


```

74     BASE = auto()
75     HOVER = auto()
76     PRESS = auto()
77
78     BLUE_BUTTON_COLOURS = {
79         WidgetState.BASE: ['0x1c2638', '0x23495d', '0x39707a', '0x95e0cc'],
80         WidgetState.HOVER: ['0xdaf2e9', '0x23495d', '0x39707a', '0x95e0cc'],
81         WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
82     }
83
84     INPUT_COLOURS = {
85         WidgetState.BASE: ['0x1c2638', '0x39707a', '0x23495d', '0x95e0cc'],
86         WidgetState.HOVER: ['0xdaf2e9', '0x39707a', '0x23495d', '0x95e0cc'],
87         WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0x1c2638', '0x39707a']
88     }
89
90     RED_BUTTON_COLOURS = {
91         WidgetState.BASE: ['0x000000', '0x1c2638', '0x9b222b', '0xf14e52'],
92         WidgetState.HOVER: ['0xdaf2e9', '0x1c2638', '0x9b222b', '0xf14e52'],
93         WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0xf14e52', '0x95e0cc']
94     }
95
96     LOCKED_RED_BUTTON_COLOURS = {
97         WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
98         WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
99         WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0xf14e52']
100    }
101
102    LOCKED_BLUE_BUTTON_COLOURS = {
103        WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
104        WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
105        WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
106    }
107
108    class StatusText(StrEnum):
109        PLAYER_MOVE = auto()
110        CPU_MOVE = auto()
111        WIN = auto()
112        DRAW = auto()
113
114    class EditorEventType(StrEnum):
115        MENU_CLICK = auto()
116        PICK_PIECE_CLICK = auto()
117        ROTATE_PIECE_CLICK = auto()
118        COPY_CLICK = auto()
119        EMPTY_CLICK = auto()
120        RESET_CLICK = auto()
121        BLUE_START_CLICK = auto()
122        RED_START_CLICK = auto()
123        START_CLICK = auto()
124        CONFIG_CLICK = auto()
125        ERASE_CLICK = auto()
126        MOVE_CLICK = auto()
127        HELP_CLICK = auto()
128
129    class ReviewEventType(StrEnum):
130        MENU_CLICK = auto()
131        PREVIOUS_CLICK = auto()
132        NEXT_CLICK = auto()
133        HELP_CLICK = auto()
134
135    class BrowserEventType(StrEnum):

```

```

136     MENU_CLICK = auto()
137     BROWSER_STRIP_CLICK = auto()
138     COPY_CLICK = auto()
139     DELETE_CLICK = auto()
140     REVIEW_CLICK = auto()
141     FILTER_COLUMN_CLICK = auto()
142     FILTER_ASCEND_CLICK = auto()
143     PAGE_CLICK = auto()
144     HELP_CLICK = auto()
145
146     class GameEventType(StrEnum):
147         BOARD_CLICK = auto()
148         PIECE_CLICK = auto()
149         PAUSE_CLICK = auto()
150         MENU_CLICK = auto()
151         GAME_CLICK = auto()
152         HELP_CLICK = auto()
153         TUTORIAL_CLICK = auto()
154         RESIGN_CLICK = auto()
155         DRAW_CLICK = auto()
156         REVIEW_CLICK = auto()
157         PIECE_DROP = auto()
158         UPDATE_PIECES = auto()
159         ROTATE_PIECE = auto()
160         SET_LASER = auto()
161         TIMER_END = auto()
162
163     class MenuEventType(StrEnum):
164         CONFIG_CLICK = auto()
165         SETTINGS_CLICK = auto()
166         BROWSER_CLICK = auto()
167         QUIT_CLICK = auto()
168         CREDITS_CLICK = auto()
169
170     class SettingsEventType(StrEnum):
171         RESET_DEFAULT = auto()
172         RESET_USER = auto()
173         MENU_CLICK = auto()
174         COLOUR_SLIDER_SLIDE = auto()
175         COLOUR_SLIDER_CLICK = auto()
176         COLOUR_PICKER_HOVER = auto()
177         PRIMARY_COLOUR_PICKER_CLICK = auto()
178         SECONDARY_COLOUR_PICKER_CLICK = auto()
179         PRIMARY_COLOUR_BUTTON_CLICK = auto()
180         SECONDARY_COLOUR_BUTTON_CLICK = auto()
181         VOLUME_SLIDER_SLIDE = auto()
182         VOLUME_SLIDER_CLICK = auto()
183         SHADER_PICKER_CLICK = auto()
184         OPENGL_CLICK = auto()
185         DROPDOWN_CLICK = auto()
186         PARTICLES_CLICK = auto()
187
188     class ConfigEventType(StrEnum):
189         GAME_CLICK = auto()
190         MENU_CLICK = auto()
191         FEN_STRING_TYPE = auto()
192         TIME_TYPE = auto()
193         TIME_CLICK = auto()
194         PVP_CLICK = auto()
195         PVC_CLICK = auto()
196         CPU_DEPTH_CLICK = auto()
197         PRESET_CLICK = auto()

```

```

198     SETUP_CLICK = auto()
199     COLOUR_CLICK = auto()
200     HELP_CLICK = auto()
201
202     class Colour(IntEnum):
203         BLUE = 0
204         RED = 1
205
206         def get_flipped_colour(self):
207             if self == Colour.BLUE:
208                 return Colour.RED
209             elif self == Colour.RED:
210                 return Colour.BLUE
211
212     class Piece(StrEnum):
213         SPHINX = 's'
214         PYRAMID = 'p'
215         ANUBIS = 'n'
216         SCARAB = 'r'
217         PHAROAH = 'f'
218
219     class Score(IntEnum):
220         PHAROAH = 0
221         SPHINX = 0
222         PYRAMID = 100
223         ANUBIS = 110
224         SCARAB = 200
225
226         MOVE = 4
227         POSITION = 11
228         PHAROAH_SAFETY = 31
229         CHECKMATE = 100000
230         INFINITE = 6969696969
231
232     class Rank(IntEnum):
233         ONE = 0
234         TWO = 1
235         THREE = 2
236         FOUR = 3
237         FIVE = 4
238         SIX = 5
239         SEVEN = 6
240         EIGHT = 7
241
242     class File(IntEnum):
243         A = 0
244         B = 1
245         C = 2
246         D = 3
247         E = 4
248         F = 5
249         G = 6
250         H = 7
251         I = 8
252         J = 9
253
254     class Rotation(StrEnum):
255         UP = 'a'
256         RIGHT = 'b'
257         DOWN = 'c'
258         LEFT = 'd'
259

```

```

260     def to_angle(self):
261         if self == Rotation.UP:
262             return 0
263         elif self == Rotation.RIGHT:
264             return 270
265         elif self == Rotation.DOWN:
266             return 180
267         elif self == Rotation.LEFT:
268             return 90
269
270     def get_clockwise(self):
271         if self == Rotation.UP:
272             return Rotation.RIGHT
273         elif self == Rotation.RIGHT:
274             return Rotation.DOWN
275         elif self == Rotation.DOWN:
276             return Rotation.LEFT
277         elif self == Rotation.LEFT:
278             return Rotation.UP
279
280     def get_anticlockwise(self):
281         if self == Rotation.UP:
282             return Rotation.LEFT
283         elif self == Rotation.RIGHT:
284             return Rotation.UP
285         elif self == Rotation.DOWN:
286             return Rotation.RIGHT
287         elif self == Rotation.LEFT:
288             return Rotation.DOWN
289
290     def get_opposite(self):
291         return self.get_clockwise().get_clockwise()
292
293     class RotationIndex(IntEnum):
294         FIRSTBIT = 0
295         SECONDBIT = 1
296
297     class RotationDirection(StrEnum):
298         CLOCKWISE = 'cw'
299         ANTICLOCKWISE = 'acw'
300
301     def get_opposite(self):
302         if self == RotationDirection.CLOCKWISE:
303             return RotationDirection.ANTICLOCKWISE
304         elif self == RotationDirection.ANTICLOCKWISE:
305             return RotationDirection.CLOCKWISE
306
307     class MoveType(StrEnum):
308         MOVE = 'm'
309         ROTATE = 'r'
310
311     class LaserType(IntEnum):
312         END = 0
313         STRAIGHT = 1
314         CORNER = 2
315
316     class LaserDirection(IntEnum):
317         FROM_TOP = 1
318         FROM_RIGHT = 2
319         FROM_BOTTOM = 3
320         FROM_LEFT = 4

```

1.1.3 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.managers.window import window
7 from data.managers.audio import audio
8 from data.managers.theme import theme
9 from data.assets import DEFAULT_FONT
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
PROCESS LEFT CLICKS
95                 return
96
97             self.state.get_event(event)
98
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()}.py')
119
120     def cleanup(self):

```



```

121         logger.info(f'cleaning {self.__class__.__name__.lower()}.py')
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()

```

1.1.4 loading_screen.py

See Section ??.

1.1.5 main.py

See Section ??.

1.1.6 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)

```

1.1.7 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

```

```

20 user32.SetProcessDPIAware() # To deal with Windows High Text Size / Low Display
    Resolution Settings
21
22 if os.name != 'nt' or sys.getwindowsversion()[0] < 6:
23     raise NotImplementedError("Incompatible OS!")

```

1.1.8 __init__.py

1.2 data\app_data

1.2.1 default_settings.json

```

1 {
2     "primaryBoardColour": "0xB98766",
3     "secondaryBoardColour": "0xF3D8B8",
4     "laserColourBlue": "0x0000ff",
5     "laserColourRed": "0xff0000",
6     "displayMode": "windowed",
7     "musicVolume": 0.5,
8     "sfxVolume": 0.5,
9     "particles": true,
10    "opengl": true,
11    "shader": "default"
12 }

```

1.2.2 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
11    "handlers": {
12        "console": {
13            "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 }

```

1.2.3 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 }

```

1.2.4 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 }

```

1.2.5 user_settings.json

```

1 {
2     "primaryBoardColour": "0xB98766",
3     "secondaryBoardColour": "0xF3D8B8",
4     "laserColourBlue": "0x0000ff",
5     "laserColourRed": "0xff0000",
6     "displayMode": "windowed",
7     "musicVolume": 0.085,
8     "sfxVolume": 0.336,
9     "particles": true,
10    "opengl": true,
11    "shader": "default"
12 }

```

1.3 data\components

1.3.1 circular_linked_list.py

See Section ??.

