
24     ],
25     Piece.ANUBIS: [
26         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
27         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
28         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
29         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
30         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
31         6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
32         4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
33         2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
34     ],
35     Piece.SCARAB: [
36         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
37         0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
38         0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
39         0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
40         0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
41         0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
42         0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
43         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
44     ],
45     Piece.PHAROAH: [
46         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
47         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
48         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

```

49      0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
50      0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
51      0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
52      0, 0, 0, 2, 2, 2, 2, 0, 0, 0,
53      0, 0, 0, 2, 4, 4, 2, 0, 0, 0,
54  ],
55 }
```

## B.18 data\states\game\cpu

### B.18.1 base.py

```

1 import time
2 from pprint import PrettyPrinter
3 from data.utils.enums import Colour, Score, Miscellaneous
4 from data.states.game.cpu.evaluator import Evaluator
5 from data.managers.logs import initialise_logger
6
7 logger = initialise_logger(__name__)
8 printer = PrettyPrinter(indent=2, sort_dicts=False)
9
10 class BaseCPU:
11     def __init__(self, callback, verbose=True):
12         self._evaluator = Evaluator(verbose=False)
13         self._verbose = verbose
14         self._callback = callback
15         self._stats = {}
16
17     def initialise_stats(self):
18         self._stats = {
19             'nodes': 0,
20             'leaf_nodes': 0,
21             'draws': 0,
22             'mates': 0,
23             'ms_per_node': 0,
24             'time_taken': time.time()
25         }
26
27     def print_stats(self, score, move):
28         """
29             Prints statistics after traversing tree.
30
31             Args:
32                 score (int): Final score obtained after traversal.
33                 move (Move): Best move obtained after traversal.
34
35             if self._verbose is False:
36                 return
37
38             self._stats['time_taken'] = round(1000 * (time.time() - self._stats['
39             time_taken']), 3)
40             self._stats['ms_per_node'] = round(self._stats['time_taken'] / self._stats
41             ['nodes'], 3)
42
43             # Prints stats across multiple lines
44             if self._verbose is True:
45                 logger.info(f'\n\n'
46                             f'{self.__str__()} Search Results:\n'
47                             f'{printer.pformat(self._stats)}\n'
48                             f'Best score: {score}    Best move: {move}\n'
49             )
```

```

48
49     # Prints stats in a compacted format
50     elif self._verbose.lower() == 'compact':
51         logger.info(self._stats)
52         logger.info(f'Best score: {score}    Best move: {move}')
53
54     def find_move(self, board, stop_event=None):
55         raise NotImplementedError
56
57     def search(self, board, depth, stop_event, absolute=False, **kwargs):
58         if stop_event and stop_event.is_set():
59             raise TimeoutError(f'Thread killed - stopping minimax function ({self.__str__().search()})')
60
61         self._stats['nodes'] += 1
62
63         if (winner := board.check_win()) is not None:
64             self._stats['leaf_nodes'] += 1
65             return self.process_win(winner, depth, absolute)
66
67         if depth == 0:
68             self._stats['leaf_nodes'] += 1
69             return self.evaluator.evaluate(board, absolute), None
70
71     def process_win(self, winner, depth, absolute):
72         self._stats['leaf_nodes'] += 1
73
74         if winner == Miscellaneous.DRAW:
75             self._stats['draws'] += 1
76             return 0, None
77         elif winner == Colour.BLUE or absolute:
78             self._stats['mates'] += 1
79             return Score.CHECKMATE + depth, None
80         elif winner == Colour.RED:
81             self._stats['mates'] += 1
82             return -Score.CHECKMATE - depth, None
83
84     def __str__(self):
85         return self.__class__.__name__

```

### B.18.2 cpu\_thread.py

See Section 3.6.6.

### B.18.3 evaluator.py

See Section 3.6.5.

### B.18.4 move\_orderer.py

```

1 from data.states.game.cpu.evaluator import Evaluator
2
3 class SimpleEvaluator:
4     def __init__(self):
5         self._evaluator = Evaluator(verbose=False)
6         self._cache = {}
7
8     def evaluate(self, board):
9         if (hashed := board.to_hash()) in self._cache:
10             return self._cache[hashed]

```

```

11         score = self._evaluator.evaluate_material(board, board.get_active_colour())
12     )
13     self._cache[hashed] = score
14
15     return score
16
17 class MoveOrderer:
18     def __init__(self):
19         self._evaluator = SimpleEvaluator()
20
21     # def get_eval(self, board, move):
22     #     laser_result = board.apply_move(move)
23     #     score = self._evaluator.evaluate(board)
24     #     board.undo_move(move, laser_result)
25     #     return score
26
27     # def score_moves(self, board, moves):
28     #     for i in range(len(moves)):
29     #         score = self.get_eval(board, moves[i])
30     #         moves[i] = (moves[i], score)
31
32     #     return moves
33
34     def best_move_to_front(self, moves, start_idx, laser_coords):
35         for i in range(start_idx + 1, len(moves)):
36             if moves[i].src in laser_coords:
37                 moves[i], moves[start_idx] = moves[start_idx], moves[i]
38         return
39
40     def get_moves(self, board, hint=None, laser_coords=None):
41         if hint:
42             yield hint
43
44         colour = board.get_active_colour()
45         moves = list(board.generate_all_moves(colour))
46
47         for i in range(len(moves)):
48             if laser_coords:
49                 self.best_move_to_front(moves, i, laser_coords)
50
51         yield moves[i]

```

### B.18.5 temp.py

```

1 from data.utils.constants import Score, Colour
2 from data.states.game.cpu.base import BaseCPU
3 from pprint import pprint
4
5 class MinimaxCPU(BaseCPU):
6     def __init__(self, max_depth, callback, verbose):
7         super().__init__(callback, verbose)
8         self._max_depth = max_depth
9
10    def find_move(self, board, stop_event):
11        # No bit_length bug as None type returned, so Move __str__ called on
12        # NoneType I think (just deal with None being returned)
13        try:
14            best_move = self.search(board, self._max_depth, -Score.INFINITE, Score
15            .INFINITE, stop_event)
16
17            if self._verbose:

```

```

16         print('\nCPU Search Results:')
17         pprint(self._stats)
18         print('Best move:', best_move, '\n')
19
20         self._callback(self._best_move)
21     except Exception as error:
22         print('(MinimaxBase.find_move) Error has occurred:')
23         raise error
24
25     def search(self, board, depth, alpha, beta, stop_event):
26         if stop_event.is_set():
27             raise Exception('Thread killed - stopping minimax function (CPU.
minimax)')
28
29         # cached_move, cached_score = self._transposition_table.get_entry(hash_key
=board.bitboards.get_hash(), depth=depth, alpha=alpha, beta=beta)
30         # if cached_move or cached_score:
31         #     if depth == self._max_depth:
32         #         self._best_move = cached_move
33         #     return cached_score
34
35
36         if depth == 0:
37             return self.evaluate(board)
38
39         is_maximiser = board.get_active_colour() == Colour.BLUE
40
41         if is_maximiser:
42             score = -Score.INFINITE
43
44             for move in board.generate_all_moves(board.get_active_colour()):
45                 before, before_score = board.bitboards.get_rotation_string(), self.
evaluate(board)
46
47                 laser_result = board.apply_move(move)
48                 new_score = self.minimax(board, depth - 1, alpha, beta, False,
stop_event)
49
50                 if new_score >= score:
51                     score = new_score
52
53                     if depth == self._max_depth:
54                         self._best_move = move
55
56                     board.undo_move(move, laser_result)
57
58                     alpha = max(alpha, score)
59                     if depth == self._max_depth: # https://stackoverflow.com/questions
/31429974/alphabeta-pruning-alpha-equals-or-greater-than-beta-why-equals
60                         if beta < alpha:
61                             break
62                     else:
63                         if beta <= alpha:
64                             break
65
66                     after, after_score = board.bitboards.get_rotation_string(), self.
evaluate(board)
67                     if (before != after or before_score != after_score):
68                         print('shit\n\n')
69
70             return score
71

```

```

72         else:
73             score = Score.INFINITE
74
75         for move in board.generate_all_moves(board.get_active_colour()):
76             bef, before_score = board.bitboards.get_rotation_string(), self.
77             evaluate(board)
78
79             laser_result = board.apply_move(move)
80             new_score = self.minimax(board, depth - 1, alpha, beta, False,
81             stop_event)
82
83             if new_score <= score:
84                 score = new_score
85                 if depth == self._max_depth:
86                     self._best_move = move
87
88                 board.undo_move(move, laser_result)
89
90                 beta = min(beta, score)
91                 if depth == self._max_depth:
92                     if beta < alpha:
93                         break
94                 else:
95                     if beta <= alpha:
96                         break
97
98                 after, after_score = board.bitboards.get_rotation_string(), self.
99                 evaluate(board)
100                if (bef != after or before_score != after_score):
101                    print('shit\n\n')
102                    raise ValueError
103
104        return score

```

**B.18.6 transposition\_table.py**

See Section 3.6.8.

**B.18.7 zobrist\_hasher.py**

See Section 3.6.7.

**B.19 data\states\game\cpu\engines****B.19.1 alpha\_beta.py**

See Section 3.6.2.

**B.19.2 iterative\_deepening.py**

See Section 3.6.4.

**B.19.3 minimax.py**

See Section 3.6.1.

### B.19.4 negamax.py

```

1  from random import choice
2  from data.states.game.cpu.engines.transposition_table import
   TranspositionTableMixin
3  from data.states.game.cpu.engines.iterative_deepening import
   IterativeDeepeningMixin
4  from data.states.game.cpu.base import BaseCPU
5  from data.utils.enums import Score
6
7  class NegamaxCPU(BaseCPU):
8      def __init__(self, max_depth, callback, verbose=False):
9          super().__init__(callback, verbose)
10         self._max_depth = max_depth
11
12     def find_move(self, board, stop_event):
13         self.initialise_stats()
14         best_score, best_move = self.search(board, self._max_depth, stop_event)
15
16         if self._verbose:
17             self.print_stats(best_score, best_move)
18
19         self._callback(best_move)
20
21     def search(self, board, depth, stop_event, moves=None):
22         if (base_case := super().search(board, depth, stop_event, absolute=True)):
23             return base_case
24
25         best_move = None
26         best_score = -Score.INFINITE
27
28         for move in board.generate_all_moves(board.get_active_colour()):
29             laser_result = board.apply_move(move)
30
31             new_score = self.search(board, depth - 1, stop_event)[0]
32             new_score = -new_score
33
34             if new_score > best_score:
35                 best_score = new_score
36                 best_move = move
37             elif new_score == best_score:
38                 best_move = choice([best_move, move])
39
40             board.undo_move(move, laser_result)
41
42         return best_score, best_move
43
44 class ABNegamaxCPU(BaseCPU):
45     def __init__(self, max_depth, callback, verbose=True):
46         super().__init__(callback, verbose)
47         self._max_depth = max_depth
48
49     def initialise_stats(self):
50         """Initialises the statistics for the search."""
51         super().initialise_stats()
52         self._stats['beta_prunes'] = 0
53
54     def find_move(self, board, stop_event):
55         """Finds the best move for the current board state.
56
57         Args:
58             board (Board): The current board state.

```

```

59         stop_event (threading.Event): The event to signal stopping the search.
60     """
61     self.initialise_stats()
62     best_score, best_move = self.search(board, self._max_depth, -Score.
63                                         INFINITE, Score.INFINITE, stop_event)
64
65     if self._verbose:
66         self.print_stats(best_score, best_move)
67
68     self._callback(best_move)
69
70     def search(self, board, depth, alpha, beta, stop_event):
71         """Searches for the best move using the Alpha-Beta Negamax algorithm.
72
73         Args:
74             board (Board): The current board state.
75             depth (int): The current depth in the game tree.
76             alpha (int): The alpha value for pruning.
77             beta (int): The beta value for pruning.
78             stop_event (threading.Event): The event to signal stopping the search.
79
80         Returns:
81             tuple: The best score and the best move found.
82         """
83         if (base_case := super().search(board, depth, stop_event, absolute=True)):
84             return base_case
85
86         best_move = None
87         best_score = alpha
88
89         for move in board.generate_all_moves(board.get_active_colour()):
90             laser_result = board.apply_move(move)
91
92             new_score = self.search(board, depth - 1, -beta, -best_score,
93                                     stop_event)[0]
94             new_score = -new_score
95
96             if new_score > best_score:
97                 best_score = new_score
98                 best_move = move
99             elif new_score == best_score:
100                 best_move = choice([best_move, move])
101
102             board.undo_move(move, laser_result)
103
104             if best_score >= beta:
105                 self._stats['beta_prunes'] += 1
106                 break
107
108     class TTNegamaxCPU(TranspositionTableMixin, ABNegamaxCPU):
109         def initialise_stats(self):
110             """Initialises the statistics for the search."""
111             super().initialise_stats()
112             self._stats['cache_hits'] = 0
113
114         def print_stats(self, score, move):
115             """Prints the statistics for the search.
116
117             Args:
118                 score (int): The best score found.

```

```

119         move (Move): The best move found.
120         """
121         self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
122             self._stats['nodes'], 3)
122         self._stats['cache_entries'] = len(self._table._table)
123         super().print_stats(score, move)
124
125 class IDNegamaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABNegamaxCPU):
126     :
127     def initialise_stats(self):
128         """Initialises the statistics for the search."""
129         super().initialise_stats()
130         self._stats['cache_hits'] = 0
131
132     def print_stats(self, score, move):
133         """Prints the statistics for the search.
134
135         Args:
136             score (int): The best score found.
137             move (Move): The best move found.
138         """
139         self._stats['cache_hits_percentage'] = self._stats['cache_hits'] / self.
140             _stats['nodes']
141         self._stats['cache_entries'] = len(self._table._table)
142         super().print_stats(score, move)

```

### B.19.5 simple.py

```

1 from data.states.game.cpu.base import BaseCPU
2 from data.utils.enums import Colour, Score
3
4 class SimpleCPU(BaseCPU):
5     def __init__(self, callback, verbose=True):
6         super().__init__(callback, verbose)
7
8     def find_move(self, board, stop_event=None):
9         self.initialise_stats()
10        best_score, best_move = self.search(board, stop_event)
11
12        if self._verbose:
13            self.print_stats(best_score, best_move)
14
15        self._callback(best_move)
16
17    def search(self, board, stop_event):
18        if stop_event and stop_event.is_set():
19            raise Exception('Thread killed - stopping simple function (SimpleCPU.
20                search)')
21
22        active_colour = board.bitboards.active_colour
23        best_score = -Score.INFINITE if active_colour == Colour.BLUE else Score.
24        INFINITE
25        best_move = None
26
27        for move in board.generate_all_moves(active_colour):
28            laser_result = board.apply_move(move)
29
30            self._stats['nodes'] += 1
31
32            if winner := board.check_win() is not None:
33                self.process_win(winner)
34            else:

```

```

33         self._stats['leaf_nodes'] += 1
34
35         score = self._evaluator.evaluate(board)
36
37         if (active_colour == Colour.BLUE and score > best_score) or (
38             active_colour == Colour.RED and score < best_score):
39             best_move = move
40             best_score = score
41
42         board.undo_move(move, laser_result)
43
44     return best_score, best_move

```

**B.19.6 transposition\_table.py**

See Section 3.6.3.

**B.19.7 \_\_init\_\_.py**

```

1 from data.states.game.cpu.engines.simple import SimpleCPU
2 from data.states.game.cpu.engines.negamax import NegamaxCPU
3 from data.states.game.cpu.engines.minimax import MinimaxCPU
4 from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU
5 from data.states.game.cpu.engines.iterative_deepening import IDMMinimaxCPU
6 from data.states.game.cpu.engines.transposition_table import TTMinimaxCPU

```

**B.20 data\states\game\mvc****B.20.1 game\_controller.py**

See Section 3.5.3.

**B.20.2 game\_model.py**

See Section 3.5.1.

**B.20.3 game\_view.py**

See Section 3.5.2.

**B.20.4 pause\_view.py**

```

1 import pygame
2 from data.states.game.widget_dict import PAUSE_WIDGETS
3 from data.components.widget_group import WidgetGroup
4 from data.utils.event_types import GameEventType
5 from data.utils.constants import PAUSE_COLOUR
6 from data.managers.window import window
7 from data.managers.audio import audio
8
9 class PauseView:
10     def __init__(self, model):
11         self._model = model
12
13         self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
14         self._screen_overlay.fill(PAUSE_COLOUR)

```

```

15
16         self._widget_group = WidgetGroup(PAUSE_WIDGETS)
17         self._widget_group.handle_resize(window.size)
18
19         self._model.register_listener(self.process_model_event, 'pause')
20
21         self._event_to_func_map = {
22             GameEventType.PAUSE_CLICK: self.handle_pause_click
23         }
24
25         self.states = {
26             'PAUSED': False
27         }
28
29     def handle_pause_click(self, event):
30         self.states['PAUSED'] = not self.states['PAUSED']
31
32         if self.states['PAUSED']:
33             audio.pause_sfx()
34         else:
35             audio.unpause_sfx()
36
37     def handle_resize(self):
38         self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
39         self._screen_overlay.fill(PAUSE_COLOUR)
40         self._widget_group.handle_resize(window.size)
41
42     def draw(self):
43         if self.states['PAUSED']:
44             window.screen.blit(self._screen_overlay, (0, 0))
45             self._widget_group.draw()
46
47     def process_model_event(self, event):
48         try:
49             self._event_to_func_map.get(event.type)(event)
50         except:
51             raise KeyError('Event type not recognized in Paused View (PauseView. process_model_event)', event)
52
53     def convert_mouse_pos(self, event):
54         return self._widget_group.process_event(event)

```

### B.20.5 win\_view.py

```

1  from data.utils.enums import Colour, Miscellaneous, CursorMode
2  from data.components.widget_group import WidgetGroup
3  from data.states.game.widget_dict import WIN_WIDGETS
4  from data.managers.window import window
5  from data.managers.cursor import cursor
6
7  class WinView:
8      def __init__(self, model):
9          self._model = model
10
11          self._widget_group = WidgetGroup(WIN_WIDGETS)
12          self._widget_group.handle_resize(window.size)
13
14      def handle_resize(self):
15          self._widget_group.handle_resize(window.size)
16
17      def draw(self):
18          if self._model.states['WINNER'] is not None:

```

```

19         if cursor.get_mode() != CursorMode.ARROW:
20             cursor.set_mode(CursorMode.ARROW)
21
22         if self._model.states['WINNER'] == Colour.BLUE:
23             WIN_WIDGETS['red_won'].kill()
24             WIN_WIDGETS['draw_won'].kill()
25         elif self._model.states['WINNER'] == Colour.RED:
26             WIN_WIDGETS['blue_won'].kill()
27             WIN_WIDGETS['draw_won'].kill()
28         elif self._model.states['WINNER'] == Miscellaneous.DRAW:
29             WIN_WIDGETS['red_won'].kill()
30             WIN_WIDGETS['blue_won'].kill()
31
32         self._widget_group.draw()
33
34     def set_win_type(self, win_type):
35         WIN_WIDGETS['by_draw'].kill()
36         WIN_WIDGETS['by_timeout'].kill()
37         WIN_WIDGETS['by_resignation'].kill()
38         WIN_WIDGETS['by_checkmate'].kill()
39
40         match win_type:
41             case 'CAPTURE':
42                 self._widget_group.add(WIN_WIDGETS['by_checkmate'])
43             case 'DRAW':
44                 self._widget_group.add(WIN_WIDGETS['by_draw'])
45             case 'RESIGN':
46                 self._widget_group.add(WIN_WIDGETS['by_resignation'])
47             case 'TIME':
48                 self._widget_group.add(WIN_WIDGETS['by_timeout'])
49
50     def convert_mouse_pos(self, event):
51         return self._widget_group.process_event(event)

```

## B.21 data\states\menu

### B.21.1 menu.py

```

1 import pygame
2 import sys
3 from random import randint
4 from data.helpers.asset_helpers import get_rotational_angle
5 from data.helpers.asset_helpers import scale_and_cache
6 from data.states.menu.widget_dict import MENU_WIDGETS
7 from data.utils.assets import GRAPHICS, MUSIC, SFX
8 from data.utils.logs import initialise_logger
9 from data.utils.event_types import MenuEventType
10 from data.managers.animation import animation
11 from data.utils.constants import ShaderType
12 from data.managers.window import window
13 from data.managers.audio import audio
14 from data.control import _State
15
16 logger = initialise_logger(__file__)
17
18 class Menu(_State):
19     def __init__(self):
20         super().__init__()
21         self._fire_laser = False
22         self._bloom_mask = None
23         self._laser_mask = None

```

```

24
25     def cleanup(self):
26         super().cleanup()
27
28         window.clear_apply_arguments(ShaderType.BLOOM)
29         window.clear_apply_arguments(ShaderType.SHAKE)
30         window.clear_effect(ShaderType.CHROMATIC_ABBREVIATION)
31
32     return None
33
34     def startup(self, persist=None):
35         super().startup(MENU_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
36         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
37 BACKGROUND_BALATRO)
38         window.set_effect(ShaderType.CHROMATIC_ABBREVIATION)
39
40         MENU_WIDGETS['credits'].kill()
41
42         self._fire_laser = False
43         self._bloom_mask = None
44         self._laser_mask = None
45
46         self.draw()
47         self.update_masks()
48
49     @property
50     def sphinx_center(self):
51         return (window.size[0] - self.sphinx_size[0] / 2, window.size[1] - self.
52 sphinx_size[1] / 2)
53
54     @property
55     def sphinx_size(self):
56         return (min(window.size) * 0.1, min(window.size) * 0.1)
57
58     @property
59     def sphinx_rotation(self):
60         mouse_pos = (pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1] + 0.01)
61         return -get_rotational_angle(mouse_pos, self.sphinx_center)
62
63     def get_event(self, event):
64         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
65             MENU_WIDGETS['credits'].kill()
66
67             if event.type == pygame.MOUSEBUTTONDOWN:
68                 self._fire_laser = True
69                 audio.play_sfx(SFX['menu_laser_windup'])
70                 audio.play_sfx(SFX['menu_laser_loop'], loop=True)
71                 animation.set_timer(SFX['menu_laser_loop'].get_length() * 1000 / 2,
72 lambda: audio.play_sfx(SFX['menu_laser_loop'], loop=True) if self._fire_laser
73 else ...) # OVERLAP TWO LOOPS TO HIDE TRANSITION
74
75             elif event.type == pygame.MOUSEBUTTONUP:
76                 self._fire_laser = False
77
78                 window.clear_effect(ShaderType.RAYS)
79                 animation.set_timer(300, lambda: window.clear_effect(ShaderType.SHAKE)
80             )
81
82             audio.stop_sfx(1000)
83
84             widget_event = self._widget_group.process_event(event)
85
86             if widget_event is None:

```

```

81         return
82
83     match widget_event.type:
84         case None:
85             return
86
87         case MenuEventType.CONFIG_CLICK:
88             self.next = 'config'
89             self.done = True
90         case MenuEventType.SETTINGS_CLICK:
91             self.next = 'settings'
92             self.done = True
93         case MenuEventType.BROWSER_CLICK:
94             self.next = 'browser'
95             self.done = True
96         case MenuEventType.QUIT_CLICK:
97             pygame.quit()
98             sys.exit()
99             logger.info('quitting...')
100        case MenuEventType.CREDITS_CLICK:
101            self._widget_group.add(MENU_WIDGETS['credits'])
102
103    def draw_sphinx(self):
104        sphinx_surface = scale_and_cache(GRAPHICS['sphinx_0_b'], self.sphinx_size)
105        sphinx_surface = pygame.transform.rotate(sphinx_surface, self.sphinx_rotation)
106        sphinx_rect = pygame.FRect(0, 0, *self.sphinx_size)
107        sphinx_rect.center = self.sphinx_center
108
109        window.screen.blit(sphinx_surface, sphinx_rect)
110
111    def update_masks(self):
112        self.draw()
113
114        widget_mask = window.screen.copy()
115        laser_mask = pygame.mask.from_surface(widget_mask)
116        laser_mask = laser_mask.to_surface(setcolor=(255, 0, 0, 255), unsetcolor
117        =(0, 0, 0, 255))
118        pygame.draw.rect(laser_mask, (0, 0, 0), (window.screen.width - self.
119        sphinx_size[0], window.screen.height - self.sphinx_size[1], *self.sphinx_size)
120        )
121        pygame.draw.rect(widget_mask, (0, 0, 0, 255), (window.screen.width - 50,
122        0, 50, 50))
123
124        self._bloom_mask = widget_mask
125        self._laser_mask = laser_mask
126
127    def draw(self):
128        self._widget_group.draw()
129        self.draw_sphinx()
130
131        if self._fire_laser:
132            window.set_apply_arguments(ShaderType.RAYS, occlusion=self._laser_mask
133            , softShadow=0.1)
134
135            window.set_apply_arguments(ShaderType.BLOOM, highlight_surface=self.
136            _bloom_mask, surface_intensity=0.3, brightness_intensity=0.6)
137
138    def update(self, **kwargs):
139        random_offset = lambda: randint(-5, 5) / 40
140        if self._fire_laser:
141            window.clear_effect(ShaderType.RAYS)

```

```

136         window.set_effect(ShaderType.RAYS, lights=[[
137             (self.sphinx_center[0] / window.size[0], self.sphinx_center[1] /
138             window.size[1]),
139             2.2,
140             (190, 190, 255),
141             0.99,
142             (self.sphinx_rotation - 2 + random_offset(), self.sphinx_rotation
143             + 2 + random_offset())
144             ]])
145
146         window.set_effect(ShaderType.SHAKE)
147         window.set_apply_arguments(ShaderType.SHAKE, intensity=1)
148         pygame.mouse.set_pos(pygame.mouse.get_pos()[0] + random_offset(),
149         pygame.mouse.get_pos()[1] + random_offset())
150
151     super().update(**kwargs)
152
153     def handle_resize(self):
154         super().handle_resize()
155         self.update_masks()

```

### B.21.2 widget\_dict.py

```

1 from data.components.custom_event import CustomEvent
2 from data.utils.event_types import MenuEventType
3 from data.utils.assets import GRAPHICS
4 from data.managers.theme import theme
5 from data.widgets import *
6
7 top_right_container = Rectangle(
8     relative_position=(0, 0),
9     relative_size=(0.15, 0.075),
10    fixed_position=(5, 5),
11    anchor_x='right',
12    scale_mode='height'
13 )
14
15 MENU_WIDGETS = {
16     'credits':
17     Icon(
18         relative_position=(0, 0),
19         relative_size=(0.7, 0.7),
20         icon=GRAPHICS['credits'],
21         anchor_x='center',
22         anchor_y='center',
23         margin=50
24     ),
25     'default': [
26         top_right_container,
27         ReactiveIconButton(
28             parent=top_right_container,
29             relative_position=(0, 0),
30             relative_size=(1, 1),
31             anchor_x='right',
32             scale_mode='height',
33             base_icon=GRAPHICS['quit_base'],
34             hover_icon=GRAPHICS['quit_hover'],
35             press_icon=GRAPHICS['quit_press'],
36             event=CustomEvent(MenuEventType.QUIT_CLICK)
37         ),
38         ReactiveIconButton(
39             parent=top_right_container,

```

```

40         relative_position=(0, 0),
41         relative_size=(1, 1),
42         scale_mode='height',
43         base_icon=GRAPHICS['credits_base'],
44         hover_icon=GRAPHICS['credits_hover'],
45         press_icon=GRAPHICS['credits_press'],
46         event=CustomEvent(MenuEventType.CREDITS_CLICK)
47     ),
48     ReactiveIconButton(
49         relative_position=(0.05, -0.2),
50         relative_size=(0, 0.15),
51         anchor_y='center',
52         base_icon=GRAPHICS['play_text_base'],
53         hover_icon=GRAPHICS['play_text_hover'],
54         press_icon=GRAPHICS['play_text_press'],
55         event=CustomEvent(MenuEventType.CONFIG_CLICK)
56     ),
57     ReactiveIconButton(
58         relative_position=(0.05, 0),
59         relative_size=(0, 0.15),
60         anchor_y='center',
61         base_icon=GRAPHICS['review_text_base'],
62         hover_icon=GRAPHICS['review_text_hover'],
63         press_icon=GRAPHICS['review_text_press'],
64         event=CustomEvent(MenuEventType.BROWSER_CLICK)
65     ),
66     ReactiveIconButton(
67         relative_position=(0.05, 0.2),
68         relative_size=(0, 0.15),
69         anchor_y='center',
70         base_icon=GRAPHICS['settings_text_base'],
71         hover_icon=GRAPHICS['settings_text_hover'],
72         press_icon=GRAPHICS['settings_text_press'],
73         event=CustomEvent(MenuEventType.SETTINGS_CLICK)
74     ),
75     Icon(
76         relative_position=(0.0, 0.1),
77         relative_size=(0.3, 0.2),
78         anchor_x='center',
79         fill_colour=theme['fillSecondary'],
80         icon=GRAPHICS['title_screen_art'],
81         stretch=False
82     ),
83 ]
84 }
85
86 # Widgets used for testing light rays effect
87 TEST_WIDGETS = {
88     'default': [
89         Rectangle(
90             relative_position=(0.4, 0.2),
91             relative_size=(0.1, 0.1),
92             scale_mode='height',
93             visible=True,
94             border_width=0,
95             fill_colour=(255, 0, 0),
96             border_radius=1000
97         ),
98         Rectangle(
99             relative_position=(0.5, 0.7),
100            relative_size=(0.1, 0.1),
101            scale_mode='height',

```

```
102         visible=True,
103         border_width=0,
104         fill_colour=(255, 0, 0),
105         border_radius=1000
106     ),
107     Rectangle(
108         relative_position=(0.6, 0.6),
109         relative_size=(0.2, 0.2),
110         scale_mode='height',
111         visible=True,
112         border_width=0,
113         fill_colour=(255, 0, 0),
114         border_radius=1000
115     ),
116     Rectangle(
117         relative_position=(0.4, 0.6),
118         relative_size=(0.1, 0.1),
119         scale_mode='height',
120         visible=True,
121         border_width=0,
122         fill_colour=(255, 0, 0),
123         border_radius=1000
124     ),
125     Rectangle(
126         relative_position=(0.6, 0.4),
127         relative_size=(0.1, 0.1),
128         scale_mode='height',
129         visible=True,
130         border_width=0,
131         fill_colour=(255, 0, 0),
132         border_radius=1000
133     ),
134     Rectangle(
135         relative_position=(0.3, 0.4),
136         relative_size=(0.1, 0.1),
137         scale_mode='height',
138         visible=True,
139         border_width=0,
140         fill_colour=(255, 0, 0),
141         border_radius=1000
142     ),
143     Rectangle(
144         relative_position=(0.475, 0.15),
145         relative_size=(0.2, 0.2),
146         scale_mode='height',
147         visible=True,
148         border_width=0,
149         fill_colour=(255, 0, 0),
150         border_radius=1000
151     ),
152     Rectangle(
153         relative_position=(0.6, 0.2),
154         relative_size=(0.1, 0.1),
155         scale_mode='height',
156         visible=True,
157         border_width=0,
158         fill_colour=(255, 0, 0),
159         border_radius=1000
160     )
161 ]
162 }
```

## B.22 data\states\review

### B.22.1 review.py

See Section 3.7.1.