1.3.2 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

```

1.3.3 custom_event.py

See Section ??.

1.3.4 game_entry.py

```
1 from data.constants import Colour
2 from data.states.game.components.move import Move
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 [
42             {
43                 'blue_time': move.split(';')[0],
44                 'red_time': move.split(';')[1],
45                 'move': Move.instance_from_notation(move.split(';')[2]),
46                 'unparsed_move': move.split(';')[2],
47             } for move in moves if move != ''
48         ]
49
50 # self.states = {
51 #     'CPU_ENABLED': game_config['CPU_ENABLED'],
52 #     'CPU_DEPTH': game_config['CPU_DEPTH'],
53 #     'AWAITING_CPU': False,
54 #     'WINNER': None,
55 #     'PAUSED': False,
56 #     'ACTIVE_COLOUR': Colour.BLUE,
57 #     'TIME_ENABLED': game_config['TIME_ENABLED'],
58 #     'TIME': game_config['TIME'],
59 #     'MOVES': []
60 }
```

```

55 # }
56
57
58 #     move_item = {
59 #         'time': {
60 #             Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
61 #             Colour.RED: GAME_WIDGETS['red_timer'].get_time()
62 #         },
63 #         'move': move_notation,
64 #         'laserResult': laser_result
65 #     }

```

1.3.5 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.FRect(*mouse_pos, 1, 1)
46         return pygame.sprite.spritecollideany(test_sprite, self)

```

1.4 data\database

1.5 data\database\migrations

1.5.1 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()
```

1.5.2 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()

```

1.5.3 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()

```

1.5.4 change_fen_string_column_name_23122024.py

See Section ??.

1.5.5 create_games_table_19112024.py

See Section ??.

1.6 data\managers

1.6.1 animation.py

```

1  import pygame
2  from data.utils.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()

```

1.6.2 audio.py

```

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

```

1.6.3 cursor.py

```

1 import pygame
2 from data.assets import GRAPHICS
3 from data.constants import CursorMode
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['
16                arrow'], (32, 32)))
17            case CursorMode.IBEAM:
18                pygame.mouse.set_cursor((15, 5), pygame.transform.scale(GRAPHICS['
19                ibeam'], (32, 32)))
20            case CursorMode.OPENHAND:
21                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['
22                hand_open'], (32, 32)))
23            case CursorMode.CLOSEDHAND:
24                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['
25                hand_closed'], (32, 32)))
26            case CursorMode.NO:
27                pygame.mouse.set_visible(False)
28
29        self._mode = mode
30
31    def get_mode(self):
32        return self._mode
33
34 cursor = CursorManager()

```

1.6.4 logs.py

```

1 import logging.config
2 from data.utils.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)

```

1.6.5 shader.py

See Section ??.

1.6.6 theme.py

See Section ??.

1.6.7 window.py

```

1 import pygame
2 import moderngl
3 from data.constants import ShaderType, SCREEN_SIZE, SHADER_MAP
4 from data.utils.data_helpers import get_user_settings
5 from data.utils.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             else:
35                 from data.assets import GRAPHICS
36                 self._background_image = GRAPHICS['temp_background']
37
38     def set_effect(self, effect, **kwargs):
39         if is_opengl:
40             self._shader_manager.apply_shader(effect, **kwargs)
41
42     def set_apply_arguments(self, effect, **kwargs):
43         if is_opengl:
44             self.shader_arguments[effect] = kwargs
45
46     def clear_apply_arguments(self, effect):
47         if is_opengl:
48             self.shader_arguments[effect] = {}
49
50     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)

```

1.7 data\shaders

1.7.1 protocol.py

```

1  import pygame
2  import moderngl
3  from typing import Protocol, Optional
4  from data.constants import ShaderType
5
6  class SMPProtocol(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
11     ]]=None, filter: Optional[int]=moderngl.NEAREST) -> None: ...
12     def render_to_fbo(self, shader_type: ShaderType, texture: moderngl.Texture,
13     output_fbo: Optional[moderngl.Framebuffer] = None, program_type: Optional[
14     ShaderType] = None, use_image: Optional[bool] = True, **kwargs) -> None: ...
15     def apply_shader(self, shader_type: ShaderType, **kwargs) -> None: ...
16     def remove_shader(self, shader_type: ShaderType) -> None: ...
17     def render_output(self, texture: moderngl.Texture) -> None: ...
18     def get_fbo_texture(self, shader_type: ShaderType) -> moderngl.Texture: ...
19     def calibrate_pygame_surface(self, pygame_surface: pygame.Surface) -> moderngl
20     .Texture: ...
21     def draw(self, surface: pygame.Surface, arguments: dict) -> None: ...
22     def __del__(self) -> None: ...
23     def cleanup(self) -> None: ...
24     def handle_resize(self, new_screen_size: tuple[int]) -> None: ...
25
26     _ctx: moderngl.Context
27     _screen_size: tuple[int]
28     _opengl_buffer: moderngl.Buffer
29     _pygame_buffer: moderngl.Buffer
30     _shader_stack: list[ShaderType]
31
32     _vert_shaders: dict
33     _frag_shaders: dict
34     _programs: dict
35     _vaos: dict
36     _textures: dict

```

```

33     _shader_passes: dict
34     framebuffers: dict

```

1.8 data\shaders\classes

1.8.1 base.py

```

1  import pygame
2  from data.constants import ShaderType
3  from data.shaders.protocol import SMPProtocol
4
5  class Base:
6      def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.2 blend.py

```

1  import moderngl
2  from data.constants import ShaderType
3  from data.shaders.protocol import SMPProtocol
4
5  class _Blend:
6      def __init__(self, shader_manager: SMPProtocol):
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,
image2Pos=opengl_pos, relativeSize=relative_size)
19          self._shader_manager._ctx.blend_func = moderngl.DEFAULT_BLENDING

```

1.8.3 bloom.py

See Section ??.

1.8.4 blur.py

See Section ??.

1.8.5 chromatic_abbreviation.py

```

1  import pygame
2  from data.constants import ShaderType
3  from data.shaders.protocol import SMPProtocol
4
5  CHROMATIC_ABBREVIATION_INTENSITY = 2.0
6
7  class ChromaticAbbreviation:
8      def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.6 crop.py

```

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
3
4 class _Crop:
5     def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.7 crt.py

```

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
3
4 class CRT:
5     def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.8 grayscale.py

```

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
3
4 class Grayscale:
5     def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.9 highlight__brightness.py

```

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
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,
        texture, threshold=HIGHLIGHT_THRESHOLD, intensity=intensity)

```

1.8.10 highlight_colour.py

```

1 from data.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,
        highlight=1, colour=colour, threshold=0.1, intensity=intensity)

```

1.8.11 lightmap.py

```

1 from data.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,
        clamp=(-180, 180)):
14        self._shader_manager.create_framebuffer(ShaderType._LIGHTMAP, size=texture
        .size)
15        self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)
16
17        _Shadowmap(self._shader_manager).apply(texture, occlusion)
18        shadow_map = self._shader_manager.get_fbo_texture(ShaderType._SHADOWMAP)
19
20        self._shader_manager.render_to_fbo(ShaderType._LIGHTMAP, shadow_map,
        resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
        clamp, softShadow=softShadow)
21
22        self._shader_manager._ctx.disable(self._shader_manager._ctx.BLEND)

```

1.8.12 occlusion.py

```

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol

```

```

3
4 class _Occlusion:
5     def __init__(self, shader_manager: SMPProtocol):
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))

```

1.8.13 rays.py

See Section ??.

1.8.14 shadowmap.py

```

1 import moderngl
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMPProtocol
4 from data.shaders.classes.occlusion import _Occlusion
5
6 LIGHT_RESOLUTION = 256
7
8 class _Shadowmap:
9     def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.15 shake.py

```

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
3 from random import randint
4
5 SHAKE_INTENSITY = 3
6
7 class Shake:
8     def __init__(self, shader_manager: SMPProtocol):
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)

```

1.8.16 __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.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 }
```

1.9 data\shaders\fragments

1.9.1 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 LIGTHING 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;
30     float uv_len = length(uv);
31
32     float speed = (SPIN_ROTATION*SPIN_EASE*0.2);
33     if(IS_ROTATE){
34         speed = time * speed;
35     }
36     speed += 302.2;
37     float new_pixel_angle = atan(uv.y, uv.x) + speed - SPIN_EASE*20.*(1.*
38     SPIN_AMOUNT*uv_len + (1. - 1.*SPIN_AMOUNT));
39     vec2 mid = (screenSize.xy/length(screenSize.xy))/2.;
40     uv = (vec2((uv_len * cos(new_pixel_angle) + mid.x), (uv_len * sin(
41     new_pixel_angle) + mid.y)) - mid);
42
43     uv *= 30.;
44     speed = time*(SPIN_SPEED);
45     vec2 uv2 = vec2(uv.x+uv.y);
46
47     for(int i=0; i < 5; i++) {
48         uv2 += sin(max(uv.x, uv.y)) + uv;
49         uv += 0.5*vec2(cos(5.1123314 + 0.353*uv2.y + speed*0.131121),sin(uv2.x -
50         0.113*speed));
51         uv -= 1.0*cos(uv.x + uv.y) - 1.0*sin(uv.x*0.711 - uv.y);
52     }
53
54     float contrast_mod = (0.25*CONTRAST + 0.5*SPIN_AMOUNT + 1.2);
55     float paint_res = min(2., max(0.,length(uv)*(0.035)*contrast_mod));
56     float c1p = max(0.,1. - contrast_mod*abs(1.-paint_res));
57     float c2p = max(0.,1. - contrast_mod*abs(paint_res));
58     float c3p = 1. - min(1., c1p + c2p);
59     float light = (LIGTHING - 0.2)*max(c1p*5. - 4., 0.) + LIGTHING*max(c2p*5. -
60     4., 0.);
61     return (0.3/CONTRAST)*COLOUR_1 + (1. - 0.3/CONTRAST)*(COLOUR_1*c1p + COLOUR_2*
62     c2p + vec4(c3p*COLOUR_3.rgb, c3p*COLOUR_1.a)) + light;
63 }
64
65 void main() {
66     f_colour = effect(screenSize.xy, uvs* screenSize.xy);
67 }

```

1.9.2 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
    License.
22 vec2 hash( vec2 p )
23 {
24     p = vec2( dot(p,vec2(2127.1,81.17)), dot(p,vec2(1269.5,283.37)) );
25     return fract(sin(p)*43758.5453);
26 }
27
28 float noise( in vec2 p )
29 {
30     vec2 i = floor( p );
31     vec2 f = fract( p );
32
33     vec2 u = f*f*(3.0-2.0*f);
34
35     float n = mix( mix( dot( -1.0+2.0*hash( i + vec2(0.0,0.0) ), f - vec2(0.0,0.0)
36         ),
37         dot( -1.0+2.0*hash( i + vec2(1.0,0.0) ), f - vec2(1.0,0.0)
38         ), u.x),
39         mix( dot( -1.0+2.0*hash( i + vec2(0.0,1.0) ), f - vec2(0.0,1.0)
40         ),
41         dot( -1.0+2.0*hash( i + vec2(1.0,1.0) ), f - vec2(1.0,1.0)
42         ), u.x), u.y);
43     return 0.5 + 0.5*n;
44 }
45
46 void main() {
47     float ratio = screenSize.x / screenSize.y;
48
49     vec2 tuv = uvs;
50     tuv -= .5;
51
52     // rotate with Noise
53     float degree = noise(vec2(time*.1, tuv.x*tuv.y));
54
55     tuv.y *= 1./ratio;
56     tuv *= Rot(radians((degree-.5)*720.+180.));
57     tuv.y *= ratio;
58
59     // Wave warp with sin
60     float frequency = 5.;
61     float amplitude = 30.;
62     float speed = time * 2.;
63     tuv.x += sin(tuv.y*frequency+speed)/amplitude;
64     tuv.y += sin(tuv.x*frequency*1.5+speed)/(amplitude*.5);
65
66     // draw the image
67     vec3 colorYellow = vec3(.957, .804, .623);
68     vec3 colorDeepBlue = vec3(.192, .384, .933);
69     vec3 layer1 = mix(colorYellow, colorDeepBlue, S(-.3, .2, (tuv*Rot(radians(-5.)
70         )).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 }