### B.22.2 widget\_dict.py

```

1 from data.widgets import *
2 from data.components.custom_event import CustomEvent
3 from data.utils.event_types import ReviewEventType
4 from data.utils.assets import GRAPHICS
5 from data.utils.enums import Colour
6
7 MOVE_LIST_WIDTH = 0.2
8
9 right_container = Rectangle(
10     relative_position=(0.05, 0),
11     relative_size=(0.2, 0.7),
12     anchor_y='center',
13     anchor_x='right'
14 )
15
16 info_container = Rectangle(
17     parent=right_container,
18     relative_position=(0, 0.5),
19     relative_size=(1, 0.5),
20     visible=True
21 )
22
23 arrow_container = Rectangle(
24     relative_position=(0, 0.05),
25     relative_size=(0.4, 0.1),
26     anchor_x='center',
27     anchor_y='bottom'
28 )
29
30 move_list = MoveList(
31     parent=right_container,
32     relative_position=(0, 0),
33     relative_width=1,
34     minimum_height=300,
35     move_list=[]
36 )
37
38 top_right_container = Rectangle(
39     relative_position=(0, 0),
40     relative_size=(0.15, 0.075),
41     fixed_position=(5, 5),
42     anchor_x='right',
43     scale_mode='height'
44 )
45
46 REVIEW_WIDGETS = {
47     'help':
48     Icon(
49         relative_position=(0, 0),
50         relative_size=(1.02, 1.02),
51         icon=GRAPHICS['review_help'],
52         anchor_x='center',

```

```

53         anchor_y='center',
54         border_width=0,
55         fill_colour=(0, 0, 0, 0)
56     ),
57     'default': [
58         arrow_container,
59         right_container,
60         info_container,
61         top_right_container,
62         ReactiveIconButton(
63             parent=top_right_container,
64             relative_position=(0, 0),
65             relative_size=(1, 1),
66             anchor_x='right',
67             scale_mode='height',
68             base_icon=GRAPHICS['home_base'],
69             hover_icon=GRAPHICS['home_hover'],
70             press_icon=GRAPHICS['home_press'],
71             event=CustomEvent(ReviewEventType.MENU_CLICK)
72         ),
73         ReactiveIconButton(
74             parent=top_right_container,
75             relative_position=(0, 0),
76             relative_size=(1, 1),
77             scale_mode='height',
78             base_icon=GRAPHICS['help_base'],
79             hover_icon=GRAPHICS['help_hover'],
80             press_icon=GRAPHICS['help_press'],
81             event=CustomEvent(ReviewEventType.HELP_CLICK)
82         ),
83         ReactiveIconButton(
84             parent=arrow_container,
85             relative_position=(0, 0),
86             relative_size=(1, 1),
87             scale_mode='height',
88             base_icon=GRAPHICS['left_arrow_filled_base'],
89             hover_icon=GRAPHICS['left_arrow_filled_hover'],
90             press_icon=GRAPHICS['left_arrow_filled_press'],
91             event=CustomEvent(ReviewEventType.PREVIOUS_CLICK)
92         ),
93         ReactiveIconButton(
94             parent=arrow_container,
95             relative_position=(0, 0),
96             relative_size=(1, 1),
97             scale_mode='height',
98             anchor_x='right',
99             base_icon=GRAPHICS['right_arrow_filled_base'],
100            hover_icon=GRAPHICS['right_arrow_filled_hover'],
101            press_icon=GRAPHICS['right_arrow_filled_press'],
102            event=CustomEvent(ReviewEventType.NEXT_CLICK)
103        ),
104    ],
105    'move_list':
106        move_list,
107    'scroll_area':
108        ScrollArea(
109            parent=right_container,
110            relative_position=(0, 0),
111            relative_size=(1, 0.5),
112            vertical=True,
113            widget=move_list
114        ),

```

```

115     'chessboard':
116     Chessboard(
117         relative_position=(0, 0),
118         relative_width=0.4,
119         scale_mode='width',
120         anchor_x='center',
121         anchor_y='center'
122     ),
123     'move_number_text':
124     Text(
125         parent=info_container,
126         relative_position=(0, 0),
127         relative_size=(1, 0.3),
128         anchor_y='bottom',
129         text='MOVE NO:',
130         fit_vertical=False,
131         margin=10,
132         border_width=0,
133         fill_colour=(0, 0, 0, 0),
134     ),
135     'move_colour_text':
136     Text(
137         parent=info_container,
138         relative_size=(1, 0.3),
139         relative_position=(0, 0),
140         anchor_y='center',
141         text='TO MOVE',
142         fit_vertical=False,
143         margin=10,
144         border_width=0,
145         fill_colour=(0, 0, 0, 0),
146     ),
147     'winner_text':
148     Text(
149         parent=info_container,
150         relative_size=(1, 0.3),
151         relative_position=(0, 0),
152         text='WINNER:',
153         fit_vertical=False,
154         margin=10,
155         border_width=0,
156         fill_colour=(0, 0, 0, 0),
157     ),
158     'blue_timer':
159     Timer(
160         relative_position=(0.05, 0.05),
161         anchor_y='center',
162         relative_size=(0.1, 0.1),
163         active_colour=Colour.BLUE,
164     ),
165     'red_timer':
166     Timer(
167         relative_position=(0.05, -0.05),
168         anchor_y='center',
169         relative_size=(0.1, 0.1),
170         active_colour=Colour.RED
171     ),
172     'timer_disabled_text':
173     Text(
174         relative_size=(0.2, 0.1),
175         relative_position=(0.05, 0),
176         anchor_y='center',

```

```

177         fit_vertical=False,
178         text='TIMER DISABLED',
179     ),
180     'blue_piece_display':
181     PieceDisplay(
182         relative_position=(0.05, 0.05),
183         relative_size=(0.2, 0.1),
184         anchor_y='bottom',
185         active_colour=Colour.BLUE
186     ),
187     'red_piece_display':
188     PieceDisplay(
189         relative_position=(0.05, 0.05),
190         relative_size=(0.2, 0.1),
191         active_colour=Colour.RED
192     ),
193 }
```

## B.23 data\states\settings

### B.23.1 settings.py

```

1 import pygame
2 from random import randint
3 from data.helpers.data_helpers import get_default_settings, get_user_settings,
4     update_user_settings
5 from data.utils.constants import WidgetState, ShaderType, SHADER_MAP
6 from data.states.settings.widget_dict import SETTINGS_WIDGETS
7 from data.utils.event_types import SettingsEventType
8 from data.managers.logs import initialise_logger
9 from data.managers.window import window
10 from data.managers.audio import audio
11 from data.widgets import ColourPicker
12 from data.utils.assets import MUSIC
13 from data.control import _State
14
15 logger = initialise_logger(__name__)
16
17 class Settings(_State):
18     def __init__(self):
19         super().__init__()
20
21         self._colour_picker = None
22         self._settings = None
23
24     def cleanup(self):
25         super().cleanup()
26
27         update_user_settings(self._settings)
28
29     return None
30
31     def startup(self, persist=None):
32         super().startup(SETTINGS_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
33
34         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
35             BACKGROUND_BALATRO)
36         self._settings = get_user_settings()
37         self.reload_settings()
38
39         self.draw()
```

```

38
39     def create_colour_picker(self, mouse_pos, button_type):
40         if button_type == SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK:
41             selected_colour = self._settings['primaryBoardColour']
42             event_type = SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK
43         else:
44             selected_colour = self._settings['secondaryBoardColour']
45             event_type = SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK
46
47             self._colour_picker = ColourPicker(
48                 relative_position=(mouse_pos[0] / window.size[0], mouse_pos[1] /
49                     window.size[1]),
50                 relative_width=0.15,
51                 selected_colour=selected_colour,
52                 event_type=event_type
53             )
54             self._widget_group.add(self._colour_picker)
55
56     def remove_colour_picker(self):
57         self._colour_picker.kill()
58
59     def reload_display_mode(self):
60         relative_mouse_pos = (pygame.mouse.get_pos()[0] / window.size[0], pygame.
61         mouse.get_pos()[1] / window.size[1])
62
63         if self._settings['displayMode'] == 'fullscreen':
64             window.setFullscreen(desktop=True)
65             window.handle_resize()
66
67         elif self._settings['displayMode'] == 'windowed':
68             window.setWindowed()
69             window.handle_resize()
70             window.restore()
71
72         self._widget_group.handle_resize(window.size)
73
74         new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
75             relative_mouse_pos[1] * window.size[1])
76         pygame.mouse.set_pos(new_mouse_pos)
77
78     def reload_shaders(self):
79         window.clear_all_effects()
80
81     for shader_type in SHADER_MAP[self._settings['shader']]:
82         window.set_effect(shader_type)
83
84     def reload_settings(self):
85         SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self.
86         _settings['primaryBoardColour'])
87         SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self.
88         _settings['secondaryBoardColour'])
89         SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.
90         BASE)
91         SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.
92         BASE)
93         SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['
94         musicVolume'])
95         SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume
96         '])
97         SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings[
98         'displayMode'])
99         SETTINGS_WIDGETS['shader_carousel'].set_to_key(self._settings['shader'])

```

```

90     SETTINGS_WIDGETS['particles_switch'].set_toggle_state(self._settings['
91         particles'])
92     SETTINGS_WIDGETS['opengl_switch'].set_toggle_state(self._settings['opengl'
93         ])
94
95     self.reload_shaders()
96     self.reload_display_mode()
97
98     def get_event(self, event):
99         widget_event = self._widget_group.process_event(event)
100
101         if widget_event is None:
102             if event.type == pygame.MOUSEBUTTONDOWN and self._colour_picker:
103                 self.remove_colour_picker()
104             return
105
106         match widget_event.type:
107             case SettingsEventType.VOLUME_SLIDER_SLIDE:
108                 return
109
110             case SettingsEventType.VOLUME_SLIDER_CLICK:
111                 if widget_event.volume_type == 'music':
112                     audio.set_music_volume(widget_event.volume)
113                     self._settings['musicVolume'] = widget_event.volume
114                 elif widget_event.volume_type == 'sfx':
115                     audio.set_sfx_volume(widget_event.volume)
116                     self._settings['sfxVolume'] = widget_event.volume
117
118             case SettingsEventType.DROPDOWN_CLICK:
119                 selected_word = SETTINGS_WIDGETS['display_mode_dropdown'].get_selected_word()
120
121                 if selected_word is None or selected_word == self._settings['
122                     displayMode']:
123
124                     self._settings['displayMode'] = selected_word
125
126                     self.reload_display_mode()
127
128             case SettingsEventType.MENU_CLICK:
129                 self.next = 'menu'
130                 self.done = True
131
132             case SettingsEventType.RESET_DEFAULT:
133                 self._settings = get_default_settings()
134                 self.reload_settings()
135
136             case SettingsEventType.RESET_USER:
137                 self._settings = get_user_settings()
138                 self.reload_settings()
139
140             case SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK | SettingsEventType
141                 .SECONDARY_COLOUR_BUTTON_CLICK:
142                 if self._colour_picker:
143                     self.remove_colour_picker()
144
145                 self.create_colour_picker(event.pos, widget_event.type)
146
147             case SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK | SettingsEventType
148                 .SECONDARY_COLOUR_PICKER_CLICK:
149                 if widget_event.colour:

```

```

146             r, g, b = widget_event.colour.rgb
147             hex.colour = f'0x{hex(r)[2:]}{hex(g)[2:]}{hex(b)[2:]}{hex(b)[2:]}.zfill(2)'}
148
149             if widget_event.type == SettingsEventType.
150                 PRIMARY_COLOUR_PICKER_CLICK:
151                     SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(widget_event.colour)
152                     SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.BASE)
153                     self._settings['primaryBoardColour'] = hex.colour
154             elif widget_event.type == SettingsEventType.
155                 SECONDARY_COLOUR_PICKER_CLICK:
156                     SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(widget_event.colour)
157                     SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.BASE)
158                     self._settings['secondaryBoardColour'] = hex.colour
159
160             case SettingsEventType.SHADER_PICKER_CLICK:
161                 self._settings['shader'] = widget_event.data
162                 self.reload_shaders()
163
164             case SettingsEventType.OPENGL_CLICK:
165                 self._settings['opengl'] = widget_event.toggled
166                 self.reload_shaders()
167
168             case SettingsEventType.PARTICLES_CLICK:
169                 self._settings['particles'] = widget_event.toggled
170
171     def draw(self):
172         self._widget_group.draw()

```

### B.23.2 widget\_dict.py

```

1  from data.widgets import *
2  from data.helpers.data_helpers import get_user_settings
3  from data.components.custom_event import CustomEvent
4  from data.utils.event_types import SettingsEventType
5  from data.utils.constants import SHADER_MAP
6  from data.utils.assets import GRAPHICS
7  from data.managers.theme import theme
8
9  user_settings = get_user_settings()
10 # font_size = text_width_to_font_size('Shaders (OPENGL GPU REQUIRED)', DEFAULT_FONT, 0.4 * window.screen.width)
11 FONT_SIZE = 21
12
13 carousel_widgets = {
14     key: Text(
15         relative_position=(0, 0),
16         relative_size=(0.25, 0.04),
17         margin=0,
18         text=key.replace('_', ' ').upper(),
19         fit_vertical=True,
20         border_width=0,
21         fill_colour=(0, 0, 0, 0),
22     ) for key in SHADER_MAP.keys()
23 }
24
25 reset_container = Rectangle(
26     relative_size=(0.2, 0.2),

```

```
27     relative_position=(0, 0),
28     fixed_position=(5, 5),
29     anchor_x='right',
30     anchor_y='bottom',
31 )
32
33 SETTINGS_WIDGETS = {
34     'default': [
35         reset_container,
36         ReactiveIconButton(
37             relative_position=(0, 0),
38             relative_size=(0.075, 0.075),
39             anchor_x='right',
40             scale_mode='height',
41             base_icon=GRAPHICS['home_base'],
42             hover_icon=GRAPHICS['home_hover'],
43             press_icon=GRAPHICS['home_press'],
44             fixed_position=(5, 5),
45             event=CustomEvent(SettingsEventType.MENU_CLICK)
46         ),
47         Text(
48             relative_position=(0.01, 0.1),
49             text='Display mode',
50             relative_size=(0.4, 0.04),
51             center=False,
52             border_width=0,
53             margin=0,
54             font_size=21,
55             fill_colour=(0, 0, 0, 0)
56         ),
57         Text(
58             relative_position=(0.01, 0.2),
59             text='Music',
60             relative_size=(0.4, 0.04),
61             center=False,
62             border_width=0,
63             margin=0,
64             font_size=21,
65             fill_colour=(0, 0, 0, 0)
66         ),
67         Text(
68             relative_position=(0.01, 0.3),
69             text='SFX',
70             relative_size=(0.4, 0.04),
71             center=False,
72             border_width=0,
73             margin=0,
74             font_size=21,
75             fill_colour=(0, 0, 0, 0)
76         ),
77         Text(
78             relative_position=(0.01, 0.4),
79             text='Primary board colour',
80             relative_size=(0.4, 0.04),
81             center=False,
82             border_width=0,
83             margin=0,
84             font_size=21,
85             fill_colour=(0, 0, 0, 0)
86         ),
87         Text(
88             relative_position=(0.01, 0.5),
```

```

89         text='Secondary board colour',
90         relative_size=(0.4, 0.04),
91         center=False,
92         border_width=0,
93         margin=0,
94         font_size=21,
95         fill_colour=(0, 0, 0, 0)
96     ),
97     Text(
98         relative_position=(0.01, 0.6),
99         text='Particles',
100        relative_size=(0.4, 0.04),
101        center=False,
102        border_width=0,
103        margin=0,
104        font_size=21,
105        fill_colour=(0, 0, 0, 0)
106    ),
107    Text(
108        relative_position=(0.01, 0.7),
109        text='Shaders (OPENGL GPU REQUIRED)',
110        relative_size=(0.4, 0.04),
111        center=False,
112        border_width=0,
113        margin=0,
114        font_size=21,
115        fill_colour=(0, 0, 0, 0)
116    ),
117    Text(
118        relative_position=(0.01, 0.8),
119        text='Super Secret Settings',
120        relative_size=(0.4, 0.04),
121        center=False,
122        border_width=0,
123        margin=0,
124        font_size=21,
125        fill_colour=(0, 0, 0, 0)
126    ),
127    TextButton(
128        parent=reset_container,
129        relative_position=(0, 0),
130        relative_size=(1, 0.5),
131        fit_vertical=False,
132        margin=10,
133        text='DISCARD CHANGES',
134        text_colour=theme['textSecondary'],
135        event=CustomEvent(SettingsEventType.RESET_USER)
136    ),
137    TextButton(
138        parent=reset_container,
139        relative_position=(0, 0.5),
140        relative_size=(1, 0.5),
141        fit_vertical=False,
142        margin=10,
143        text='RESET TO DEFAULT',
144        text_colour=theme['textSecondary'],
145        event=CustomEvent(SettingsEventType.RESET_DEFAULT)
146    )
147 ],
148 'display_mode_dropdown':
149     Dropdown(
150         relative_position=(0.4, 0.1),

```

```

151         relative_width=0.2,
152         word_list=['fullscreen', 'windowed'],
153         fill_colour=(255, 100, 100),
154         event=CustomEvent(SettingsEventType.DROPODOWN_CLICK)
155     ),
156     'primary_colour_button':
157     ColourButton(
158         relative_position=(0.4, 0.4),
159         relative_size=(0.08, 0.05),
160         fill_colour=user_settings['primaryBoardColour'],
161         border_width=5,
162         event=CustomEvent(SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK)
163     ),
164     'secondary_colour_button':
165     ColourButton(
166         relative_position=(0.4, 0.5),
167         relative_size=(0.08, 0.05),
168         fill_colour=user_settings['secondaryBoardColour'],
169         border_width=5,
170         event=CustomEvent(SettingsEventType.SECONDARY_COLOUR_BUTTON_CLICK)
171     ),
172     'music_volume_slider':
173     VolumeSlider(
174         relative_position=(0.4, 0.2),
175         relative_length=(0.5),
176         default_volume=user_settings['musicVolume'],
177         border_width=5,
178         volume_type='music'
179     ),
180     'sfx_volume_slider':
181     VolumeSlider(
182         relative_position=(0.4, 0.3),
183         relative_length=(0.5),
184         default_volume=user_settings['sfxVolume'],
185         border_width=5,
186         volume_type='sfx'
187     ),
188     'shader_carousel':
189     Carousel(
190         relative_position = (0.4, 0.8),
191         margin=5,
192         border_width=0,
193         fill_colour=(0, 0, 0, 0),
194         widgets_dict=carousel_widgets,
195         event=CustomEvent(SettingsEventType.SHADER_PICKER_CLICK),
196     ),
197     'particles_switch':
198     Switch(
199         relative_position=(0.4, 0.6),
200         relative_height=0.04,
201         event=CustomEvent(SettingsEventType.PARTICLES_CLICK)
202     ),
203     'opengl_switch':
204     Switch(
205         relative_position=(0.4, 0.7),
206         relative_height=0.04,
207         event=CustomEvent(SettingsEventType.OPENGL_CLICK)
208     ),
209 }
```

## B.24 data\utils

## B.24.1 assets.py

```
1 from pathlib import Path
2 from data.helpers.load_helpers import *
3
4 module_path = Path(__file__).parent
5 GRAPHICS = load_all_gfx((module_path / '../../../../../resources/graphics').resolve())
6 FONTS = load_all_fonts((module_path / '../../../../../resources/fonts').resolve())
7 SFX = load_all_sfx((module_path / '../../../../../resources/sfx').resolve())
8 MUSIC = load_all_music((module_path / '../../../../../resources/music').resolve())
9
10 DEFAULT_FONT = FONTS['vhs-gothic']
11 DEFAULT_FONT.strong = True
12 DEFAULT_FONT.strength = 0.05
```

## B.24.2 constants.py

```

32
33 BLUE_BUTTON_COLOURS = {
34     WidgetState.BASE: ['0x1c2638', '0x23495d', '0x39707a', '0x95e0cc'],
35     WidgetState.HOVER: ['0xdaf2e9', '0x23495d', '0x39707a', '0x95e0cc'],
36     WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
37 }
38
39 INPUT_COLOURS = {
40     WidgetState.BASE: ['0x1c2638', '0x39707a', '0x23495d', '0x95e0cc'],
41     WidgetState.HOVER: ['0xdaf2e9', '0x39707a', '0x23495d', '0x95e0cc'],
42     WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0x1c2638', '0x39707a']
43 }
44
45 RED_BUTTON_COLOURS = {
46     WidgetState.BASE: ['0x000000', '0x1c2638', '0x9b222b', '0xf14e52'],
47     WidgetState.HOVER: ['0xdaf2e9', '0x1c2638', '0x9b222b', '0xf14e52'],
48     WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0xf14e52', '0x95e0cc']
49 }
50
51 LOCKED_RED_BUTTON_COLOURS = {
52     WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
53     WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
54     WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0xf14e52']
55 }
56
57 LOCKED_BLUE_BUTTON_COLOURS = {
58     WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
59     WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
60     WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
61 }

```

### B.24.3 enums.py

```

1 from enum import IntEnum, StrEnum, auto
2
3 class CursorMode(IntEnum):
4     ARROW = auto()
5     IBEAM = auto()
6     OPENHAND = auto()
7     CLOSEDHAND = auto()
8     NO = auto()
9
10 class ShaderType(StrEnum):
11     BASE = auto()
12     SHAKE = auto()
13     BLOOM = auto()
14     GRayscale = auto()
15     CRT = auto()
16     RAYS = auto()
17     CHROMATIC_ABBREVIATION = auto()
18     BACKGROUND_WAVES = auto()
19     BACKGROUND_BALATRO = auto()
20     BACKGROUND_LASERS = auto()
21     BACKGROUND_GRADIENT = auto()
22     BACKGROUND_NONE = auto()
23
24     _BLUR = auto()
25     _HIGHLIGHT_BRIGHTNESS = auto()
26     _HIGHLIGHT_COLOUR = auto()
27     _CALIBRATE = auto()
28     _LIGHTMAP = auto()
29     _SHADOWMAP = auto()

```

```
30     _OCCULTION = auto()
31     _BLEND = auto()
32     _CROP = auto()
33
34 class TranspositionFlag(StrEnum):
35     LOWER = auto()
36     EXACT = auto()
37     UPPER = auto()
38
39 class Miscellaneous(StrEnum):
40     PLACEHOLDER = auto()
41     DRAW = auto()
42
43 class WidgetState(StrEnum):
44     BASE = auto()
45     HOVER = auto()
46     PRESS = auto()
47
48 class StatusText(StrEnum):
49     PLAYER_MOVE = auto()
50     CPU_MOVE = auto()
51     WIN = auto()
52     DRAW = auto()
53
54 class Colour(IntEnum):
55     BLUE = 0
56     RED = 1
57
58     def get_flipped_colour(self):
59         if self == Colour.BLUE:
60             return Colour.RED
61         elif self == Colour.RED:
62             return Colour.BLUE
63
64 class Piece(StrEnum):
65     SPHINX = 's'
66     PYRAMID = 'p'
67     ANUBIS = 'n'
68     SCARAB = 'r'
69     PHAROAH = 'f'
70
71 class Score(IntEnum):
72     PHAROAH = 0
73     SPHINX = 0
74     PYRAMID = 100
75     ANUBIS = 110
76     SCARAB = 200
77
78     MOVE = 4
79     POSITION = 11
80     PHAROAH_SAFETY = 31
81     CHECKMATE = 100000
82     INFINITE = 6969696969
83
84 class Rank(IntEnum):
85     ONE = 0
86     TWO = 1
87     THREE = 2
88     FOUR = 3
89     FIVE = 4
90     SIX = 5
91     SEVEN = 6
```

```
92     EIGHT = 7
93
94 class File(IntEnum):
95     A = 0
96     B = 1
97     C = 2
98     D = 3
99     E = 4
100    F = 5
101    G = 6
102    H = 7
103    I = 8
104    J = 9
105
106 class Rotation(StrEnum):
107     UP = 'a'
108     RIGHT = 'b'
109     DOWN = 'c'
110     LEFT = 'd'
111
112     def to_angle(self):
113         if self == Rotation.UP:
114             return 0
115         elif self == Rotation.RIGHT:
116             return 270
117         elif self == Rotation.DOWN:
118             return 180
119         elif self == Rotation.LEFT:
120             return 90
121
122     def get_clockwise(self):
123         if self == Rotation.UP:
124             return Rotation.RIGHT
125         elif self == Rotation.RIGHT:
126             return Rotation.DOWN
127         elif self == Rotation.DOWN:
128             return Rotation.LEFT
129         elif self == Rotation.LEFT:
130             return Rotation.UP
131
132     def get_anticlockwise(self):
133         if self == Rotation.UP:
134             return Rotation.LEFT
135         elif self == Rotation.RIGHT:
136             return Rotation.UP
137         elif self == Rotation.DOWN:
138             return Rotation.RIGHT
139         elif self == Rotation.LEFT:
140             return Rotation.DOWN
141
142     def get_opposite(self):
143         return self.get_clockwise().get_clockwise()
144
145 class RotationIndex(IntEnum):
146     FIRSTBIT = 0
147     SECONDBIT = 1
148
149 class RotationDirection(StrEnum):
150     CLOCKWISE = 'cw'
151     ANTICLOCKWISE = 'acw'
152
153     def get_opposite(self):
```

```

154     if self == RotationDirection.CLOCKWISE:
155         return RotationDirection.ANTICLOCKWISE
156     elif self == RotationDirection.ANTICLOCKWISE:
157         return RotationDirection.CLOCKWISE
158
159 class MoveType(StrEnum):
160     MOVE = 'm'
161     ROTATE = 'r'
162
163 class LaserType(IntEnum):
164     END = 0
165     STRAIGHT = 1
166     CORNER = 2
167
168 class LaserDirection(IntEnum):
169     FROM_TOP = 1
170     FROM_RIGHT = 2
171     FROM_BOTTOM = 3
172     FROM_LEFT = 4

```

#### B.24.4 event\_types.py

```

1 from enum import IntEnum, StrEnum, auto
2
3 class EditorEventType(StrEnum):
4     MENU_CLICK = auto()
5     PICK_PIECE_CLICK = auto()
6     ROTATE_PIECE_CLICK = auto()
7     COPY_CLICK = auto()
8     EMPTY_CLICK = auto()
9     RESET_CLICK = auto()
10    BLUE_START_CLICK = auto()
11    RED_START_CLICK = auto()
12    START_CLICK = auto()
13    CONFIG_CLICK = auto()
14    ERASE_CLICK = auto()
15    MOVE_CLICK = auto()
16    HELP_CLICK = auto()
17
18 class ReviewEventType(StrEnum):
19     MENU_CLICK = auto()
20     PREVIOUS_CLICK = auto()
21     NEXT_CLICK = auto()
22     HELP_CLICK = auto()
23
24 class BrowserEventType(StrEnum):
25     MENU_CLICK = auto()
26     BROWSER_STRIP_CLICK = auto()
27     COPY_CLICK = auto()
28     DELETE_CLICK = auto()
29     REVIEW_CLICK = auto()
30     FILTER_COLUMN_CLICK = auto()
31     FILTER_ASCEND_CLICK = auto()
32     PAGE_CLICK = auto()
33     HELP_CLICK = auto()
34
35 class GameEventType(StrEnum):
36     BOARD_CLICK = auto()
37     PIECE_CLICK = auto()
38     PAUSE_CLICK = auto()
39     MENU_CLICK = auto()
40     GAME_CLICK = auto()

```

```

41     HELP_CLICK = auto()
42     TUTORIAL_CLICK = auto()
43     RESIGN_CLICK = auto()
44     DRAW_CLICK = auto()
45     REVIEW_CLICK = auto()
46     PIECE_DROP = auto()
47     UPDATE_PIECES = auto()
48     ROTATE_PIECE = auto()
49     SET LASER = auto()
50     TIMER_END = auto()

51
52 class MenuEventType(StrEnum):
53     CONFIG_CLICK = auto()
54     SETTINGS_CLICK = auto()
55     BROWSER_CLICK = auto()
56     QUIT_CLICK = auto()
57     CREDITS_CLICK = auto()

58
59 class SettingsEventType(StrEnum):
60     RESET_DEFAULT = auto()
61     RESET_USER = auto()
62     MENU_CLICK = auto()
63     COLOUR_SLIDER_SLIDE = auto()
64     COLOUR_SLIDER_CLICK = auto()
65     COLOUR_PICKER_HOVER = auto()
66     PRIMARY_COLOUR_PICKER_CLICK = auto()
67     SECONDARY_COLOUR_PICKER_CLICK = auto()
68     PRIMARY_COLOUR_BUTTON_CLICK = auto()
69     SECONDARY_COLOUR_BUTTON_CLICK = auto()
70     VOLUME_SLIDER_SLIDE = auto()
71     VOLUME_SLIDER_CLICK = auto()
72     SHADER_PICKER_CLICK = auto()
73     OPENGL_CLICK = auto()
74     DROPODOWN_CLICK = auto()
75     PARTICLES_CLICK = auto()

76
77 class ConfigEventType(StrEnum):
78     GAME_CLICK = auto()
79     MENU_CLICK = auto()
80     FEN_STRING_TYPE = auto()
81     TIME_TYPE = auto()
82     TIME_CLICK = auto()
83     PVP_CLICK = auto()
84     PVC_CLICK = auto()
85     CPU_DEPTH_CLICK = auto()
86     PRESET_CLICK = auto()
87     SETUP_CLICK = auto()
88     COLOUR_CLICK = auto()
89     HELP_CLICK = auto()

```

## B.25 data\ widgets

### B.25.1 board\_thumbnail.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.chessboard import Chessboard
4 from data.states.game.components.piece_group import PieceGroup
5 from data.states.game.components.bitboard_collection import BitboardCollection
6
7 class BoardThumbnail(_Widget):

```

```

8     def __init__(self, relative_width, fen_string='', **kwargs):
9         super().__init__(relative_size=(relative_width, relative_width * 0.8), **
10                      kwargs)
11
12         self._board = Chessboard(
13             parent=self._parent,
14             relative_position=(0, 0),
15             scale_mode=kwargs.get('scale_mode'),
16             relative_width=relative_width
17         )
18
19         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
20
21         self.initialise_board(fen_string)
22         self.set_image()
23         self.set_geometry()
24
25     def initialise_board(self, fen_string):
26         if len(fen_string) == 0:
27             piece_list = []
28         else:
29             piece_list = BitboardCollection(fen_string).convert_to_piece_list()
30
31         self._piece_group = PieceGroup()
32         self._piece_group.initialise_pieces(piece_list, (0, 0), self.size)
33
34         self._board.refresh_board()
35         self.set_image()
36
37     def set_image(self):
38         self.image = pygame.transform.scale(self._empty_surface, self.size)
39
40         self._board.set_image()
41         self.image.blit(self._board.image, (0, 0))
42
43         self._piece_group.draw(self.image)
44
45     def set_geometry(self):
46         super().__set_geometry__()
47         self._board.set_geometry()
48
49     def set_surface_size(self, new_surface_size):
50         super().__set_surface_size(new_surface_size)
51         self._board.set_surface_size(new_surface_size)
52         self._piece_group.handle_resize((0, 0), self.size)
53
54     def process_event(self, event):
55         pass

```

### B.25.2 board\_thumbnail\_button.py

```

1  from data.widgets.bases.pressable import _Pressable
2  from data.widgets.board_thumbnail import BoardThumbnail
3  from data.utils.constants import WidgetState
4  from data.components.custom_event import CustomEvent
5
6  class BoardThumbnailButton(_Pressable, BoardThumbnail):
7      def __init__(self, event, **kwargs):
8          _Pressable.__init__(
9              self,
10              event=CustomEvent(**vars(event), fen_string=kwargs.get('fen_string')),
11              hover_func=lambda: self.set_state_colour(WidgetState.HOVER),

```