```

1.9.3 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.xy * .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 lzt = 0.5+0.5*sn;
47     lzt = lzt*lzt*lzt*lzt*lzt;
48     float glow = pow(clamp(sn, 0.0, 1.0),100.0);

```

```

49     return lzf+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     f_colour = pow(col, vec4(0.75));
75 }

```

1.9.4 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 }

```

1.9.5 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)) *
        wave_time_mul * vi / t) * 2.0 * PI * wave_size);
46
47     if (d < s) i--;
48     if (d > s + 1.0 / t) i++;
49     i = clamp(i, 0.0, t - 1.0);
50
51     f_colour = lerp(top_color, bottom_color, i / (t - 1.0));
52 }

```

1.9.6 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 }

```

1.9.7 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 -
    image2Pos.y) / relativeSize.y);
19
20     float withinBounds = step(image2Pos.x, uvs.x) * step(uvs.x, (image2Pos.x +
    relativeSize.x)) * step(image2Pos.y, uvs.y) * step(uvs.y, (image2Pos.y +
    relativeSize.y));
21
22     f_colour = vec4(colour + (texture(image2, image2Coords).rgb * withinBounds),
    1.0);
23
24     // if (image2Pos.x < uvs.x &&
25     //     uvs.x < (image2Pos.x + relativeSize.x) &&
26     //     image2Pos.y < uvs.y &&
27     //     uvs.y < (image2Pos.y + relativeSize.y)) {
28
29     //     vec2 image2Coords = vec2((uvs.x - image2Pos.x) / relativeSize.x, (uvs.y
    - image2Pos.y) / relativeSize.y);
30     //     colour += texture(image2, image2Coords).rgb;
31     // }
32
33     // f_colour = vec4(colour, 1.0);
34 }

```

1.9.8 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 }

```

1.9.9 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 }

```

1.9.10 blur.frag

See Section ??.

1.9.11 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) /
                textureSize).rgba;

```

```

23
24         count += 1.0;
25     }
26 }
27
28     f_colour.rgb /= count;
29 }

```

1.9.12 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 }

```

1.9.13 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 }

```

1.9.14 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);
20 }

```

1.9.15 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 }

```

1.9.16 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 }

```

1.9.17 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 }

```

1.9.18 highlight_brightness.frag

See Section ??.

1.9.19 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.
18 g - normColour.g), threshold) * step(abs(pixel.b - normColour.b), threshold);
19
20     if (isClose == 1.0) {
21         f_colour = vec4(vec3(pixel.rgb * intensity), 1.0);
22     } else {
23         f_colour = vec4(texture(highlight, uvs).rgb, 1.0);
24     }
25 }

```

1.9.20 lightmap.frag

See Section ??.

1.9.21 occlusion.frag

See Section ??.

1.9.22 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 }

```

1.9.23 shadowmap.frag

See Section ??.

1.9.24 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 }

```

1.10 data\shaders\vertex

1.10.1 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 }
```

1.11 data\states

1.11.1 __init__.py

1.12 data\states\browser

1.12.1 browser.py

```
1 import pygame
2 import pyperclip
3 from data.constants import BrowserEventType, ShaderType, GAMES_PER_PAGE
4 from data.utils.database_helpers import delete_game, get_ordered_games
5 from data.states.browser.widget_dict import BROWSER_WIDGETS
6 from data.managers.logs import initialise_logger
7 from data.managers.window import window
8 from data.control import _State
9 from data.assets import MUSIC
10 from random import randint
11
12 logger = initialise_logger(__name__)
13
14 class Browser(_State):
15     def __init__(self):
16         super().__init__()
17
18         self._selected_index = None
19         self._filter_column = 'number_of_ply'
20         self._filter_ascend = False
21         self._games_list = []
22         self._page_number = 1
23
24     def cleanup(self):
25         super().cleanup()
26
27         if self._selected_index is not None:
28             return self._games_list[self._selected_index]
29
30         return None
31
32     def startup(self, persist=None):
33         self.refresh_games_list() # BEFORE RESIZE TO FILL WIDGET BEFORE RESIZING
34         super().startup(BROWSER_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
```

```

35
36         self._filter_column = 'number_of_ply'
37         self._filter_ascend = False
38
39         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
40
41         BROWSER_WIDGETS['help'].kill()
42         BROWSER_WIDGETS['browser_strip'].kill()
43
44         self.draw()
45
46     def refresh_games_list(self):
47         column_map = {
48             'moves': 'number_of_ply',
49             'winner': 'winner',
50             'time': 'created_dt'
51         }
52
53         ascend_map = {
54             'asc': True,
55             'desc': False
56         }
57
58         filter_column = BROWSER_WIDGETS['filter_column_dropdown'].
get_selected_word()
59         filter_ascend = BROWSER_WIDGETS['filter_ascend_dropdown'].
get_selected_word()
60
61         self._selected_index = None
62
63         start_row = (self._page_number - 1) * GAMES_PER_PAGE + 1
64         end_row = (self._page_number) * GAMES_PER_PAGE
65         self._games_list = get_ordered_games(column_map[filter_column], ascend_map
[filter_ascend], start_row=start_row, end_row=end_row)
66
67         BROWSER_WIDGETS['browser_strip'].initialise_games_list(self._games_list)
68         BROWSER_WIDGETS['browser_strip'].set_surface_size(window.size)
69         BROWSER_WIDGETS['scroll_area'].set_image()
70
71     def get_event(self, event):
72         widget_event = self._widget_group.process_event(event)
73
74         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
75             BROWSER_WIDGETS['help'].kill()
76
77         if widget_event is None:
78             return
79
80         match widget_event.type:
81             case BrowserEventType.MENU_CLICK:
82                 self.next = 'menu'
83                 self.done = True
84
85             case BrowserEventType.BROWSER_STRIP_CLICK:
86                 self._selected_index = widget_event.selected_index
87
88             case BrowserEventType.COPY_CLICK:
89                 if self._selected_index is None:
90                     return
91                 logger.info('COPYING TO CLIPBOARD:', self._games_list[self.
_selected_index]['fen_string'])

```



```

92         pyperclip.copy(self._games_list[self._selected_index]['fen_string']
93     ])
94
95     case BrowserEventType.DELETE_CLICK:
96         if self._selected_index is None:
97             return
98         delete_game(self._games_list[self._selected_index]['id'])
99         self.refresh_games_list()
100
101     case BrowserEventType.REVIEW_CLICK:
102         if self._selected_index is None:
103             return
104
105         self.next = 'review'
106         self.done = True
107
108     case BrowserEventType.FILTER_COLUMN_CLICK:
109         selected_word = BROWSER_WIDGETS['filter_column_dropdown'].
110         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'].
119         get_selected_word()
120
121         if selected_word is None:
122             return
123
124         self.refresh_games_list()
125
126     case BrowserEventType.PAGE_CLICK:
127         self._page_number = widget_event.data
128
129         self.refresh_games_list()
130
131     case BrowserEventType.HELP_CLICK:
132         self._widget_group.add(BROWSER_WIDGETS['help'])
133         self._widget_group.handle_resize(window.size)
134
135     def draw(self):
136         self._widget_group.draw()

```

1.12.2 widget_dict.py

```

1 from data.components.custom_event import CustomEvent
2 from data.constants import BrowserEventType, GAMES_PER_PAGE
3 from data.assets import GRAPHICS
4 from data.widgets import *
5 from data.utils.database_helpers import get_number_of_games
6
7 BROWSER_HEIGHT = 0.6
8
9 browser_strip = BrowserStrip(
10     relative_position=(0.0, 0.0),
11     relative_height=BROWSER_HEIGHT,
12     games_list=[]
13 )
14

```

```

15 number_of_pages = get_number_of_games() // GAMES_PER_PAGE + 1
16
17 carousel_widgets = {
18     i: Text(
19         relative_position=(0, 0),
20         relative_size=(0.3, 0.1),
21         text=f"PAGE {i} OF {number_of_pages}",
22         fill_colour=(0, 0, 0, 0),
23         fit_vertical=False,
24         border_width=0,
25     )
26     for i in range(1, number_of_pages + 1)
27 }
28
29 sort_by_container = Rectangle(
30     relative_size=(0.5, 0.1),
31     relative_position=(0.01, 0.77),
32     anchor_x='right',
33     visible=True
34 )
35
36 buttons_container = Rectangle(
37     relative_position=(0, 0.025),
38     relative_size=(0.5, 0.1),
39     scale_mode='height',
40     anchor_x='center'
41 )
42
43 top_right_container = Rectangle(
44     relative_position=(0, 0),
45     relative_size=(0.15, 0.075),
46     fixed_position=(5, 5),
47     anchor_x='right',
48     scale_mode='height'
49 )
50
51 BROWSER_WIDGETS = {
52     'help':
53     Icon(
54         relative_position=(0, 0),
55         relative_size=(1.02, 1.02),
56         icon=GRAPHICS['browser_help'],
57         anchor_x='center',
58         anchor_y='center',
59         border_width=0,
60         fill_colour=(0, 0, 0, 0)
61     ),
62     'default': [
63         buttons_container,
64         sort_by_container,
65         top_right_container,
66         ReactiveIconButton(
67             parent=top_right_container,
68             relative_position=(0, 0),
69             relative_size=(1, 1),
70             anchor_x='right',
71             scale_mode='height',
72             base_icon=GRAPHICS['home_base'],
73             hover_icon=GRAPHICS['home_hover'],
74             press_icon=GRAPHICS['home_press'],
75             event=CustomEvent(BrowserEventType.MENU_CLICK)
76         ),