```
12         down_func=lambda: self.set_state_colour(WidgetState.PRESS),
13         up_func=lambda: self.set_state_colour(WidgetState.BASE),
14     )
15     BoardThumbnail.__init__(self, **kwargs)
16
17     self.initialise_new_colours(self._fill_colour)
18     self.set_state_colour(WidgetState.BASE)
```

### B.25.3 browser item.py

```
1 import pygame
2 from data.helpers.font_helpers import text_width_to_font_size
3 from data.helpers.browser_helpers import get_winner_string
4 from data.widgets.board_thumbnail import BoardThumbnail
5 from data.helpers.asset_helpers import scale_and_cache
6 from data.widgets.bases.widget import _Widget
7
8 FONT_DIVISION = 7
9
10 class BrowserItem(_Widget):
11     def __init__(self, relative_width, game, **kwargs):
12         super().__init__(relative_size=(relative_width, relative_width * 2),
13                         scale_mode='height', **kwargs)
14
15         self._relative_font_size = text_width_to_font_size('YYYY-MM-DD HH:MM:SS',
16                                                          self._font, self.size[0]) / self.surface_size[1]
17
18         self._game = game
19         self._board_thumbnail = BoardThumbnail(
20             relative_position=(0, 0),
21             scale_mode='height',
22             relative_width=relative_width,
23             fen_string=self._game['final_fen_string'])
24
25         self.set_image()
26         self.set_geometry()
27
28     def get_text_to_render(self):
29         depth_to_text = {
30             2: 'EASY',
31             3: 'MEDIUM',
32             4: 'HARD'
33         }
34
35         format_moves = lambda no_of_moves: int(no_of_moves / 2) if (no_of_moves /
36         2 % 1 == 0) else round(no_of_moves / 2, 1)
37
38         if self._game['cpu_enabled'] == 1:
39             depth_text = depth_to_text[self._game['cpu_depth']]
40             cpu_text = f'PVC ({depth_text})'
41         else:
42             cpu_text = 'PVP'
43
44         return [
45             cpu_text,
46             self._game['created_dt'].strftime('%Y-%m-%d %H:%M:%S'),
47             f'WINNER: {get_winner_string(self._game['winner'])}',
48             f'NO. MOVES: {format_moves(self._game['number_of_ply'])}'
49         ]
50
51     def set_image(self):
```

```

50         self.image = pygame.Surface(self.size, pygame.SRCALPHA)
51         resized_board = scale_and_cache(self._board_thumbnail.image, (self.size
52 [0], self.size[0] * 0.8))
53         self.image.blit(resized_board, (0, 0))
54
55         get_line_y = lambda line: (self.size[0] * 0.8) + ((self.size[0] * 0.8) /
56 FONT_DIVISION) * (line + 0.5)
57
58         text_to_render = self.get_text_to_render()
59
60         for index, text in enumerate(text_to_render):
61             self._font.render_to(self.image, (0, get_line_y(index)), text, fgcolor
62 =self._text_colour, size=self.font_size)
63
64     def process_event(self, event):
65         pass

```

#### B.25.4 browser\_strip.py

```

1 import pygame
2 from data.components.custom_event import CustomEvent
3 from data.utils.event_types import BrowserEventType
4 from data.widgets.browser_item import BrowserItem
5 from data.widgets.bases.widget import _Widget
6
7 WIDTH_FACTOR = 0.3
8
9 class BrowserStrip(_Widget):
10     def __init__(self, relative_height, games_list, **kwargs):
11         super().__init__(relative_size=None, **kwargs)
12         self._relative_item_width = relative_height / 2
13         self._get_rect = None
14
15         self._games_list = []
16         self._items_list = []
17         self._selected_index = None
18
19         self.initialise_games_list(games_list)
20
21     @property
22     def item_width(self):
23         return self._relative_item_width * self.surface_size[1]
24
25     @property
26     def size(self):
27         if self._get_rect:
28             height = self._get_rect().height
29         else:
30             height = 0
31         width = max(0, len(self._games_list) * (self.item_width + self.margin) +
32         self.margin)
33
34         return (width, height)
35
36     def register_get_rect(self, get_rect_func):
37         self._get_rect = get_rect_func
38
39     def initialise_games_list(self, games_list):
40         self._items_list = []
41         self._games_list = games_list
42         self._selected_index = None
43
44

```

```

43         for game in games_list:
44             browser_item = BrowserItem(relative_position=(0, 0), game=game,
45                                         relative_width=self._relative_item_width)
46             self._items_list.append(browser_item)
47
48         self.set_image()
49         self.set_geometry()
50
51     def set_image(self):
52         self.image = pygame.Surface(self.size, pygame.SRCALPHA)
53         browser_list = []
54
55         for index, item in enumerate(self._items_list):
56             item.set_image()
57             browser_list.append((item.image, (index * (self.item_width + self.
58 margin) + self.margin, self.margin)))
59
60         self.image.blit(browser_list)
61
62         if self._selected_index is not None:
63             border_position = (self._selected_index * (self.item_width + self.
64 margin), 0)
65             border_size = (self.item_width + 2 * self.margin, self.size[1])
66             pygame.draw.rect(self.image, (255, 255, 255), (*border_position, *
67 border_size), width=int(self.item_width / 20))
68
69     def set_geometry(self):
70         super().set_geometry()
71         for item in self._items_list:
72             item.set_geometry()
73
74     def set_surface_size(self, new_surface_size):
75         super().set_surface_size(new_surface_size)
76
77         for item in self._items_list:
78             item.set_surface_size(new_surface_size)
79
80     def process_event(self, event, scrolled_pos):
81         parent_pos = self._get_rect().topleft
82         self.rect.topleft = parent_pos
83
84         if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
85             self._selected_index = None
86             self.set_image()
87             return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
88 selected_index=None)
89
90         if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.
91 pos):
92             relative_mouse_pos = (event.pos[0] - parent_pos[0], event.pos[1] -
93 parent_pos[1])
94             self._selected_index = int(max(0, (relative_mouse_pos[0] - self.margin
95 ) // (self.item_width + self.margin)))
96             self.set_image()
97             return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
98 selected_index=self._selected_index)

```

### B.25.5 carousel.py

```

1 import pygame
2 from data.widgets.reactive_icon_button import ReactiveIconButton
3 from data.components.custom_event import CustomEvent

```

```

4  from data.widgets.bases.circular import _Circular
5  from data.widgets.bases.widget import _Widget
6  from data.utils.assets import GRAPHICS, SFX
7  from data.utils.enums import Miscellaneous
8
9  class Carousel(_Circular, _Widget):
10     def __init__(self, event, widgets_dict, **kwargs):
11         _Circular.__init__(self, items_dict=widgets_dict)
12         _Widget.__init__(self, relative_size=None, **kwargs)
13
14         max_widget_size = (
15             max([widget.rect.width for widget in widgets_dict.values()]),
16             max([widget.rect.height for widget in widgets_dict.values()])
17         )
18
19         self._relative_max_widget_size = (max_widget_size[0] / self.surface_size
20 [1], max_widget_size[1] / self.surface_size[1])
21         self._relative_size = ((max_widget_size[0] + 2 * (self.margin + self.
22 arrow_size[0])) / self.surface_size[1], (max_widget_size[1]) / self.
23 surface_size[1])
24
25         self._left_arrow = ReactiveIconButton(
26             relative_position=(0, 0),
27             relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
28             scale_mode='height',
29             base_icon=GRAPHICS['left_arrow_base'],
30             hover_icon=GRAPHICS['left_arrow_hover'],
31             press_icon=GRAPHICS['left_arrow_press'],
32             event=CustomEvent(Miscellaneous.PLACEHOLDER),
33             sfx=SFX['carousel_click']
34         )
35         self._right_arrow = ReactiveIconButton(
36             relative_position=(0, 0),
37             relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
38             scale_mode='height',
39             base_icon=GRAPHICS['right_arrow_base'],
40             hover_icon=GRAPHICS['right_arrow_hover'],
41             press_icon=GRAPHICS['right_arrow_press'],
42             event=CustomEvent(Miscellaneous.PLACEHOLDER),
43             sfx=SFX['carousel_click']
44         )
45
46         self._event = event
47         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
48
49         self.set_image()
50         self.set_geometry()
51
52     @property
53     def max_widget_size(self):
54         return (self._relative_max_widget_size[0] * self.surface_size[1], self.
55         _relative_max_widget_size[1] * self.surface_size[1])
56
57     @property
58     def arrow_size(self):
59         height = self.max_widget_size[1] * 0.75
60         width = (GRAPHICS['left_arrow_base'].width / GRAPHICS['left_arrow_base'].
61         height) * height
62         return (width, height)
63
64     @property
65     def size(self):

```

```
1      return ((self.arrow_size[0] + self.margin) * 2 + self.max_widget_size[0],
2      self.max_widget_size[1])
3
4      @property
5      def left_arrow_position(self):
6          return (0, (self.size[1] - self.arrow_size[1]) / 2)
7
8      @property
9      def right_arrow_position(self):
10         return (self.size[0] - self.arrow_size[0], (self.size[1] - self.arrow_size
11 [1]) / 2)
12
13     def set_image(self):
14         self.image = pygame.transform.scale(self._empty_surface, self.size)
15         self.image.fill(self._fill_colour)
16
17         if self.border_width:
18             pygame.draw.rect(self.image, self._border_colour, (0, 0, *self.size),
19             width=int(self.border_width), border_radius=int(self.border_radius))
20
21         self._left_arrow.set_image()
22         self.image.blit(self._left_arrow.image, self.left_arrow_position)
23
24         self.current_item.set_image()
25         self.image.blit(self.current_item.image, ((self.size[0] - self.
26 current_item.rect.size[0]) / 2, (self.size[1] - self.current_item.rect.size
27 [1]) / 2))
28
29         self._right_arrow.set_image()
30         self.image.blit(self._right_arrow.image, self.right_arrow_position)
31
32     def set_geometry(self):
33         super().set_geometry()
34
35         self.current_item.set_geometry()
36         self._left_arrow.set_geometry()
37         self._right_arrow.set_geometry()
38
39         self.current_item.rect.center = self.rect.center
40         self._left_arrow.rect.topleft = (self.position[0] + self.
41 left_arrow_position[0], self.position[1] + self.left_arrow_position[1])
42         self._right_arrow.rect.topleft = (self.position[0] + self.
43 right_arrow_position[0], self.position[1] + self.right_arrow_position[1])
44
45     def set_surface_size(self, new_surface_size):
46         super().set_surface_size(new_surface_size)
47         self._left_arrow.set_surface_size(new_surface_size)
48         self._right_arrow.set_surface_size(new_surface_size)
49
50         for item in self._items_dict.values():
51             item.set_surface_size(new_surface_size)
52
53     def process_event(self, event):
54         self.current_item.process_event(event)
55         left_arrow_event = self._left_arrow.process_event(event)
56         right_arrow_event = self._right_arrow.process_event(event)
57
58         if left_arrow_event:
59             self.set_previous_item()
60             self.current_item.set_surface_size(self._raw_surface_size)
61
62         elif right_arrow_event:
```

```

116         self.set_next_item()
117         self.current_item.set_surface_size(self._raw_surface_size)
118
119     if left_arrow_event or right_arrow_event:
120         self.set_image()
121         self.set_geometry()
122
123     return CustomEvent(**vars(self._event), data=self.current_key)
124
125 elif event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
126 MOUSEMOTION]:
127     self.set_image()
128     self.set_geometry()

```

### B.25.6 chessboard.py

```

1 import pygame
2 from data.helpers.data_helpers import get_user_settings
3 from data.helpers.board_helpers import create_board
4 from data.widgets.bases.widget import _Widget
5 from data.utils.enums import CursorMode
6 from data.managers.cursor import cursor
7
8 class Chessboard(_Widget):
9     def __init__(self, relative_width, change_cursor=True, **kwargs):
10         super().__init__(relative_size=(relative_width, relative_width * 0.8), **
11                         kwargs)
12
13         self._board_surface = None
14         self._change_cursor = change_cursor
15         self._cursor_is_hand = False
16
17         self.refresh_board()
18         self.set_image()
19         self.set_geometry()
20
21     def refresh_board(self):
22         user_settings = get_user_settings()
23         self._board_surface = create_board(self.size, user_settings['
24             primaryBoardColour'], user_settings['secondaryBoardColour'])
25
26         self.set_image()
27
28     def set_image(self):
29         self.image = pygame.transform.smoothscale(self._board_surface, self.size)
30
31     def process_event(self, event):
32         if self._change_cursor and event.type in [pygame.MOUSEMOTION, pygame.
33             MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN]:
34             current_cursor = cursor.get_mode()
35
36             if self.rect.collidepoint(event.pos):
37                 if current_cursor == CursorMode.ARROW:
38                     cursor.set_mode(CursorMode.OPENHAND)
39                 elif current_cursor == CursorMode.OPENHAND and (pygame.mouse.
40                     get_pressed()[0] is True or event.type == pygame.MOUSEBUTTONDOWN):
41                     cursor.set_mode(CursorMode.CLOSEDHAND)
42                 elif current_cursor == CursorMode.CLOSEDHAND and (pygame.mouse.
43                     get_pressed()[0] is False or event.type == pygame.MOUSEBUTTONUP):
44                     cursor.set_mode(CursorMode.OPENHAND)
45             else:

```

```

41     if current_cursor == CursorMode.OPENHAND or (current_cursor ==
42         CursorMode.CLOSEDHAND and event.type == pygame.MOUSEBUTTONUP):
43             cursor.set_mode(CursorMode.ARROW)

```

### B.25.7 colour\_button.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.utils.constants import WidgetState
5
6 class ColourButton(_Pressable, _Widget):
7     def __init__(self, event, **kwargs):
8         _Pressable.__init__(
9             self,
10            event=event,
11            hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
12            down_func=lambda: self.set_state_colour(WidgetState.PRESS),
13            up_func=lambda: self.set_state_colour(WidgetState.BASE),
14            sfx=None
15        )
16        _Widget.__init__(self, **kwargs)
17
18        self._empty_surface = pygame.Surface(self.size)
19
20        self.initialise_new_colours(self._fill_colour)
21        self.set_state_colour(WidgetState.BASE)
22
23        self.set_image()
24        self.set_geometry()
25
26    def set_image(self):
27        self.image = pygame.transform.scale(self._empty_surface, self.size)
28        self.image.fill(self._fill_colour)
29        pygame.draw.rect(self.image, self._border_colour, (0, 0, self.size[0],
self.size[1]), width=int(self.border_width))

```

### B.25.8 colour\_display.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3
4 class _ColourDisplay(_Widget):
5     def __init__(self, **kwargs):
6         super().__init__(**kwargs)
7
8         self._colour = None
9
10        self._empty_surface = pygame.Surface(self.size)
11
12    def set_colour(self, new_colour):
13        self._colour = new_colour
14
15    def set_image(self):
16        self.image = pygame.transform.scale(self._empty_surface, self.size)
17        self.image.fill(self._colour)
18
19    def process_event(self, event):
20        pass

```

### B.25.9 colour\_picker.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.colour_square import _ColourSquare
4 from data.widgets.colour_slider import _ColourSlider
5 from data.widgets.colour_display import _ColourDisplay
6 from data.components.custom_event import CustomEvent
7
8 class ColourPicker(_Widget):
9     def __init__(self, relative_width, event_type, **kwargs):
10         super().__init__(relative_size=(relative_width, relative_width),
11                         scale_mode='width', **kwargs)
12
13         self.image = pygame.Surface(self.size)
14         self.rect = self.image.get_rect()
15
16         self._square = _ColourSquare(
17             parent=self,
18             relative_position=(0.1, 0.1),
19             relative_width=0.5,
20             event_type=event_type
21         )
22         self._square.set_colour(kwargs.get('selected_colour'))
23
24         self._slider = _ColourSlider(
25             parent=self,
26             relative_position=(0.0, 0.7),
27             relative_width=1.0,
28             border_width=self.border_width,
29             border_colour=self._border_colour
30         )
31         self._slider.set_colour(kwargs.get('selected_colour'))
32
33         self._display = _ColourDisplay(
34             parent=self,
35             relative_position=(0.7, 0.1),
36             relative_size=(0.2, 0.5)
37         )
38         self._display.set_colour(kwargs.get('selected_colour'))
39
40         self._event_type = event_type
41         self._hover_event_type = event_type
42
43         self.set_image()
44         self.set_geometry()
45
46     def global_to_relative_pos(self, global_pos):
47         return (global_pos[0] - self.position[0], global_pos[1] - self.position[1])
48
49     def set_image(self):
50         self.image = pygame.Surface(self.size)
51         self.image.fill(self._fill_colour)
52
53         self._square.set_image()
54         self._square.set_geometry()
55         self.image.blit(self._square.image, self.global_to_relative_pos(self._square.position))
56
57         self._slider.set_image()

```

```

57         self._slider.set_geometry()
58         self.image.blit(self._slider.image, self.global_to_relative_pos(self.
59                         _slider.position))
60
60         self._display.set_image()
61         self._display.set_geometry()
62         self.image.blit(self._display.image, self.global_to_relative_pos(self.
63                         _display.position))
64
64         pygame.draw.rect(self.image, self._border_colour, (0, 0, self.size[0],
65                         self.size[1]), width=int(self.border_width))
65
66     def set_surface_size(self, new_surface_size):
67         super().set_surface_size(new_surface_size)
68         self._square.set_surface_size(self.size)
69         self._slider.set_surface_size(self.size)
70         self._display.set_surface_size(self.size)
71
72     def get_picker_position(self):
73         return self.position
74
75     def process_event(self, event):
76         slider_colour = self._slider.process_event(event)
77         square_colour = self._square.process_event(event)
78
79         if square_colour:
80             self._display.set_colour(square_colour)
81             self.set_image()
82
83         if slider_colour:
84             self._square.set_colour(slider_colour)
85             self.set_image()
86
87         if event.type in [pygame.MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN, pygame.
88                         MOUSEMOTION] and self.rect.collidepoint(event.pos):
89             return CustomEvent(self._event_type, colour=square_colour)

```

### B.25.10 colour\_slider.py

See Section 3.4.4.