```

```

77     ReactiveIconButton(
78         parent=top_right_container,
79         relative_position=(0, 0),
80         relative_size=(1, 1),
81         scale_mode='height',
82         base_icon=GRAPHICS['help_base'],
83         hover_icon=GRAPHICS['help_hover'],
84         press_icon=GRAPHICS['help_press'],
85         event=CustomEvent(BrowserEventType.HELP_CLICK)
86     ),
87     ReactiveIconButton(
88         parent=buttons_container,
89         relative_position=(0, 0),
90         relative_size=(1, 1),
91         scale_mode='height',
92         base_icon=GRAPHICS['copy_base'],
93         hover_icon=GRAPHICS['copy_hover'],
94         press_icon=GRAPHICS['copy_press'],
95         event=CustomEvent(BrowserEventType.COPY_CLICK),
96     ),
97     ReactiveIconButton(
98         parent=buttons_container,
99         relative_position=(0, 0),
100        relative_size=(1, 1),
101        scale_mode='height',
102        anchor_x='center',
103        base_icon=GRAPHICS['delete_base'],
104        hover_icon=GRAPHICS['delete_hover'],
105        press_icon=GRAPHICS['delete_press'],
106        event=CustomEvent(BrowserEventType.DELETE_CLICK),
107    ),
108    ReactiveIconButton(
109        parent=buttons_container,
110        relative_position=(0, 0),
111        relative_size=(1, 1),
112        scale_mode='height',
113        anchor_x='right',
114        base_icon=GRAPHICS['review_base'],
115        hover_icon=GRAPHICS['review_hover'],
116        press_icon=GRAPHICS['review_press'],
117        event=CustomEvent(BrowserEventType.REVIEW_CLICK),
118    ),
119    Text(
120        parent=sort_by_container,
121        relative_position=(0, 0),
122        relative_size=(0.3, 1),
123        fit_vertical=False,
124        text='SORT BY:',
125        border_width=0,
126        fill_colour=(0, 0, 0, 0)
127    )
128 ],
129 'browser_strip':
130     browser_strip,
131 'scroll_area':
132     ScrollArea(
133         relative_position=(0.0, 0.15),
134         relative_size=(1, BROWSER_HEIGHT),
135         vertical=False,
136         widget=browser_strip
137     ),
138 'filter_column_dropdown':

```

```

139     Dropdown(
140         parent=sort_by_container,
141         relative_position=(0.3, 0),
142         relative_height=0.75,
143         anchor_x='right',
144         word_list=['time', 'moves', 'winner'],
145         fill_colour=(255, 100, 100),
146         event=CustomEvent(BrowserEventType.FILTER_COLUMN_CLICK)
147     ),
148     'filter_ascend_dropdown':
149     Dropdown(
150         parent=sort_by_container,
151         relative_position=(0, 0),
152         relative_height=0.75,
153         anchor_x='right',
154         word_list=['desc', 'asc'],
155         fill_colour=(255, 100, 100),
156         event=CustomEvent(BrowserEventType.FILTER_ASCEND_CLICK)
157     ),
158     'page_carousel':
159     Carousel(
160         relative_position=(0.01, 0.77),
161         margin=5,
162         widgets_dict=carousel_widgets,
163         event=CustomEvent(BrowserEventType.PAGE_CLICK),
164     )
165 }

```

1.13 data\states\config

1.13.1 config.py

```

1  import pygame
2  from data.constants import ConfigEventType, Colour, ShaderType
3  from data.states.config.default_config import default_config
4  from data.states.config.widget_dict import CONFIG_WIDGETS
5  from data.managers.logs import initialise_logger
6  from data.managers.animation import animation
7  from data.managers.window import window
8  from data.managers.audio import audio
9  from data.managers.theme import theme
10 from data.assets import MUSIC, SFX
11 from data.control import _State
12 from random import randint
13
14 logger = initialise_logger(__name__)
15
16 class Config(_State):
17     def __init__(self):
18         super().__init__()
19
20         self._config = None
21         self._valid_fen = True
22         self._selected_preset = None
23
24     def cleanup(self):
25         super().cleanup()
26
27         window.clear_apply_arguments(ShaderType.BLOOM)
28
29         return self._config

```

```

30
31 def startup(self, persist=None):
32     super().startup(CONFIG_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
33     window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.9)
34
35     CONFIG_WIDGETS['invalid_fen_string'].kill()
36     CONFIG_WIDGETS['help'].kill()
37
38     self._config = default_config
39
40     if persist:
41         self._config['FEN_STRING'] = persist
42
43     self.set_fen_string(self._config['FEN_STRING'])
44     self.toggle_pvc(self._config['CPU_ENABLED'])
45     self.set_active_colour(self._config['COLOUR'])
46
47     CONFIG_WIDGETS['cpu_depth_carousel'].set_to_key(self._config['CPU_DEPTH'])
48     if self._config['CPU_ENABLED']:
49         self.create_depth_picker()
50     else:
51         self.remove_depth_picker()
52
53     self.draw()
54
55 def create_depth_picker(self):
56     # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.8))
57     # CONFIG_WIDGETS['start_button'].set_image()
58     CONFIG_WIDGETS['cpu_depth_carousel'].set_surface_size(window.size)
59     CONFIG_WIDGETS['cpu_depth_carousel'].set_image()
60     CONFIG_WIDGETS['cpu_depth_carousel'].set_geometry()
61     self._widget_group.add(CONFIG_WIDGETS['cpu_depth_carousel'])
62
63 def remove_depth_picker(self):
64     # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.7))
65     # CONFIG_WIDGETS['start_button'].set_image()
66
67     CONFIG_WIDGETS['cpu_depth_carousel'].kill()
68
69 def toggle_pvc(self, pvc_enabled):
70     if pvc_enabled:
71         CONFIG_WIDGETS['pvc_button'].set_locked(True)
72         CONFIG_WIDGETS['pvp_button'].set_locked(False)
73     else:
74         CONFIG_WIDGETS['pvp_button'].set_locked(True)
75         CONFIG_WIDGETS['pvc_button'].set_locked(False)
76
77     self._config['CPU_ENABLED'] = pvc_enabled
78
79     if self._config['CPU_ENABLED']:
80         self.create_depth_picker()
81     else:
82         self.remove_depth_picker()
83
84 def set_fen_string(self, new_fen_string):
85     CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
86     self._config['FEN_STRING'] = new_fen_string
87
88     self.set_preset_overlay(new_fen_string)
89
90     try:

```

```

91         CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
92         CONFIG_WIDGETS['invalid_fen_string'].kill()
93
94         if new_fen_string[-1].lower() == 'r':
95             self.set_active_colour(Colour.RED)
96         else:
97             self.set_active_colour(Colour.BLUE)
98
99         self._valid_fen = True
100     except:
101         CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
102         self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])
103
104         window.set_effect(ShaderType.SHAKE)
105         animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE))
106
107     )
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.MOUSEBUTTONDOWN, 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)

```

```

152
153         case ConfigEventType.PRESET_CLICK:
154             self.set_fen_string(widget_event.fen_string)
155
156         case ConfigEventType.SETUP_CLICK:
157             self.next = 'editor'
158             self.done = True
159
160         case ConfigEventType.COLOUR_CLICK:
161             self.set_active_colour(widget_event.data.get_flipped_colour())
162
163         case ConfigEventType.HELP_CLICK:
164             self._widget_group.add(CONFIG_WIDGETS['help'])
165             self._widget_group.handle_resize(window.size)
166
167     def set_preset_overlay(self, fen_string):
168         fen_string_widget_map = {
169             'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
170 PdNaFaNa3Sa b': 'preset_1',
171             'sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3/10/2
172 RaNaFaNa3Sa b': 'preset_2',
173             'sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3RaNaPa3Pc/4Fa5
174 /3PdNaPa3Sa b': 'preset_3'
175         }
176
177         if fen_string in fen_string_widget_map:
178             self._selected_preset = CONFIG_WIDGETS[fen_string_widget_map[
179 fen_string]]
180         else:
181             self._selected_preset = None
182
183     def set_active_colour(self, colour):
184         if self._config['COLOUR'] != colour:
185             CONFIG_WIDGETS['to_move_button'].set_next_icon()
186
187             self._config['COLOUR'] = colour
188
189             if colour == Colour.BLUE:
190                 CONFIG_WIDGETS['to_move_text'].set_text('BLUE TO MOVE')
191             elif colour == Colour.RED:
192                 CONFIG_WIDGETS['to_move_text'].set_text('RED TO MOVE')
193
194             if self._valid_fen:
195                 self._config['FEN_STRING'] = self._config['FEN_STRING'][:-1] + colour.
196 name[0].lower()
197                 CONFIG_WIDGETS['fen_string_input'].set_text(self._config['FEN_STRING'
198 ])
199
200     def draw(self):
201         self._widget_group.draw()
202
203         if self._selected_preset:
204             pygame.draw.rect(window.screen, theme['borderPrimary'], (*self.
205 _selected_preset.position, *self._selected_preset.size), width=int(theme['
206 borderWidth']))
207
208     def update(self, **kwargs):
209         self._widget_group.update()
210         super().update(**kwargs)

```

1.13.2 default_config.py

```

1 from data.constants import Colour
2
3 default_config = {
4     'CPU_ENABLED': False,
5     'CPU_DEPTH': 2,
6     'FEN_STRING': 'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
Pa2/2PdNaFaNa3Sa b',
7     'TIME_ENABLED': True,
8     'TIME': 5,
9     'COLOUR': Colour.BLUE,
10 }

```

1.13.3 widget_dict.py

```

1 import pygame
2 from data.widgets import *
3 from data.states.config.default_config import default_config
4 from data.components.custom_event import CustomEvent
5 from data.constants import ConfigEventType, Colour
6 from data.assets import GRAPHICS
7 from data.utils.asset_helpers import get_highlighted_icon
8 from data.managers.theme import theme
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(
GRAPHICS['timer'])}
30 else:
31     time_enabled_icons = {False: get_highlighted_icon(GRAPHICS['timer']), True:
GRAPHICS['timer']}
32
33 if default_config['COLOUR'] == Colour.BLUE:
34     colour_icons = {Colour.BLUE: GRAPHICS['pharoah_0_a'], Colour.RED: GRAPHICS['
pharoah_1_a']}
35 else:
36     colour_icons = {Colour.RED: GRAPHICS['pharoah_1_a'], Colour.BLUE: GRAPHICS['
pharoah_0_a']}
37
38 preview_container = Rectangle(
39     relative_position=(-0.15, 0),
40     relative_size=(0.65, 0.9),
41     anchor_x='center',
42     anchor_y='center',
43 )
44

```



```

45 config_container = Rectangle(
46     relative_position=(0.325, 0),
47     relative_size=(0.3, 0.9),
48     anchor_x='center',
49     anchor_y='center',
50 )
51
52 to_move_container = Rectangle(
53     parent=config_container,
54     relative_size=(0.9, 0.15),
55     relative_position=(0, 0.1),
56     anchor_x='center'
57 )
58
59 board_thumbnail = BoardThumbnail(
60     parent=preview_container,
61     relative_position=(0, 0),
62     relative_width=0.7,
63     scale_mode='width',
64     anchor_x='right',
65 )
66
67 top_right_container = Rectangle(
68     relative_position=(0, 0),
69     relative_size=(0.15, 0.075),
70     fixed_position=(5, 5),
71     anchor_x='right',
72     scale_mode='height'
73 )
74
75 CONFIG_WIDGETS = {
76     'help':
77         Icon(
78             relative_position=(0, 0),
79             relative_size=(1.02, 1.02),
80             icon=GRAPHICS['config_help'],
81             anchor_x='center',
82             anchor_y='center',
83             border_width=0,
84             fill_colour=(0, 0, 0, 0)
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',

```

```

1107         base_icon=GRAPHICS['help_base'],
1108         hover_icon=GRAPHICS['help_hover'],
1109         press_icon=GRAPHICS['help_press'],
1110         event=CustomEvent(ConfigEventType.HELP_CLICK)
1111     ),
1112     TextInput(
1113         parent=config_container,
1114         relative_position=(0.3, 0.3),
1115         relative_size=(0.65, 0.15),
1116         fit_vertical=True,
1117         placeholder='TIME CONTROL (DEFAULT 5)',
1118         default=str(default_config['TIME']),
1119         border_width=5,
1120         margin=20,
1121         validator=float_validator,
1122         event=CustomEvent(ConfigEventType.TIME_TYPE)
1123     ),
1124     Text(
1125         parent=config_container,
1126         fit_vertical=False,
1127         relative_position=(0.75, 0.3),
1128         relative_size=(0.2, 0.15),
1129         text='MINS',
1130         border_width=0,
1131         fill_colour=(0, 0, 0, 0)
1132     ),
1133     TextButton(
1134         parent=preview_container,
1135         relative_position=(0.3, 0),
1136         relative_size=(0.15, 0.15),
1137         text='CUSTOM',
1138         anchor_y='bottom',
1139         fit_vertical=False,
1140         margin=10,
1141         event=CustomEvent(ConfigEventType.SETUP_CLICK)
1142     )
1143 ],
1144 'board_thumbnail':
1145     board_thumbnail,
1146 'fen_string_input':
1147     TextInput(
1148         parent=preview_container,
1149         relative_position=(0, 0),
1150         relative_size=(0.55, 0.15),
1151         fit_vertical=False,
1152         placeholder='ENTER FEN STRING',
1153         default='sc3ncfcncpb2/2pc7/3Pd7/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
1154 Pa2/2PdNaFaNa3Sa b',
1155         border_width=5,
1156         anchor_y='bottom',
1157         anchor_x='right',
1158         margin=20,
1159         event=CustomEvent(ConfigEventType.FEN_STRING_TYPE)
1160     ),
1161 'start_button':
1162     TextButton(
1163         parent=config_container,
1164         relative_position=(0, 0),
1165         relative_size=(0.9, 0.3),
1166         anchor_y='bottom',
1167         anchor_x='center',
1168         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="s c3ncf cncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3
228         /7Pa2/2PdNaFaNa3Sa b",
229         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="sc3ncfcncra2/10/3Pd2pa3/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="sc3pcncpb3/5fc4/pa3pcncra3/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 }

```

1.14 data\states\editor

1.14.1 editor.py

```

1  import pygame
2  import pyperclip
3  from data.constants import EditorEventType, Colour, RotationDirection, Piece,
   Rotation
4  from data.states.game.components.bitboard_collection import BitboardCollection
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.states.game.components.father import DragAndDrop
9  from data.utils.bitboard_helpers import coords_to_bitboard
10 from data.states.editor.widget_dict import EDITOR_WIDGETS
11 from data.utils.board_helpers import screen_pos_to_coords
12 from data.managers.logs import initialise_logger
13 from data.managers.window import window
14 from data.control import _State
15
16 logger = initialise_logger(__name__)
17
18 class Editor(_State):
19     def __init__(self):
20         super().__init__()
21
22         self._bitboards = None
23         self._piece_group = None
24         self._selected_coords = None
25         self._selected_tool = None
26         self._selected_tool_colour = None
27         self._initial_fen_string = None
28         self._starting_colour = None
29
30         self._drag_and_drop = None
31         self._overlay_draw = None
32
33     def cleanup(self):
34         super().cleanup()
35
36         self.deselect_tool()
37
38     return encode_fen_string(self._bitboards)

```

```

39
40     def startup(self, persist):
41         super().startup(EDITOR_WIDGETS)
42         EDITOR_WIDGETS['help'].kill()
43
44         self._drag_and_drop = DragAndDrop(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
45         self._overlay_draw = OverlayDraw(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
46         self._bitboards = BitboardCollection(persist['FEN_STRING'])
47         self._piece_group = PieceGroup()
48
49         self._selected_coords = None
50         self._selected_tool = None
51         self._selected_tool_colour = None
52         self._initial_fen_string = persist['FEN_STRING']
53         self._starting_colour = Colour.BLUE
54
55         self.refresh_pieces()
56         self.set_starting_colour(Colour.BLUE if persist['FEN_STRING'][-1].lower()
== 'b' else Colour.RED)
57         self.draw()
58
59     @property
60     def selected_coords(self):
61         return self._selected_coords
62
63     @selected_coords.setter
64     def selected_coords(self, new_coords):
65         self._overlay_draw.set_selected_coords(new_coords)
66         self._selected_coords = new_coords
67
68     def get_event(self, event):
69         widget_event = self._widget_group.process_event(event)
70
71         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
72             EDITOR_WIDGETS['help'].kill()
73
74         if event.type == pygame.MOUSEBUTTONDOWN:
75             clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
76
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
, Rotation.UP)
90
91                 return
92
93         if event.type == pygame.MOUSEBUTTONUP:
94             clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
chessboard'].position, EDITOR_WIDGETS['chessboard'].size)

```

```

95
96         if self._drag_and_drop.dragged_sprite:
97             self.remove_dragged_piece(clicked_coords)
98             return
99
100     if widget_event is None:
101         if event.type == pygame.MOUSEBUTTONDOWN and self._widget_group.
on_widget(event.pos) is False:
102             self.selected_coords = None
103
104         return
105
106     match widget_event.type:
107         case None:
108             return
109
110         case EditorEventType.MENU_CLICK:
111             self.next = 'menu'
112             self.done = True
113
114         case EditorEventType.PICK_PIECE_CLICK:
115             if widget_event.piece == self._selected_tool and widget_event.
active_colour == self._selected_tool_colour:
116                 self.deselect_tool()
117             else:
118                 self.select_tool(widget_event.piece, widget_event.
active_colour)
119
120         case EditorEventType.ROTATE_PIECE_CLICK:
121             self.rotate_piece(widget_event.rotation_direction)
122
123         case EditorEventType.EMPTY_CLICK:
124             self._bitboards = BitboardCollection(fen_string='sc9
/10/10/10/10/10/10/9Sa b')
125             self.refresh_pieces()
126
127         case EditorEventType.RESET_CLICK:
128             self.reset_board()
129
130         case EditorEventType.COPY_CLICK:
131             logger.info(f'COPYING TO CLIPBOARD: {encode_fen_string(self.
_bitboards)}')
132             pyperclip.copy(encode_fen_string(self._bitboards))
133
134         case EditorEventType.BLUE_START_CLICK:
135             self.set_starting_colour(Colour.BLUE)
136
137         case EditorEventType.RED_START_CLICK:
138             self.set_starting_colour(Colour.RED)
139
140         case EditorEventType.START_CLICK:
141             self.next = 'config'
142             self.done = True
143
144         case EditorEventType.CONFIG_CLICK:
145             self.reset_board()
146             self.next = 'config'
147             self.done = True
148
149         case EditorEventType.ERASE_CLICK:
150             if self._selected_tool == 'ERASE':
151                 self.deselect_tool()

```

```

152         else:
153             self.select_tool('ERASE', None)
154
155         case EditorEventType.MOVE_CLICK:
156             if self._selected_tool == 'MOVE':
157                 self.deselect_tool()
158             else:
159                 self.select_tool('MOVE', None)
160
161         case EditorEventType.HELP_CLICK:
162             self._widget_group.add(EDITOR_WIDGETS['help'])
163             self._widget_group.handle_resize(window.size)
164
165     def reset_board(self):
166         self._bitboards = BitboardCollection(self._initial_fen_string)
167         self.refresh_pieces()
168
169     def refresh_pieces(self):
170         self._piece_group.initialise_pieces(self._bitboards.convert_to_piece_list
171         (), EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
172
173     def set_starting_colour(self, new_colour):
174         if new_colour == Colour.BLUE:
175             EDITOR_WIDGETS['blue_start_button'].set_locked(True)
176             EDITOR_WIDGETS['red_start_button'].set_locked(False)
177         elif new_colour == Colour.RED:
178             EDITOR_WIDGETS['blue_start_button'].set_locked(False)
179             EDITOR_WIDGETS['red_start_button'].set_locked(True)
180
181         if new_colour != self._starting_colour:
182             EDITOR_WIDGETS['blue_start_button'].set_next_icon()
183             EDITOR_WIDGETS['red_start_button'].set_next_icon()
184
185         self._starting_colour = new_colour
186         self._bitboards.active_colour = new_colour
187
188     def set_dragged_piece(self, coords):
189         bitboard_under_mouse = coords_to_bitboard(coords)
190         dragged_piece = self._bitboards.get_piece_on(bitboard_under_mouse, Colour.
191         BLUE) or self._bitboards.get_piece_on(bitboard_under_mouse, Colour.RED)
192
193         if dragged_piece is None:
194             return
195
196         dragged_colour = self._bitboards.get_colour_on(bitboard_under_mouse)
197         dragged_rotation = self._bitboards.get_rotation_on(bitboard_under_mouse)
198
199         self._drag_and_drop.set_dragged_piece(dragged_piece, dragged_colour,
200         dragged_rotation)
201         self._overlay_draw.set_hover_limit(False)
202
203     def remove_dragged_piece(self, coords):
204         piece, colour, rotation = self._drag_and_drop.get_dragged_info()
205
206         if coords and coords != self._selected_coords and piece != Piece.SPHINX:
207             self.remove_piece()
208             self.selected_coords = coords
209             self.set_piece(piece, colour, rotation)
210             self.selected_coords = None
211
212         self._drag_and_drop.remove_dragged_piece()
213         self._overlay_draw.set_hover_limit(True)