### B.25.11 colour\_square.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.helpers.widget_helpers import create_square_gradient
4
5 class _ColourSquare(_Widget):
6     def __init__(self, relative_width, **kwargs):
7         super().__init__(relative_size=(relative_width, relative_width),
8                         scale_mode='width', **kwargs)
8
9         self._colour = None
10
11     def set_colour(self, new_colour):
12         self._colour = pygame.Color(new_colour)
13
14     def get_colour(self):
15         return self._colour
16

```

```

17     def set_image(self):
18         self.image = create_square_gradient(side_length=self.size[0], colour=self._colour)
19
20     def process_event(self, event):
21         if event.type == pygame.MOUSEBUTTONDOWN:
22             relative_mouse_pos = (event.pos[0] - self.position[0], event.pos[1] - self.position[1])
23
24             if (
25                 0 > relative_mouse_pos[0] or
26                 self.size[0] < relative_mouse_pos[0] or
27                 0 > relative_mouse_pos[1] or
28                 self.size[1] < relative_mouse_pos[1]
29             ): return None
30
31             self.set_colour(self.image.get_at(relative_mouse_pos))
32
33         return self._colour
34
35     return None

```

### B.25.12 dropdown.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.utils.constants import WidgetState
5 from data.helpers.data_helpers import get_user_settings
6 from data.helpers.font_helpers import text_width_to_font_size,
7     text_height_to_font_size
8 from data.utils.assets import GRAPHICS
9
10 user_settings = get_user_settings()
11
12 class Dropdown(_Pressable, _Widget):
13     def __init__(self, word_list, event=None, **kwargs):
14         _Pressable.__init__(
15             self,
16             event=event,
17             hover_func=self.hover_func,
18             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
19             up_func=self.up_func,
20             sfx=None
21         )
22         _Widget.__init__(self, relative_size=None, **kwargs)
23
24         if kwargs.get('relative_width'):
25             self._relative_font_size = text_width_to_font_size(max(word_list, key=len), self._font, kwargs.get('relative_width') * self.surface_size[0] - self.margin) / self.surface_size[1]
26         elif kwargs.get('relative_height'):
27             self._relative_font_size = text_height_to_font_size(max(word_list, key=len), self._font, kwargs.get('relative_height') * self.surface_size[1] - self.margin) / self.surface_size[1]
28
29         self._word_list = [word_list[0].capitalize()]
30         self._word_list_copy = [word.capitalize() for word in word_list]
31
32         self._expanded = False
33         self._hovered_index = None

```

```

34         self._empty_surface = pygame.Surface((0, 0))
35         self._background_colour = self._fill_colour
36
37         self.initialise_new_colours(self._fill_colour)
38         self.set_state_colour(WidgetState.BASE)
39
40         self.set_image()
41         self.set_geometry()
42
43     @property
44     def size(self):
45         max_word = sorted(self._word_list_copy, key=len)[-1]
46         max_word_rect = self._font.get_rect(max_word, size=self.font_size)
47         all_words_rect = pygame.Rect(0, 0, max_word_rect.size[0], (max_word_rect.
48             size[1] * len(self._word_list) + (self.margin * (len(self._word_list) - 1))))
49         all_words_rect = all_words_rect.inflate(2 * self.margin, 2 * self.margin)
50         return (all_words_rect.size[0] + max_word_rect.size[1], all_words_rect.
51             size[1])
52
53     def get_selected_word(self):
54         return self._word_list[0].lower()
55
56     def toggle_expanded(self):
57         if self._expanded:
58             self._word_list = [self._word_list_copy[0]]
59         else:
60             self._word_list = [*self._word_list_copy]
61
62         self._expanded = not(self._expanded)
63
64     def hover_func(self):
65         mouse_position = pygame.mouse.get_pos()
66         relative_position = (mouse_position[0] - self.position[0], mouse_position
67             [1] - self.position[1])
68         self._hovered_index = self.calculate_hovered_index(relative_position)
69         self.set_state_colour(WidgetState.HOVER)
70
71     def set_selected_word(self, word):
72         index = self._word_list_copy.index(word.capitalize())
73         selected_word = self._word_list_copy.pop(index)
74         self._word_list_copy.insert(0, selected_word)
75
76         if self._expanded:
77             self._word_list.pop(index)
78             self._word_list.insert(0, selected_word)
79         else:
80             self._word_list = [selected_word]
81
82     def up_func(self):
83         if self.get_widget_state() == WidgetState.PRESS:
84             if self._expanded and self._hovered_index is not None:
85                 self.set_selected_word(self._word_list_copy[self._hovered_index])
86
87             self.toggle_expanded()
88
89             self._hovered_index = None
90
91             self.set_state_colour(WidgetState.BASE)
92             self.set_geometry()
93
94

```

```

93     def calculate_hovered_index(self, mouse_pos):
94         return int(mouse_pos[1] // (self.size[1] / len(self._word_list)))
95
96     def set_image(self):
97         text_surface = pygame.transform.scale(self._empty_surface, self.size)
98         self.image = text_surface
99
100        fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
101        pygame.draw.rect(self.image, self._background_colour, fill_rect)
102        pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int(
103            self.border_width))
104
105        word_box_height = (self.size[1] - (2 * self.margin) - ((len(self.
106            _word_list) - 1) * self.margin)) / len(self._word_list)
107
108        arrow_size = (GRAPHICS['dropdown_arrow_open'].width / GRAPHICS['
109            dropdown_arrow_open'].height * word_box_height, word_box_height)
110        open_arrow_surface = pygame.transform.scale(GRAPHICS['dropdown_arrow_open'
111            ], arrow_size)
112        closed_arrow_surface = pygame.transform.scale(GRAPHICS['
113            dropdown_arrow_close'], arrow_size)
114        arrow_position = (self.size[0] - arrow_size[0] - self.margin, (
115            word_box_height) / 3)
116
116        if self._expanded:
117            self.image.blit(closed_arrow_surface, arrow_position)
118        else:
119            self.image.blit(open_arrow_surface, arrow_position)
120
121        for index, word in enumerate(self._word_list):
122            word_position = (self.margin, self.margin + (word_box_height + self.
123                margin) * index)
124            self._font.render_to(self.image, word_position, word, fgcolor=self.
125                _text_colour, size=self.font_size)
126
127        if self._hovered_index is not None:
128            overlay_surface = pygame.Surface((self.size[0], word_box_height + 2 *
129                self.margin), pygame.SRCALPHA)
130            overlay_surface.fill(*self._fill_colour.rgb, 128)
131            overlay_position = (0, (word_box_height + self.margin) * self.
132                _hovered_index)
133            self.image.blit(overlay_surface, overlay_position)

```

### B.25.13 icon.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.helpers.widget_helpers import create_text_box
4
5 class Icon(_Widget):
6     def __init__(self, icon, stretch=False, is_mask=False, smooth=False, fit_icon=
7         False, box_colours=None, **kwargs):
8         super().__init__(**kwargs)
9
10        if fit_icon:
11            aspect_ratio = icon.width / icon.height
12            self._relative_size = (self._relative_size[1] * aspect_ratio, self.
13                _relative_size[1])
14
15        self._icon = icon
16        self._is_mask = is_mask
17        self._stretch = stretch

```

```

16         self._smooth = smooth
17         self._box_colours = box_colours
18
19         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
20
21         self.set_image()
22         self.set_geometry()
23
24     def set_icon(self, icon):
25         self._icon = icon
26         self.set_image()
27
28     def set_image(self):
29         if self._box_colours:
30             self.image = create_text_box(self.size, self.border_width, self.
31 _box_colours)
32         else:
33             self.image = pygame.transform.scale(self._empty_surface, self.size)
34
35             if self._fill_colour:
36                 pygame.draw.rect(self.image, self._fill_colour, self.image.
37 get_rect(), border_radius=int(self.border_radius))
38
39             if self._stretch:
40                 if self._smooth:
41                     scaled_icon = pygame.transform.smoothscale(self._icon, (self.size
42 [0] - (2 * self.margin), self.size[1] - (2 * self.margin)))
43                 else:
44                     scaled_icon = pygame.transform.scale(self._icon, (self.size[0] -
45 (2 * self.margin), self.size[1] - (2 * self.margin)))
46
47                 icon_position = (self.margin, self.margin)
48             else:
49                 max_height = self.size[1] - (2 * self.margin)
50                 max_width = self.size[0] - (2 * self.margin)
51                 scale_factor = min(max_width / self._icon.width, max_height / self.
52 _icon.height)
53
54                 if self._smooth:
55                     scaled_icon = pygame.transform.smoothscale_by(self._icon, (
56 scale_factor, scale_factor))
57                 else:
58                     scaled_icon = pygame.transform.scale_by(self._icon, (scale_factor,
59 scale_factor))
60                 icon_position = ((self.size[0] - scaled_icon.width) / 2, (self.size[1]
61 - scaled_icon.height) / 2)
62
63             if self._is_mask:
64                 self.image.blit(scaled_icon, icon_position, None, pygame.
BLEND_RGBA_MULT)
65             else:
66                 self.image.blit(scaled_icon, icon_position)
67
68             if self._box_colours is None and self.border_width:
69                 pygame.draw.rect(self.image, self._border_colour, self.image.get_rect
70 (), width=int(self.border_width), border_radius=int(self.border_radius))
71
72     def process_event(self, event):
73         pass

```

### B.25.14 icon\_button.py

```

1 from data.widgets.bases.pressable import _Pressable
2 from data.widgets.bases.box import _Box
3 from data.widgets.icon import Icon
4 from data.utils.constants import WidgetState, RED_BUTTON_COLOURS
5
6 class IconButton(_Box, _Pressable, Icon):
7     def __init__(self, event, box_colours=RED_BUTTON_COLOURS, **kwargs):
8         _Box.__init__(self, box_colours=box_colours)
9         _Pressable.__init__(
10             self,
11             event=event,
12             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
13             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
14             up_func=lambda: self.set_state_colour(WidgetState.BASE),
15         )
16         Icon.__init__(self, box_colours=box_colours[WidgetState.BASE], **kwargs)
17
18     self.initialise_new_colours(self._fill_colour)
19     self.set_state_colour(WidgetState.BASE)

```

### B.25.15 move\_list.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.helpers.font_helpers import width_to_font_size
4
5 class MoveList(_Widget):
6     def __init__(self, relative_width, minimum_height=0, move_list=[], **kwargs):
7         super().__init__(relative_size=None, **kwargs)
8
9         self._relative_width = relative_width * self.surface_size[0] / self.
surface_size[1]
10        self._relative_minimum_height = minimum_height / self.surface_size[1]
11        self._move_list = move_list
12        self._relative_font_size = width_to_font_size(self._font, self.
surface_size[0] / 3.5) / self.surface_size[1]
13
14        self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
15
16        self.set_image()
17        self.set_geometry()
18
19    @property
20    def size(self):
21        font_metrics = self._font.get_metrics('j', size=self.font_size)
22
23        width = self._relative_width * self.surface_size[1]
24        minimum_height = self._relative_minimum_height * self.surface_size[1]
25        row_gap = font_metrics[0][3] - font_metrics[0][2]
26        number_of_rows = 2 * ((len(self._move_list) + 1) // 2) + 1
27
28        return (width, max(minimum_height, row_gap * number_of_rows))
29
30    def register_get_rect(self, get_rect_func):
31        pass
32
33    def reset_move_list(self):
34        self._move_list = []
35        self.set_image()
36        self.set_geometry()
37
38    def append_to_move_list(self, new_move):

```

```

39         self._move_list.append(new_move)
40         self.set_image()
41         self.set_geometry()
42
43     def pop_from_move_list(self):
44         self._move_list.pop()
45         self.set_image()
46         self.set_geometry()
47
48     def set_image(self):
49         self.image = pygame.transform.scale(self._empty_surface, self.size)
50         self.image.fill(self._fill_colour)
51
52         font_metrics = self._font.get_metrics('j', size=self.font_size)
53         row_gap = font_metrics[0][3] - font_metrics[0][2]
54
55         for index, move in enumerate(self._move_list):
56             if index % 2 == 0:
57                 text_position = (self.size[0] / 7, row_gap * (1 + 2 * (index // 2)))
58             else:
59                 text_position = (self.size[0] * 4 / 7, row_gap * (1 + 2 * (index // 2)))
60
61             self._font.render_to(self.image, text_position, text=move, size=self.font_size, fgcolor=self._text_colour)
62
63             move_number = (index // 2) + 1
64             move_number_position = (self.size[0] / 14, row_gap * (1 + 2 * (index // 2)))
65             self._font.render_to(self.image, move_number_position, text=str(move_number), size=self.font_size, fgcolor=self._text_colour)
66
67     def process_event(self, event, scrolled_pos=None):
68         pass

```

### B.25.16 multiple\_icon\_button.py

```

1 import pygame
2 from data.utils.constants import WidgetState, LOCKED_BLUE_BUTTON_COLOURS,
3     LOCKED_RED_BUTTON_COLOURS, RED_BUTTON_COLOURS, BLUE_BUTTON_COLOURS
4 from data.components.custom_event import CustomEvent
5 from data.widgets.bases.circular import _Circular
6 from data.widgets.icon_button import IconButton
7 from data.widgets.bases.box import _Box
8
9 class MultipleIconButton(_Circular, IconButton):
10     def __init__(self, icons_dict, **kwargs):
11         _Circular.__init__(self, items_dict=icons_dict)
12         IconButton.__init__(self, icon=self.current_item, **kwargs)
13
14         self._fill_colour_copy = self._fill_colour
15
16         self._locked = None
17
18     def set_locked(self, is_locked):
19         self._locked = is_locked
20
21         if self._locked:
22             r, g, b, a = pygame.Color(self._fill_colour_copy).rgba
23             if self._box_colours_dict == BLUE_BUTTON_COLOURS:
24                 _Box.__init__(self, box_colours=LOCKED_BLUE_BUTTON_COLOURS)
25             elif self._box_colours_dict == RED_BUTTON_COLOURS:

```

```

24         _Box.__init__(self, box_colours=LOCKED_RED_BUTTON_COLOURS)
25     else:
26         self.initialise_new_colours((max(r + 50, 0), max(g + 50, 0), max(b + 50,
27             0), a))
28     else:
29         if self._box_colours_dict == LOCKED_BLUE_BUTTON_COLOURS:
30             _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
31         elif self._box_colours_dict == LOCKED_RED_BUTTON_COLOURS:
32             _Box.__init__(self, box_colours=RED_BUTTON_COLOURS)
33         else:
34             self.initialise_new_colours(self._fill_colour_copy)
35
36     if self.rect.collidepoint(pygame.mouse.get_pos()):
37         self.set_state_colour(WidgetState.HOVER)
38     else:
39         self.set_state_colour(WidgetState.BASE)
40
41     def set_next_icon(self):
42         super().set_next_item()
43         self._icon = self.current_item
44         self.set_image()
45
46     def process_event(self, event):
47         widget_event = super().process_event(event)
48
49     if widget_event:
50         return CustomEvent(**vars(widget_event), data=self.current_key)

```

### B.25.17 piece\_display.py

```

1 import pygame
2 from data.utils.constants import WidgetState, BLUE_BUTTON_COLOURS,
3     RED_BUTTON_COLOURS
4 from data.states.game.components.piece_sprite import PieceSprite
5 from data.helpers.widget_helpers import create_text_box
6 from data.helpers.asset_helpers import scale_and_cache
7 from data.utils.enums import Score, Rotation, Colour
8 from data.widgets.bases.widget import _Widget
9
9 class PieceDisplay(_Widget):
10     def __init__(self, active_colour, **kwargs):
11         super().__init__(**kwargs)
12
13         self._active_colour = active_colour
14         self._piece_list = []
15         self._piece_surface = None
16         self._box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour
17         == Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]
18
19         self.initialise_piece_surface()
20
21         self.set_image()
22         self.set_geometry()
23
24     def add_piece(self, piece):
25         self._piece_list.append(piece)
26         self._piece_list.sort(key=lambda piece: Score[piece.name])
27         self.initialise_piece_surface()
28
29     def remove_piece(self, piece):
30         self._piece_list.remove(piece)
31         self.initialise_piece_surface()

```

```

31
32     def reset_piece_list(self):
33         self._piece_list = []
34         self.initialise_piece_surface()
35
36     def initialise_piece_surface(self):
37         self._piece_surface = pygame.Surface((self.size[0] - 2 * self.margin, self.size[1] - 2 * self.margin), pygame.SRCALPHA)
38
39         if (len(self._piece_list) == 0):
40             self.set_image()
41             return
42
43         piece_width = min(self.size[1] - 2 * self.margin, (self.size[0] - 2 * self.margin) / len(self._piece_list))
44         piece_list = []
45
46         for index, piece in enumerate(self._piece_list):
47             piece_instance = PieceSprite(piece, self._active_colour.
48               get_flipped_colour(), Rotation.UP)
49             piece_instance.set_geometry((0, 0), piece_width)
50             piece_instance.set_image()
51             piece_list.append((piece_instance.image, (piece_width * index, (self.
52               _piece_surface.height - piece_width) / 2)))
53
54         self._piece_surface.fblits(piece_list)
55
56     def set_image(self):
57         self.image = create_text_box(self.size, self.border_width, self.
58           _box_colours)
59
60         resized_piece_surface = scale_and_cache(self._piece_surface, (self.size[0]
61           - 2 * self.margin, self.size[1] - 2 * self.margin))
62         self.image.blit(resized_piece_surface, (self.margin, self.margin))
63
64     def process_event(self, event):
65         pass

```

**B.25.18 reactive\_button.py**

See Section 3.4.4.

**B.25.19 reactive\_icon\_button.py**

See Section 3.4.4.

**B.25.20 rectangle.py**

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3
4 class Rectangle(_Widget):
5     def __init__(self, visible=False, **kwargs):
6         super().__init__(**kwargs)
7
8         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
9         self._visible = visible
10

```

```

11         self.set_image()
12         self.set_geometry()
13
14     def set_image(self):
15         self.image = pygame.transform.scale(self._empty_surface, self.size)
16         if self._visible:
17             pygame.draw.rect(self.image, self._fill_colour, self.image.get_rect(),
18                             border_radius=int(self.border_radius))
19
20             if self.border_width:
21                 pygame.draw.rect(self.image, self._border_colour, self.image.
22                                 get_rect(), width=int(self.border_width), border_radius=int(self.border_radius))
23
24     def process_event(self, event):
25         pass

```

### B.25.21 scrollbar.py

```

1 import pygame
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.bases.widget import _Widget
4 from data.utils.constants import WidgetState
5 from data.utils.enums import Miscellaneous
6
7 class _Scrollbar(_Pressable, _Widget):
8     def __init__(self, vertical, **kwargs):
9         _Pressable.__init__(
10             self,
11             event=Miscellaneous.PLACEHOLDER,
12             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
13             down_func=self.down_func,
14             up_func=self.up_func,
15             prolonged=True,
16             sfx=None
17         )
18         _Widget.__init__(self, **kwargs)
19
20         self._vertical = vertical
21         self._last_mouse_px = None
22
23         self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
24
25         self.initialise_new_colours(self._fill_colour)
26         self.set_state_colour(WidgetState.BASE)
27
28         self.set_image()
29         self.set_geometry()
30
31     def down_func(self):
32         if self._vertical:
33             self._last_mouse_px = pygame.mouse.get_pos()[1]
34         else:
35             self._last_mouse_px = pygame.mouse.get_pos()[0]
36
37         self.set_state_colour(WidgetState.PRESS)
38
39     def up_func(self):
40         self._last_mouse_px = None
41         self.set_state_colour(WidgetState.BASE)
42
43     def set_relative_position(self, relative_position):

```

```

44         self._relative_position = relative_position
45         self.set_geometry()
46
47     def set_relative_size(self, new_relative_size):
48         self._relative_size = new_relative_size
49
50     def set_image(self):
51         self.image = pygame.transform.scale(self._empty_surface, self.size)
52
53         if self._vertical:
54             rounded_radius = self.size[0] / 2
55         else:
56             rounded_radius = self.size[1] / 2
57
58         pygame.draw.rect(self.image, self._fill_colour, (0, 0, self.size[0], self.size[1]), border_radius=int(rounded_radius))
59
60     def process_event(self, event):
61         before_state = self.get_widget_state()
62         widget_event = super().process_event(event)
63         after_state = self.get_widget_state()
64
65         if event.type == pygame.MOUSEMOTION and self._last_mouse_px:
66             if self._vertical:
67                 offset_from_last_frame = event.pos[1] - self._last_mouse_px
68                 self._last_mouse_px = event.pos[1]
69
70                 return offset_from_last_frame
71             else:
72                 offset_from_last_frame = event.pos[0] - self._last_mouse_px
73                 self._last_mouse_px = event.pos[0]
74
75             return offset_from_last_frame
76
77
78         if widget_event or before_state != after_state:
79             return 0

```

### B.25.22 scroll\_area.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.scrollbar import _Scrollbar
4 from data.managers.theme import theme
5
6 SCROLLBAR_WIDTH_FACTOR = 0.05
7
8 class ScrollArea(_Widget):
9     def __init__(self, widget, vertical, scroll_factor=15, **kwargs):
10         super().__init__(**kwargs)
11         if vertical is False:
12             self._relative_size = kwargs.get('relative_size')
13
14             self._relative_scroll_factor = scroll_factor / self.surface_size[1]
15
16             self._scroll_percentage = 0
17             self._widget = widget
18             self._vertical = vertical
19
20             self._widget.register_get_rect(self.calculate_widget_rect)
21
22         if self._vertical:

```

```

23         anchor_x = 'right'
24         anchor_y = 'top'
25         scale_mode = 'height'
26     else:
27         anchor_x = 'left'
28         anchor_y = 'bottom'
29         scale_mode = 'width'
30
31     self._scrollbar = _Scrollbar(
32         parent=self,
33         relative_position=(0, 0),
34         relative_size=None,
35         anchor_x=anchor_x,
36         anchor_y=anchor_y,
37         fill_colour=theme['borderPrimary'],
38         scale_mode=scale_mode,
39         vertical=vertical,
40     )
41
42     self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
43
44     self.set_image()
45     self.set_geometry()
46
47     @property
48     def scroll_factor(self):
49         return self._relative_scroll_factor * self.surface_size[1]
50
51     @property
52     def scrollbar_size(self):
53         if self._vertical:
54             return (self.size[0] * SCROLLBAR_WIDTH_FACTOR, min(1, self.size[1] /
55             self._widget.rect.height) * self.size[1])
56         else:
57             return (min(1, self.size[0] / (self._widget.rect.width + 0.001)) *
58             self.size[0], self.size[1] * SCROLLBAR_WIDTH_FACTOR)
59
60     @property
61     def size(self):
62         if self._vertical is False:
63             return (self._relative_size[0] * self.surface_size[0], self.
64             _relative_size[1] * self.surface_size[1]) # scale with horizontal width to
65             always fill entire length of screen
66         else:
67             return super().size
68
69     def calculate_scroll_percentage(self, offset, scrollbar=False):
70         if self._vertical:
71             widget_height = self._widget.rect.height
72
73             if widget_height < self.size[1]:
74                 return 0
75
76             if scrollbar:
77                 self._scroll_percentage += offset / (self.size[1] - self.
78                 scrollbar_size[1] + 0.001)
79             else:
80                 max_scroll_height = widget_height - self.size[1]
81                 current_scroll_height = self._scroll_percentage *
82                 max_scroll_height
83                 self._scroll_percentage = (current_scroll_height + offset) / (
84                 max_scroll_height + 0.001)

```

```

78         else:
79             widget_width = self._widget.rect.width
80
81             if widget_width < self.size[0]:
82                 return 0
83
84             if scrollbar:
85                 self._scroll_percentage += offset / (self.size[0] - self.
86 scrollbar_size[0] + 0.001)
87             else:
88                 max_scoll_width = widget_width - self.size[0]
89                 current_scroll_width = self._scroll_percentage * max_scoll_width
90                 self._scroll_percentage = (current_scroll_width + offset) /
91 max_scoll_width
92
93             return min(1, max(0, self._scroll_percentage))
94
95     def calculate_widget_rect(self):
96         widget_position = self.calculate_widget_position()
97         return pygame.FRect(widget_position[0] - self.position[0], self.position
98 [1] + widget_position[1], self.size[0], self.size[1])
99
100    def calculate_widget_position(self):
101        if self._vertical:
102            return (0, -self._scroll_percentage * (self._widget.rect.height - self.
103 size[1]))
104        else:
105            return (-self._scroll_percentage * (self._widget.rect.width - self.
106 size[0]), 0)
107
108    def calculate_relative_scrollbar_position(self):
109        if self._vertical:
110            vertical_offset = (self.size[1] - self.scrollbar_size[1]) * self.
111 _scroll_percentage
112            scrollbar_position = (0, vertical_offset)
113        else:
114            horizontal_offset = (self.size[0] - self.scrollbar_size[0]) * self.
115 _scroll_percentage
116            scrollbar_position = (horizontal_offset, 0)
117
118        return (scrollbar_position[0] / self.size[0], scrollbar_position[1] / self.
119 size[1]))
120
121    def set_widget(self, new_widget):
122        self._widget = new_widget
123        self.set_image()
124        self.set_geometry()
125
126    def set_image(self):
127        self.image = pygame.transform.scale(self._empty_surface, self.size)
128        self.image.fill(theme['fillPrimary'])
129
130        self._widget.set_image()
131        self.image.blit(self._widget.image, self.calculate_widget_position())
132
133        self._scrollbar.set_relative_position(self.
134 calculate_relative_scrollbar_position()) # WRONG USING RELATIVE
135        self._scrollbar.set_relative_size((self.scrollbar_size[0] / self.size[1],
136 self.scrollbar_size[1] / self.size[1]))
137        self._scrollbar.set_image()
138        relative_scrollbar_position = (self._scrollbar.rect.left - self.position
139 [0], self._scrollbar.rect.top - self.position[1])

```

```

129         self.image.blit(self._scrollbar.image, relative_scrollbar_position)
130
131     def set_geometry(self):
132         super().set_geometry()
133         self._widget.set_geometry()
134         self._scrollbar.set_geometry()
135
136     def set_surface_size(self, new_surface_size):
137         super().set_surface_size(new_surface_size)
138         self._widget.set_surface_size(new_surface_size)
139         self._scrollbar.set_surface_size(new_surface_size)
140
141     def process_event(self, event):
142         # WAITING FOR PYGAME-CE 2.5.3 TO RELEASE TO FIX SCROLL FLAGS
143         # self.image.scroll(0, SCROLL_FACTOR)
144         # self.image.scroll(0, -SCROLL_FACTOR)
145
146         offset = self._scrollbar.process_event(event)
147
148         if offset is not None:
149             self.set_image()
150
151             if abs(offset) > 0:
152                 self._scroll_percentage = self.calculate_scroll_percentage(offset,
153 scrollbar=True)
154
155             if self.rect.collidepoint(pygame.mouse.get_pos()):
156                 if event.type == pygame.MOUSEBUTTONDOWN:
157                     if event.button == 4:
158                         self._scroll_percentage = self.calculate_scroll_percentage(-
self.scroll_factor)
159                         self.set_image()
160                         return
161                     elif event.button == 5:
162                         if self._scroll_percentage == 100:
163                             return
164
165                         self._scroll_percentage = self.calculate_scroll_percentage(
self.scroll_factor)
166                         self.set_image()
167                         return
168
169             widget_event = self._widget.process_event(event, scrolled_pos=self.
calculate_widget_position())
170             if widget_event is not None:
171                 self.set_image()
172             return widget_event

```

### B.25.23 slider\_thumb.py

```

1 from data.widgets.bases.pressable import _Pressable
2 from data.utils.constants import WidgetState
3 from data.helpers.widget_helpers import create_slider_thumb
4 from data.managers.theme import theme
5
6 class _SliderThumb(_Pressable):
7     def __init__(self, radius, border_colour=theme['borderPrimary'], fill_colour=
theme['fillPrimary']):
8         super().__init__(
9             event=None,
10            down_func=self.down_func,
11            up_func=self.up_func,

```

```

12         hover_func=self.hover_func,
13         prolonged=True,
14         sfx=None
15     )
16     self._border_colour = border_colour
17     self._radius = radius
18     self._percent = None
19
20     self.state = WidgetState.BASE
21     self.initialise_new_colours(fill_colour)
22
23     def get_position(self):
24         return (self.rect.x, self.rect.y)
25
26     def set_position(self, position):
27         self.rect = self._thumb_surface.get_rect()
28         self.rect.topleft = position
29
30     def get_surface(self):
31         return self._thumb_surface
32
33     def set_surface(self, radius, border_width):
34         self._thumb_surface = create_slider_thumb(radius, self._colours[self.state],
35             self._border_colour, border_width)
35
36     def get_pressed(self):
37         return self._pressed
38
39     def down_func(self):
40         self.state = WidgetState.PRESS
41
42     def up_func(self):
43         self.state = WidgetState.BASE
44
45     def hover_func(self):
46         self.state = WidgetState.HOVER