```



```

211
212 def set_piece(self, piece, colour, rotation):
213     if self.selected_coords is None or self.selected_coords == (0, 7) or self.
selected_coords == (9, 0):
214         return
215
216     self.remove_piece()
217
218     selected_bitboard = coords_to_bitboard(self.selected_coords)
219     self._bitboards.set_square(selected_bitboard, piece, colour)
220     self._bitboards.set_rotation(selected_bitboard, rotation)
221
222     self.refresh_pieces()
223
224 def remove_piece(self):
225     if self.selected_coords is None or self.selected_coords == (0, 7) or self.
selected_coords == (9, 0):
226         return
227
228     selected_bitboard = coords_to_bitboard(self.selected_coords)
229     self._bitboards.clear_square(selected_bitboard, Colour.BLUE)
230     self._bitboards.clear_square(selected_bitboard, Colour.RED)
231     self._bitboards.clear_rotation(selected_bitboard)
232
233     self.refresh_pieces()
234
235 def rotate_piece(self, rotation_direction):
236     if self.selected_coords is None or self.selected_coords == (0, 7) or self.
selected_coords == (9, 0):
237         return
238
239     selected_bitboard = coords_to_bitboard(self.selected_coords)
240
241     if self._bitboards.get_piece_on(selected_bitboard, Colour.BLUE) is None
and self._bitboards.get_piece_on(selected_bitboard, Colour.RED) is None:
242         return
243
244     current_rotation = self._bitboards.get_rotation_on(selected_bitboard)
245
246     if rotation_direction == RotationDirection.CLOCKWISE:
247         self._bitboards.update_rotation(selected_bitboard, selected_bitboard,
current_rotation.get_clockwise())
248     elif rotation_direction == RotationDirection.ANTICLOCKWISE:
249         self._bitboards.update_rotation(selected_bitboard, selected_bitboard,
current_rotation.get_anticlockwise())
250
251     self.refresh_pieces()
252
253 def select_tool(self, piece, colour):
254     dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: '
red_piece_buttons' }
255
256     self.deselect_tool()
257
258     if piece == 'ERASE':
259         EDITOR_WIDGETS['erase_button'].set_locked(True)
260         EDITOR_WIDGETS['erase_button'].set_next_icon()
261     elif piece == 'MOVE':
262         EDITOR_WIDGETS['move_button'].set_locked(True)
263         EDITOR_WIDGETS['move_button'].set_next_icon()
264     else:
265         EDITOR_WIDGETS[dict_name_map[colour]][piece].set_locked(True)

```

```

266         EDITOR_WIDGETS[dict_name_map[colour]][piece].set_next_icon()
267
268         self._selected_tool = piece
269         self._selected_tool_colour = colour
270
271     def deselect_tool(self):
272         dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: '
red_piece_buttons' }
273
274         if self._selected_tool:
275             if self._selected_tool == 'ERASE':
276                 EDITOR_WIDGETS['erase_button'].set_locked(False)
277                 EDITOR_WIDGETS['erase_button'].set_next_icon()
278             elif self._selected_tool == 'MOVE':
279                 EDITOR_WIDGETS['move_button'].set_locked(False)
280                 EDITOR_WIDGETS['move_button'].set_next_icon()
281             else:
282                 EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self.
_selected_tool].set_locked(False)
283                 EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self.
_selected_tool].set_next_icon()
284
285         self._selected_tool = None
286         self._selected_tool_colour = None
287
288     def handle_resize(self):
289         super().handle_resize()
290         self._piece_group.handle_resize(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
291         self._drag_and_drop.handle_resize(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
292         self._overlay_draw.handle_resize(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
293
294     def draw(self):
295         self._widget_group.draw()
296         self._overlay_draw.draw(window.screen)
297         self._piece_group.draw(window.screen)
298         self._drag_and_drop.draw(window.screen)

```

1.14.2 widget_dict.py

```

1 from data.constants import Piece, Colour, RotationDirection, EditorEventType,
BLUE_BUTTON_COLOURS
2 from data.utils.asset_helpers import get_highlighted_icon
3 from data.components.custom_event import CustomEvent
4 from data.assets import GRAPHICS
5 from data.widgets import *
6
7 blue_pieces_container = Rectangle(
8     relative_position=(0.25, 0),
9     relative_size=(0.13, 0.65),
10    scale_mode='height',
11    anchor_y='center',
12    anchor_x='center'
13 )
14
15 red_pieces_container = Rectangle(
16     relative_position=(-0.25, 0),
17     relative_size=(0.13, 0.65),
18     scale_mode='height',
19     anchor_y='center',

```

```

20     anchor_x='center'
21 )
22
23 bottom_actions_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 top_actions_container = Rectangle(
31     relative_position=(0, 0.05),
32     relative_size=(0.3, 0.1),
33     anchor_x='center',
34     scale_mode='height'
35 )
36
37 top_right_container = Rectangle(
38     relative_position=(0, 0),
39     relative_size=(0.15, 0.075),
40     fixed_position=(5, 5),
41     anchor_x='right',
42     scale_mode='height'
43 )
44
45 EDITOR_WIDGETS = {
46     'help':
47     Icon(
48         relative_position=(0, 0),
49         relative_size=(1.02, 1.02),
50         icon=GRAPHICS['editor_help'],
51         anchor_x='center',
52         anchor_y='center',
53         border_width=0,
54         fill_colour=(0, 0, 0, 0)
55     ),
56     'default': [
57         red_pieces_container,
58         blue_pieces_container,
59         bottom_actions_container,
60         top_actions_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(EditorEventType.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(EditorEventType.HELP_CLICK)

```

```

82     ),
83     ReactiveIconButton(
84         parent=bottom_actions_container,
85         relative_position=(0.06, 0),
86         relative_size=(1, 1),
87         anchor_x='center',
88         scale_mode='height',
89         base_icon=GRAPHICS['clockwise_arrow_base'],
90         hover_icon=GRAPHICS['clockwise_arrow_hover'],
91         press_icon=GRAPHICS['clockwise_arrow_press'],
92         event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
rotation_direction=RotationDirection.CLOCKWISE)
93     ),
94     ReactiveIconButton(
95         parent=bottom_actions_container,
96         relative_position=(-0.06, 0),
97         relative_size=(1, 1),
98         anchor_x='center',
99         scale_mode='height',
100        base_icon=GRAPHICS['anticlockwise_arrow_base'],
101        hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
102        press_icon=GRAPHICS['anticlockwise_arrow_press'],
103        event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
rotation_direction=RotationDirection.ANTICLOCKWISE)
104    ),
105    ReactiveIconButton(
106        parent=top_actions_container,
107        relative_position=(0, 0),
108        relative_size=(1, 1),
109        scale_mode='height',
110        anchor_x='right',
111        base_icon=GRAPHICS['copy_base'],
112        hover_icon=GRAPHICS['copy_hover'],
113        press_icon=GRAPHICS['copy_press'],
114        event=CustomEvent(EditorEventType.COPY_CLICK),
115    ),
116    ReactiveIconButton(
117        parent=top_actions_container,
118        relative_position=(0, 0),
119        relative_size=(1, 1),
120        scale_mode='height',
121        base_icon=GRAPHICS['delete_base'],
122        hover_icon=GRAPHICS['delete_hover'],
123        press_icon=GRAPHICS['delete_press'],
124        event=CustomEvent(EditorEventType.EMPTY_CLICK),
125    ),
126    ReactiveIconButton(
127        parent=top_actions_container,
128        relative_position=(0, 0),
129        relative_size=(1, 1),
130        scale_mode='height',
131        anchor_x='center',
132        base_icon=GRAPHICS['discard_arrow_base'],
133        hover_icon=GRAPHICS['discard_arrow_hover'],
134        press_icon=GRAPHICS['discard_arrow_press'],
135        event=CustomEvent(EditorEventType.RESET_CLICK),
136    ),
137    ReactiveIconButton(
138        relative_position=(0, 0),
139        fixed_position=(10, 0),
140        relative_size=(0.1, 0.1),
141        anchor_x='right',

```

```

142         anchor_y='center',
143         scale_mode='height',
144         base_icon=GRAPHICS['play_arrow_base'],
145         hover_icon=GRAPHICS['play_arrow_hover'],
146         press_icon=GRAPHICS['play_arrow_press'],
147         event=CustomEvent(EditorEventType.START_CLICK),
148     ),
149     ReactiveIconButton(
150         relative_position=(0, 0),
151         fixed_position=(10, 0),
152         relative_size=(0.1, 0.1),
153         anchor_y='center',
154         scale_mode='height',
155         base_icon=GRAPHICS['return_arrow_base'],
156         hover_icon=GRAPHICS['return_arrow_hover'],
157         press_icon=GRAPHICS['return_arrow_press'],
158         event=CustomEvent(EditorEventType.CONFIG_CLICK),
159     )
160 ],
161 'blue_piece_buttons': {},
162 'red_piece_buttons': {},
163 'erase_button':
164 MultipleIconButton(
165     parent=red_pieces_container,
166     relative_position=(0, 0),
167     relative_size=(0.2, 0.2),
168     scale_mode='height',
169     margin=10,
170     icons_dict={True: GRAPHICS['eraser'], False: get_highlighted_icon(GRAPHICS
171 ['eraser'])},
172     event=CustomEvent(EditorEventType.ERASE_CLICK),
173 ),
174 'move_button':
175 MultipleIconButton(
176     parent=blue_pieces_container,
177     relative_position=(0, 0),
178     relative_size=(0.2, 0.2),
179     scale_mode='height',
180     box_colours=BLUE_BUTTON_COLOURS,
181     icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
182 ['finger'])},
183     event=CustomEvent(EditorEventType.MOVE_CLICK),
184 ),
185 'chessboard':
186 Chessboard(
187     relative_position=(0, 0),
188     relative_width=0.4,
189     scale_mode='width',
190     anchor_x='center',
191     anchor_y='center'
192 ),
193 'blue_start_button':
194 MultipleIconButton(
195     parent=bottom_actions_container,
196     relative_position=(0, 0),
197     relative_size=(1, 1),
198     scale_mode='height',
199     anchor_x='right',
200     box_colours=BLUE_BUTTON_COLOURS,
201     icons_dict={False: get_highlighted_icon(GRAPHICS['pharoah_0_a']), True:
202 GRAPHICS['pharoah_0_a']},
203     event=CustomEvent(EditorEventType.BLUE_START_CLICK)

```

```

201     ),
202     'red_start_button':
203     MultipleIconButton(
204         parent=bottom_actions_container,
205         relative_position=(0, 0),
206         relative_size=(1, 1),
207         scale_mode='height',
208         icons_dict={True: GRAPHICS['pharoah_1_a'], False: get_highlighted_icon(
209             GRAPHICS['pharoah_1_a'])},
210         event=CustomEvent(EditorEventType.RED_START_CLICK)
211     )
212 }
213 for index, piece in enumerate([piece for piece in Piece if piece != Piece.SPHINX])
214 :
215     blue_icon = GRAPHICS[f'{piece.name.lower()}_0_a']
216     dimmed_blue_icon = get_highlighted_icon(blue_icon)
217     EDITOR_WIDGETS['blue_piece_buttons'][piece] = MultipleIconButton(
218         parent=blue_pieces_container,
219         relative_position=(0, (index + 1) / 5),
220         relative_size=(0.2, 0.2),
221         scale_mode='height',
222         box_colours=BLUE_BUTTON_COLOURS,
223         icons_dict={True: blue_icon, False: dimmed_blue_icon},
224         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
225             active_colour=Colour.BLUE)
226     )
227     red_icon = GRAPHICS[f'{piece.name.lower()}_1_a']
228     dimmed_red_icon = get_highlighted_icon(red_icon)
229     EDITOR_WIDGETS['red_piece_buttons'][piece] = MultipleIconButton(
230         parent=red_pieces_container,
231         relative_position=(0, (index + 1) / 5),
232         relative_size=(0.2, 0.2),
233         scale_mode='height',
234         icons_dict={True: red_icon, False: dimmed_red_icon},
235         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
236             active_colour=Colour.RED)
237     )
238 )

```

1.15 data\states\game

1.15.1 game.py

```

1  import pygame
2  from functools import partial
3  from data.states.game.mvc.game_controller import GameController
4  from data.utils.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.constants import ShaderType
14 from data.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     def get_fen_string(self):
30         game_entry = GameEntry(self.model.states, final_fen_string=self.model.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'] - 2]]
45         if persist['CPU_ENABLED'] else MUSIC['pvp']
46         super().startup(music=music)
47
48         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.BACKGROUND_LASERS)
49         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
50         binded_startup = partial(self.startup, persist)
51
52         self.model = GameModel(persist)
53         self.view = GameView(self.model)
54         self.pause_view = PauseView(self.model)
55         self.win_view = WinView(self.model)
56         self.controller = GameController(self.model, self.view, self.win_view, self.pause_view, self.switch_to_menu, self.switch_to_review, binded_startup)
57
58         self.view.draw()
59
60         audio.play_sfx(SFX['game_start_1'])
61         audio.play_sfx(SFX['game_start_2'])
62
63     def get_event(self, event):
64         self.controller.handle_event(event)
65
66     def handle_resize(self):
67         self.view.handle_resize()
68         self.win_view.handle_resize()
69         self.pause_view.handle_resize()
70
71     def draw(self):
72         self.view.draw()
73         self.win_view.draw()

```

```

72         self.pause_view.draw()
73
74     def update(self):
75         self.controller.check_cpu()
76         super().update()

```

1.15.2 widget_dict.py

```

1  from data.widgets import *
2  from data.components.custom_event import CustomEvent
3  from data.constants import GameEventType, RotationDirection, Colour
4  from data.assets import GRAPHICS
5
6  right_container = Rectangle(
7      relative_position=(0.05, 0),
8      relative_size=(0.2, 0.5),
9      anchor_y='center',
10     anchor_x='right',
11 )
12
13 rotate_container = Rectangle(
14     relative_position=(0, 0.05),
15     relative_size=(0.2, 0.1),
16     anchor_x='center',
17     anchor_y='bottom',
18 )
19
20 move_list = MoveList(
21     parent=right_container,
22     relative_position=(0, 0),
23     relative_width=1,
24     minimum_height=300,
25     move_list=[]
26 )
27
28 resign_button = TextButton(
29     parent=right_container,
30     relative_position=(0, 0),
31     relative_size=(0.5, 0.2),
32     fit_vertical=False,
33     anchor_y='bottom',
34     text="    Resign",
35     margin=5,
36     event=CustomEvent(GameEventType.RESIGN_CLICK)
37 )
38
39 draw_button = TextButton(
40     parent=right_container,
41     relative_position=(0, 0),
42     relative_size=(0.5, 0.2),
43     fit_vertical=False,
44     anchor_x='right',
45     anchor_y='bottom',
46     text="    Draw",
47     margin=5,
48     event=CustomEvent(GameEventType.DRAW_CLICK)
49 )
50
51 top_right_container = Rectangle(
52     relative_position=(0, 0),
53     relative_size=(0.225, 0.075),
54     fixed_position=(5, 5),

```



```

55     anchor_x='right',
56     scale_mode='height'
57 )
58
59 GAME_WIDGETS = {
60     'help':
61     Icon(
62         relative_position=(0, 0),
63         relative_size=(1.02, 1.02),
64         icon=GRAPHICS['game_help'],
65         anchor_x='center',
66         anchor_y='center',
67         border_width=0,
68         fill_colour=(0, 0, 0, 0)
69     ),
70     'tutorial':
71     Icon(
72         relative_position=(0, 0),
73         relative_size=(0.9, 0.9),
74         icon=GRAPHICS['game_tutorial'],
75         anchor_x='center',
76         anchor_y='center',
77     ),
78     'default': [
79         right_container,
80         rotate_container,
81         top_right_container,
82         ReactiveIconButton(
83             parent=top_right_container,
84             relative_position=(0, 0),
85             relative_size=(1, 1),
86             anchor_x='right',
87             scale_mode='height',
88             base_icon=GRAPHICS['home_base'],
89             hover_icon=GRAPHICS['home_hover'],
90             press_icon=GRAPHICS['home_press'],
91             event=CustomEvent(GameEventType.MENU_CLICK)
92         ),
93         ReactiveIconButton(
94             parent=top_right_container,
95             relative_position=(0, 0),
96             relative_size=(1, 1),
97             scale_mode='height',
98             base_icon=GRAPHICS['tutorial_base'],
99             hover_icon=GRAPHICS['tutorial_hover'],
100             press_icon=GRAPHICS['tutorial_press'],
101             event=CustomEvent(GameEventType.TUTORIAL_CLICK)
102         ),
103         ReactiveIconButton(
104             parent=top_right_container,
105             relative_position=(0.33, 0),
106             relative_size=(1, 1),
107             scale_mode='height',
108             base_icon=GRAPHICS['help_base'],
109             hover_icon=GRAPHICS['help_hover'],
110             press_icon=GRAPHICS['help_press'],
111             event=CustomEvent(GameEventType.HELP_CLICK)
112         ),
113         ReactiveIconButton(
114             parent=rotate_container,
115             relative_position=(0, 0),
116             relative_size=(1, 1),

```

```

117         scale_mode='height',
118         anchor_x='right',
119         base_icon=GRAPHICS['clockwise_arrow_base'],
120         hover_icon=GRAPHICS['clockwise_arrow_hover'],
121         press_icon=GRAPHICS['clockwise_arrow_press'],
122         event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
RotationDirection.CLOCKWISE)
123     ),
124     ReactiveIconButton(
125         parent=rotate_container,
126         relative_position=(0, 0),
127         relative_size=(1, 1),
128         scale_mode='height',
129         base_icon=GRAPHICS['anticlockwise_arrow_base'],
130         hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
131         press_icon=GRAPHICS['anticlockwise_arrow_press'],
132         event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
RotationDirection.ANTICLOCKWISE)
133     ),
134     resign_button,
135     draw_button,
136     Icon(
137         parent=resign_button,
138         relative_position=(0, 0),
139         relative_size=(0.75, 0.75),
140         fill_colour=(0, 0, 0, 0),
141         scale_mode='height',
142         anchor_y='center',
143         border_radius=0,
144         border_width=0,
145         margin=5,
146         icon=GRAPHICS['resign']
147     ),
148     Icon(
149         parent=draw_button,
150         relative_position=(0, 0),
151         relative_size=(0.75, 0.75),
152         fill_colour=(0, 0, 0, 0),
153         scale_mode='height',
154         anchor_y='center',
155         border_radius=0,
156         border_width=0,
157         margin=5,
158         icon=GRAPHICS['draw']
159     ),
160 ],
161 'scroll_area': # REMEMBER SCROLL AREA AFTER CONTAINER FOR RESIZING
162 ScrollArea(
163     parent=right_container,
164     relative_position=(0, 0),
165     relative_size=(1, 0.8),
166     vertical=True,
167     widget=move_list
168 ),
169 'move_list':
170     move_list,
171 'blue_timer':
172 Timer(
173     relative_position=(0.05, 0.05),
174     anchor_y='center',
175     relative_size=(0.1, 0.1),
176     active_colour=Colour.BLUE,

```

```

177         event=CustomEvent(GameEventType.TIMER_END),
178     ),
179     'red_timer':
180     Timer(
181         relative_position=(0.05, -0.05),
182         anchor_y='center',
183         relative_size=(0.1, 0.1),
184         active_colour=Colour.RED,
185         event=CustomEvent(GameEventType.TIMER_END),
186     ),
187     'status_text':
188     Text(
189         relative_position=(0, 0.05),
190         relative_size=(0.4, 0.1),
191         anchor_x='center',
192         fit_vertical=False,
193         margin=10,
194         text="g",
195         minimum_width=400
196     ),
197     'chessboard':
198     Chessboard(
199         relative_position=(0, 0),
200         anchor_x='center',
201         anchor_y='center',
202         scale_mode='width',
203         relative_width=0.4
204     ),
205     'blue_piece_display':
206     PieceDisplay(
207         relative_position=(0.05, 0.05),
208         relative_size=(0.2, 0.1),
209         anchor_y='bottom',
210         active_colour=Colour.BLUE
211     ),
212     'red_piece_display':
213     PieceDisplay(
214         relative_position=(0.05, 0.05),
215         relative_size=(0.2, 0.1),
216         active_colour=Colour.RED
217     )
218 }
219
220 PAUSE_WIDGETS = {
221     'default': [
222         TextButton(
223             relative_position=(0, -0.125),
224             relative_size=(0.3, 0.2),
225             anchor_x='center',
226             anchor_y='center',
227             text='GO TO MENU',
228             fit_vertical=False,
229             event=CustomEvent(GameEventType.MENU_CLICK)
230         ),
231         TextButton(
232             relative_position=(0, 0.125),
233             relative_size=(0.3, 0.2),
234             anchor_x='center',
235             anchor_y='center',
236             text='RESUME GAME',
237             fit_vertical=False,
238             event=CustomEvent(GameEventType.PAUSE_CLICK)

```

```

239     )
240 ]
241 }
242
243 win_container = Rectangle(
244     relative_position=(0, 0),
245     relative_size=(0.4, 0.8),
246     scale_mode='height',
247     anchor_x='center',
248     anchor_y='center',
249     fill_colour=(128, 128, 128, 200),
250     visible=True
251 )
252
253 WIN_WIDGETS = {
254     'default': [
255         win_container,
256         TextButton(
257             parent=win_container,
258             relative_position=(0, 0.5),
259             relative_size=(0.8, 0.15),
260             text='GO TO MENU',
261             anchor_x='center',
262             fit_vertical=False,
263             event=CustomEvent(GameEventType.MENU_CLICK)
264         ),
265         TextButton(
266             parent=win_container,
267             relative_position=(0, 0.65),
268             relative_size=(0.8, 0.15),
269             text='REVIEW GAME',
270             anchor_x='center',
271             fit_vertical=False,
272             event=CustomEvent(GameEventType.REVIEW_CLICK)
273         ),
274         TextButton(
275             parent=win_container,
276             relative_position=(0, 0.8),
277             relative_size=(0.8, 0.15),
278             text='NEW GAME',
279             anchor_x='center',
280             fit_vertical=False,
281             event=CustomEvent(GameEventType.GAME_CLICK)
282         ),
283     ],
284     'blue_won':
285     Icon(
286         parent=win_container,
287         relative_position=(0, 0.05),
288         relative_size=(0.8, 0.3),
289         anchor_x='center',
290         border_width=0,
291         margin=0,
292         icon=GRAPHICS['blue_won'],
293         fill_colour=(0, 0, 0, 0),
294     ),
295     'red_won':
296     Icon(
297         parent=win_container,
298         relative_position=(0, 0.05),
299         relative_size=(0.8, 0.3),
300         anchor_x='center',