```

### B.25.24 switch.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.utils.constants import WidgetState
5 from data.helpers.widget_helpers import create_switch
6 from data.components.custom_event import CustomEvent
7 from data.managers.theme import theme
8
9 class Switch(_Pressable, _Widget):
10     def __init__(self, relative_height, event, fill_colour=theme['fillTertiary'],
11                  on_colour=theme['fillSecondary'], off_colour=theme['fillPrimary'], **kwargs):
12         _Pressable.__init__(
13             self,
14             event=event,
15             hover_func=self.hover_func,
16             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
17             up_func=self.up_func,
18         )
19         _Widget.__init__(self, relative_size=(relative_height * 2, relative_height),
20                         scale_mode='height', fill_colour=fill_colour, **kwargs)
21
22         self._on_colour = on_colour
23         self._off_colour = off_colour

```

```

22         self._background_colour = None
23
24         self._is_toggled = None
25         self.set_toggle_state(False)
26
27         self.initialise_new_colours(self._fill_colour)
28         self.set_state_colour(WidgetState.BASE)
29
30         self.set_image()
31         self.set_geometry()
32
33     def hover_func(self):
34         self.set_state_colour(WidgetState.HOVER)
35
36     def set_toggle_state(self, is_toggled):
37         self._is_toggled = is_toggled
38         if is_toggled:
39             self._background_colour = self._on_colour
40         else:
41             self._background_colour = self._off_colour
42
43         self.set_image()
44
45     def up_func(self):
46         if self.get_widget_state() == WidgetState.PRESS:
47             toggle_state = not(self._is_toggled)
48             self.set_toggle_state(toggle_state)
49
50         self.set_state_colour(WidgetState.BASE)
51
52     def draw_thumb(self):
53         margin = self.size[1] * 0.1
54         thumb_radius = (self.size[1] / 2) - margin
55
56         if self._is_toggled:
57             thumb_center = (self.size[0] - margin - thumb_radius, self.size[1] /
58                             2)
58         else:
59             thumb_center = (margin + thumb_radius, self.size[1] / 2)
60
61         pygame.draw.circle(self.image, self._fill_colour, thumb_center,
62                            thumb_radius)
62
63     def set_image(self):
64         self.image = create_switch(self.size, self._background_colour)
65         self.draw_thumb()
66
67     def process_event(self, event):
68         data = super().process_event(event)
69
70         if data:
71             return CustomEvent(**vars(data), toggled=self._is_toggled)

```

### B.25.25 text.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.helpers.font_helpers import text_width_to_font_size,
4                                         text_height_to_font_size, height_to_font_size
5 from data.helpers.widget_helpers import create_text_box
6
6 class Text(_Widget): # Pure text

```

```

7     def __init__(self, text, center=True, fit_vertical=True, box_colours=None,
8      strength=0.05, font_size=None, **kwargs):
9         super().__init__(**kwargs)
10        self._text = text
11        self._fit_vertical = fit_vertical
12        self._strength = strength
13        self._box_colours = box_colours
14
15        if fit_vertical:
16            self._relative_font_size = text_height_to_font_size(self._text, self.
17            _font, (self.size[1] - 2 * (self.margin + self.border_width)) / self.
18            surface_size[1])
19        else:
20            self._relative_font_size = text_width_to_font_size(self._text, self.
21            _font, (self.size[0] - 2 * (self.margin + self.border_width)) / self.
22            surface_size[1])
23
24        if font_size:
25            self._relative_font_size = font_size / self.surface_size[1]
26
27        self._center = center
28        self.rect = self._font.get_rect(self._text, size=self.font_size)
29        self.rect.topleft = self.position
30
31        self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
32
33        self.set_image()
34        self.set_geometry()
35
36    def resize_text(self):
37        if self._fit_vertical:
38            self._relative_font_size = text_height_to_font_size(self._text, self.
39            _font, (self.size[1] - 2 * (self.margin + self.border_width)) / self.
40            surface_size[1])
41        else:
42            ideal_font_size = height_to_font_size(self._font, target_height=(self.
43            size[1] - (self.margin + self.border_width)) / self.surface_size[1]
44            new_font_size = text_width_to_font_size(self._text, self._font, (self.
45            size[0] - (self.margin + self.border_width)) / self.surface_size[1])
46
47            if new_font_size < ideal_font_size:
48                self._relative_font_size = new_font_size
49            else:
50                self._relative_font_size = ideal_font_size
51
52    def set_text(self, new_text):
53        self._text = new_text
54
55        self.resize_text()
56        self.set_image()
57
58    def set_image(self):
59        if self._box_colours:
60            self.image = create_text_box(self.size, self.border_width, self.
61            _box_colours)
62        else:
63            text_surface = pygame.transform.scale(self._empty_surface, self.size)
64            self.image = text_surface
65
66        if self._fill_colour:
67            fill_rect = pygame.Rect(0, 0, self.size[0], self.size[1])
68            pygame.draw.rect(self.image, self._fill_colour, fill_rect,

```

```

    border_radius=int(self.border_radius))

59
60     self._font.strength = self._strength
61     font_rect_size = self._font.get_rect(self._text, size=self.font_size).size
62     if self._center:
63         font_position = ((self.size[0] - font_rect_size[0]) / 2, (self.size[1]
64 - font_rect_size[1]) / 2)
65     else:
66         font_position = (self.margin / 2, (self.size[1] - font_rect_size[1]) /
67 2)
68     self._font.render_to(self.image, font_position, self._text, fgcolor=self.
69 _text_colour, size=self.font_size)
70
71     if self._box_colours is None and self.border_width:
72         fill_rect = pygame.Rect(0, 0, self.size[0], self.size[1])
73         pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int
74 (self.border_width), border_radius=int(self.border_radius))
75
76 def process_event(self, event):
77     pass

```

**B.25.26 text\_button.py**

```

1 from data.widgets.bases.pressable import _Pressable
2 from data.widgets.bases.box import _Box
3 from data.widgets.text import Text
4 from data.utils.constants import WidgetState, BLUE_BUTTON_COLOURS
5
6 class TextButton(_Box, _Pressable, Text):
7     def __init__(self, event, **kwargs):
8         _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
9         _Pressable.__init__(
10             self,
11             event=event,
12             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
13             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
14             up_func=lambda: self.set_state_colour(WidgetState.BASE),
15         )
16         Text.__init__(self, box_colours=BLUE_BUTTON_COLOURS[WidgetState.BASE], **
17         kwargs)
18
19         self.initialise_new_colours(self._fill_colour)
20         self.set_state_colour(WidgetState.BASE)

```

**B.25.27 text\_input.py**

See Section 3.4.4.

**B.25.28 timer.py**

```

1 import pygame
2 from data.utils.constants import WidgetState, BLUE_BUTTON_COLOURS,
3     RED_BUTTON_COLOURS
4 from data.components.custom_event import CustomEvent
5 from data.managers.animation import animation
6 from data.utils.enums import Colour
7 from data.widgets.text import Text
8
9 class Timer(Text):
10     def __init__(self, active_colour, event=None, start_mins=60, **kwargs):

```

```

10     box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour ==
11     Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]
12
13     self._current_ms = float(start_mins) * 60 * 1000
14     self._active_colour = active_colour
15     self._active = False
16     self._timer_running = False
17     self._event = event
18
19     super().__init__(text=self.format_to_text(), fit_vertical=False,
20     box_colours=box_colours, **kwargs)
21
22     def set_active(self, is_active):
23         if self._active == is_active:
24             return
25
26         if is_active and self._timer_running is False:
27             self._timer_running = True
28             animation.set_timer(1000, self.decrement_second)
29
30         self._active = is_active
31
32     def set_time(self, milliseconds):
33         self._current_ms = milliseconds
34         self._text = self.format_to_text()
35         self.set_image()
36         self.set_geometry()
37
38     def get_time(self):
39         return self._current_ms / (1000 * 60)
40
41     def decrement_second(self):
42         if self._active:
43             self.set_time(self._current_ms - 1000)
44
45             if self._current_ms <= 0:
46                 self._active = False
47                 self._timer_running = False
48                 self.set_time(0)
49                 pygame.event.post(pygame.event.Event(pygame.MOUSEMOTION, pos=
50                 pygame.mouse.get_pos())) # RANDOM EVENT TO TRIGGER process_event
51             else:
52                 animation.set_timer(1000, self.decrement_second)
53         else:
54             self._timer_running = False
55
56     def format_to_text(self):
57         raw_seconds = self._current_ms / 1000
58         minutes, seconds = divmod(raw_seconds, 60)
59         return f'{str(int(minutes)).zfill(2)}:{str(int(seconds)).zfill(2)}'
60
61     def process_event(self, event):
62         if self._current_ms <= 0:
63             return CustomEvent(**vars(self._event), active_colour=self.
64             _active_colour)

```

### B.25.29 volume\_slider.py

```

1 import pygame
2 from data.helpers.asset_helpers import scale_and_cache
3 from data.helpers.widget_helpers import create_slider
4 from data.utils.event_types import SettingsEventType

```

```

5  from data.components.custom_event import CustomEvent
6  from data.widgets.slider_thumb import _SliderThumb
7  from data.widgets.bases.widget import _Widget
8  from data.utils.constants import WidgetState
9  from data.managers.theme import theme
10
11 class VolumeSlider(_Widget):
12     def __init__(self, relative_length, default_volume, volume_type, thumb_colour=
13                  theme['fillSecondary'], **kwargs):
14         super().__init__(relative_size=(relative_length, relative_length * 0.2),
15                          **kwargs)
16
17         self._volume_type = volume_type
18         self._selected_percent = default_volume
19         self._last_mouse_x = None
20
21         self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
22                                     _border_colour, fill_colour=thumb_colour)
23         self._gradient_surface = create_slider(self.calculate_slider_size(), self.
24                                              _fill_colour, self.border_width, self._border_colour)
25
26         self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
27
28     @property
29     def position(self):
30         '''Minus so easier to position slider by starting from the left edge of
31         the slider instead of the thumb'''
32         return (self._relative_position[0] * self.surface_size[0] - (self.size[1]
33 / 2), self._relative_position[1] * self.surface_size[1])
34
35     def calculate_slider_position(self):
36         return (self.size[1] / 2, self.size[1] / 4)
37
38     def calculate_slider_size(self):
39         return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
40
41     def calculate_selected_percent(self, mouse_pos):
42         if self._last_mouse_x is None:
43             return
44
45         x_change = (mouse_pos[0] - self._last_mouse_x) / (self.
46 calculate_slider_size()[0] - 2 * self.border_width)
47         return max(0, min(self._selected_percent + x_change, 1))
48
49     def calculate_thumb_position(self):
50         gradient_size = self.calculate_slider_size()
51         x = gradient_size[0] * self._selected_percent
52         y = 0
53
54         return (x, y)
55
56     def relative_to_global_position(self, position):
57         relative_x, relative_y = position
58         return (relative_x + self.position[0], relative_y + self.position[1])
59
60     def set_image(self):
61         gradient_scaled = scale_and_cache(self._gradient_surface, self.
62 calculate_slider_size())
63         gradient_position = self.calculate_slider_position()
64
65         self.image = pygame.transform.scale(self._empty_surface, (self.size))
66         self.image.blit(gradient_scaled, gradient_position)

```

```

59
60         thumb_position = self.calculate_thumb_position()
61         self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
62 border_width)
62         self._thumb.set_position(self.relative_to_global_position((thumb_position
63 [0], thumb_position[1])))
63
64         thumb_surface = self._thumb.get_surface()
65         self.image.blit(thumb_surface, thumb_position)
66
67     def set_volume(self, volume):
68         self._selected_percent = volume
69         self.set_image()
70
71     def process_event(self, event):
72         if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
72 MOUSEBUTTONUP]:
73             return
74
75         before_state = self._thumb.state
76         self._thumb.process_event(event)
77         after_state = self._thumb.state
78
79         if before_state != after_state:
80             self.set_image()
81
82         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:
83             self._last_mouse_x = None
84             return CustomEvent(SettingsEventType.VOLUME_SLIDER_CLICK, volume=
round(self._selected_percent, 3), volume_type=self._volume_type)
85
86         if self._thumb.state == WidgetState.PRESS:
87             selected_percent = self.calculate_selected_percent(event.pos)
88             self._last_mouse_x = event.pos[0]
89
90         if selected_percent:
91             self._selected_percent = selected_percent
92             self.set_image()
93             return CustomEvent(SettingsEventType.VOLUME_SLIDER_SLIDE)

```

### B.25.30 \_\_init\_\_.py

```

1 from data.widgets.bases.widget import _Widget
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.bases.circular import _Circular
4 from data.widgets.bases.box import _Box
5 from data.widgets.colour_display import _ColourDisplay
6 from data.widgets.colour_square import _ColourSquare
7 from data.widgets.colour_slider import _ColourSlider
8 from data.widgets.slider_thumb import _SliderThumb
9 from data.widgets.scrollbar import _Scrollbar
10
11 from data.widgets.board_thumbnail_button import BoardThumbnailButton
12 from data.widgets.multiple_icon_button import MultipleIconButton
13 from data.widgets.reactive_icon_button import ReactiveIconButton
14 from data.widgets.board_thumbnail import BoardThumbnail
15 from data.widgets.reactive_button import ReactiveButton
16 from data.widgets.volume_slider import VolumeSlider
17 from data.widgets.colour_picker import ColourPicker
18 from data.widgets.colour_button import ColourButton
19 from data.widgets.browser_strip import BrowserStrip
20 from data.widgets.piece_display import PieceDisplay

```

```

21 from data.widgets.browser_item import BrowserItem
22 from data.widgets.text_button import TextButton
23 from data.widgets.icon_button import IconButton
24 from data.widgets.scroll_area import ScrollArea
25 from data.widgets.chessboard import Chessboard
26 from data.widgets.text_input import TextInput
27 from data.widgets.rectangle import Rectangle
28 from data.widgets.move_list import MoveList
29 from data.widgets.dropdown import Dropdown
30 from data.widgets.carousel import Carousel
31 from data.widgets.switch import Switch
32 from data.widgets.timer import Timer
33 from data.widgets.text import Text
34 from data.widgets.icon import Icon
35
36 __all__ = ['Text', 'TextButton', 'ColourPicker', 'ColourButton', 'Switch', '',
    'Dropdown', 'IconButton', 'Icon', 'VolumeSlider', 'TextInput', '',
    'MultipleIconButton', 'Carousel', 'Timer', 'Rectangle', 'Chessboard', '',
    'ScrollArea', 'MoveList', 'BoardThumbnail', 'BrowserStrip', 'BrowserItem', '',
    'PieceDisplay', 'BoardThumbnailButton', 'ReactiveButton', 'ReactiveIconButton']

```

## B.26 data\widgets\bases

### B.26.1 box.py

```

1 from data.utils.constants import WidgetState
2
3 class _Box:
4     def __init__(self, box_colours):
5         self._box_colours_dict = box_colours
6         self._box_colours = self._box_colours_dict[WidgetState.BASE]
7
8     def set_state_colour(self, state):
9         self._box_colours = self._box_colours_dict[state]
10        super().set_state_colour(state)

```

### B.26.2 circular.py

See Section 3.4.3.

### B.26.3 pressable.py

```

1 import pygame
2 from data.utils.constants import WidgetState
3 from data.managers.audio import audio
4 from data.utils.assets import SFX
5
6 class _Pressable:
7     def __init__(self, event, down_func=None, up_func=None, hover_func=None,
8                  prolonged=False, sfx=SFX['button_click'], **kwargs):
9         self._down_func = down_func
10        self._up_func = up_func
11        self._hover_func = hover_func
12        self._pressed = False
13        self._prolonged = prolonged
14        self._sfx = sfx
15
16        self._event = event

```



```
76             if self._widget_state in [WidgetState.PRESS, WidgetState.
77                                         HOVER]:
78                 self._widget_state = WidgetState.BASE
79                 self._up_func()
80             elif self._widget_state == WidgetState.BASE:
81                 return
82             elif self._prolonged is True:
83                 if self._widget_state in [WidgetState.PRESS, WidgetState.
84                                         BASE]:
85                     return
86             else:
87                 self._widget_state = WidgetState.BASE
88                 self._up_func()
```

#### B.26.4 widget.py

See Section 3.4.3.