```

```

301         border_width=0,
302         margin=0,
303         icon=GRAPHICS['red_won'],
304         fill_colour=(0, 0, 0, 0),
305         fit_icon=True,
306     ),
307     'draw_won':
308     Icon(
309         parent=win_container,
310         relative_position=(0, 0.05),
311         relative_size=(0.8, 0.3),
312         anchor_x='center',
313         border_width=0,
314         margin=0,
315         icon=GRAPHICS['draw_won'],
316         fill_colour=(0, 0, 0, 0),
317     ),
318     'by_checkmate':
319     Icon(
320         parent=win_container,
321         relative_position=(0, 0.375),
322         relative_size=(0.8, 0.1),
323         anchor_x='center',
324         border_width=0,
325         margin=0,
326         icon=GRAPHICS['by_checkmate'],
327         fill_colour=(0, 0, 0, 0),
328     ),
329     'by_resignation':
330     Icon(
331         parent=win_container,
332         relative_position=(0, 0.375),
333         relative_size=(0.8, 0.1),
334         anchor_x='center',
335         border_width=0,
336         margin=0,
337         icon=GRAPHICS['by_resignation'],
338         fill_colour=(0, 0, 0, 0),
339     ),
340     'by_draw':
341     Icon(
342         parent=win_container,
343         relative_position=(0, 0.375),
344         relative_size=(0.8, 0.1),
345         anchor_x='center',
346         border_width=0,
347         margin=0,
348         icon=GRAPHICS['by_draw'],
349         fill_colour=(0, 0, 0, 0),
350     ),
351     'by_timeout':
352     Icon(
353         parent=win_container,
354         relative_position=(0, 0.375),
355         relative_size=(0.8, 0.1),
356         anchor_x='center',
357         border_width=0,
358         margin=0,
359         icon=GRAPHICS['by_timeout'],
360         fill_colour=(0, 0, 0, 0),
361     )
362 }

```

1.16 data\states\game\components

1.16.1 bitboard_collection.py

See Section ??.

1.16.2 board.py

See Section ??.

1.16.3 capture_draw.py

```
1 from data.states.game.components.particles_draw import ParticlesDraw
2 from data.utils.board_helpers import coords_to_screen_pos
3 from data.constants import Colour, ShaderType
4 from data.managers.window import window
5 from data.managers.animation import animation
6
7 class CaptureDraw:
8     def __init__(self, board_position, board_size):
9         self._board_position = board_position
10        self._square_size = board_size[0] / 10
11        self._particles_draw = ParticlesDraw()
12
13    def add_capture(self, piece, colour, rotation, piece_coords, sphinx_coords,
14    active_colour, particles=True, shake=True):
15        if particles:
16            self._particles_draw.add_captured_piece(
17                piece,
18                colour,
19                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
37    def draw(self, screen):
38        self._particles_draw.draw(screen)
39
40    def update(self):
41        self._particles_draw.update()
42
43    def handle_resize(self, board_position, board_size):
44        self._board_position = board_position
45        self._square_size = board_size[0] / 10
```

1.16.4 father.py

```

1 import pygame
2 from data.constants import CursorMode
3 from data.states.game.components.piece_sprite import PieceSprite
4 from data.managers.cursor import cursor
5 from data.managers.audio import audio
6 from data.assets import SFX
7
8 DRAG_THRESHOLD = 500
9
10 class DragAndDrop:
11     def __init__(self, board_position, board_size, change_cursor=True):
12         self._board_position = board_position
13         self._board_size = board_size
14         self._change_cursor = change_cursor
15         self._ticks_since_drag = 0
16
17         self.dragged_sprite = None
18
19     def set_dragged_piece(self, piece, colour, rotation):
20         sprite = PieceSprite(piece=piece, colour=colour, rotation=rotation)
21         sprite.set_geometry((0, 0), self._board_size[0] / 10)
22         sprite.set_image()
23
24         self.dragged_sprite = sprite
25         self._ticks_since_drag = pygame.time.get_ticks()
26
27         if self._change_cursor:
28             cursor.set_mode(CursorMode.CLOSEDHAND)
29
30     def remove_dragged_piece(self):
31         self.dragged_sprite = None
32         time_dragged = pygame.time.get_ticks() - self._ticks_since_drag
33         self._ticks_since_drag = 0
34
35         if self._change_cursor:
36             cursor.set_mode(CursorMode.OPENHAND)
37
38         return time_dragged > DRAG_THRESHOLD
39
40     def get_dragged_info(self):
41         return self.dragged_sprite.type, self.dragged_sprite.colour, self.dragged_sprite.rotation
42
43     def draw(self, screen):
44         if self.dragged_sprite is None:
45             return
46
47         self.dragged_sprite.rect.center = pygame.mouse.get_pos()
48         screen.blit(self.dragged_sprite.image, self.dragged_sprite.rect.topleft)
49
50     def handle_resize(self, board_position, board_size):
51         if self.dragged_sprite:
52             self.dragged_sprite.set_geometry(board_position, board_size[0] / 10)
53
54         self._board_position = board_position
55         self._board_size = board_size

```

1.16.5 fen_parser.py

```

1 from data.constants import Colour, RotationIndex, Rotation, Piece, EMPTY_BB
2 from data.utils.bitboard_helpers import occupied_squares, print_bitboard,
   bitboard_to_index

```

```

3
4 def parse_fen_string(fen_string):
5     #sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
6     PdNaFaNa3Sa b
7     piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char: EMPTY_BB for
8     char in Piece}]
9     rotation_bitboards = [EMPTY_BB, EMPTY_BB]
10    combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
11    combined_all_bitboard = 0
12    part_1, part_2 = fen_string.split(' ')
13
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
40            case Rotation.LEFT:
41                rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
42            case _:
43                rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
44                raise ValueError('Invalid FEN String - piece character not
45                followed by rotational character')
46
47        file += 1
48    elif character in '0123456789':
49        if character == '1' and fen_string[index + 1] == '0':
50            file += 10
51            continue
52
53        file += int(character)
54    elif character == '/':
55        rank = rank - 1
56        file = 0
57    elif character in Rotation:
58        continue
59    else:
60        raise ValueError('Invalid FEN String - invalid character found:',
61        character)
62
63    if piece_count['s'] != 1 or piece_count['S'] != 1:
64        raise ValueError('Invalid FEN string - invalid number of Sphinx pieces')
65    # COMMENTED OUT AS NO PHAROAH PIECES IS OKAY IF PARSING FEN STRING FOR

```



```

60 FINISHED GAME BOARD THUMBNAIL
61 elif piece_count['f'] > 1 or piece_count['F'] > 1:
62     raise ValueError('Invalid FEN string - invalid number of Pharoah pieces')
63
64 if part_2 == 'b':
65     colour = Colour.BLUE
66 elif part_2 == 'r':
67     colour = Colour.RED
68 else:
69     raise ValueError('Invalid FEN string - invalid active colour')
70
71 for piece in Piece:
72     combined_colour_bitboards[Colour.BLUE] |= piece_bitboards[Colour.BLUE][
piece]
73     combined_colour_bitboards[Colour.RED] |= piece_bitboards[Colour.RED][piece
]
74
75 combined_all_bitboard = combined_colour_bitboards[Colour.BLUE] |
combined_colour_bitboards[Colour.RED]
76 return (piece_bitboards, combined_colour_bitboards, combined_all_bitboard,
rotation_bitboards, colour)
77
78 def encode_fen_string(bitboard_collection):
79     blue_bitboards = bitboard_collection.piece_bitboards[Colour.BLUE]
80     red_bitboards = bitboard_collection.piece_bitboards[Colour.RED]
81
82     fen_string_list = [''] * 80
83
84     for piece, bitboard in blue_bitboards.items():
85         for individual_bitboard in occupied_squares(bitboard):
86             index = bitboard_to_index(individual_bitboard)
87             rotation = bitboard_collection.get_rotation_on(individual_bitboard)
88             fen_string_list[index] = piece.upper() + rotation
89
90     for piece, bitboard in red_bitboards.items():
91         for individual_bitboard in occupied_squares(bitboard):
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     fen_string = fen_string[1:]
118
119     if bitboard_collection.active_colour == Colour.BLUE:
120         colour = 'b'
121     else:
122         colour = 'r'
123
124     return fen_string + ' ' + colour

```

1.16.6 laser.py

```

1 from data.utils import bitboard_helpers as bb_helpers
2 from data.constants import Piece, Colour, Rotation, A_FILE_MASK, J_FILE_MASK,
   ONE_RANK_MASK, EIGHT_RANK_MASK, EMPTY_BB
3 from data.utils.bitboard_helpers import print_bitboard
4
5 class Laser:
6     def __init__(self, bitboards):
7         self._bitboards = bitboards
8         self.hit_square_bitboard, self.piece_hit, self.laser_path, self.
path_bitboard, self.pieces_on_trajectory = self.calculate_trajectory()
9
10         if (self.hit_square_bitboard != EMPTY_BB):
11             self.piece_rotation = self._bitboards.get_rotation_on(self.
hit_square_bitboard)
12             self.piece_colour = self._bitboards.get_colour_on(self.
hit_square_bitboard)
13
14     def calculate_trajectory(self):
15         current_square = self._bitboards.get_piece_bitboard(Piece.SPHINX, self.
_bitboards.active_colour)
16         previous_direction = self._bitboards.get_rotation_on(current_square)
17         trajectory_bitboard = 0b0
18         trajectory_list = []
19         square_animation_states = []
20         pieces_on_trajectory = []
21
22         while current_square:
23             current_piece = self._bitboards.get_piece_on(current_square, Colour.
BLUE) or self._bitboards.get_piece_on(current_square, Colour.RED)
24             current_rotation = self._bitboards.get_rotation_on(current_square)
25
26             next_square, direction, piece_hit = self.calculate_next_square(
current_square, current_piece, current_rotation, previous_direction)
27
28             trajectory_bitboard |= current_square
29             trajectory_list.append(bb_helpers.bitboard_to_coords(current_square))
30             square_animation_states.append(direction)
31
32             if previous_direction != direction:
33                 pieces_on_trajectory.append(current_square)
34
35             if next_square == EMPTY_BB:
36                 hit_square_bitboard = 0b0
37
38                 if piece_hit:
39                     hit_square_bitboard = current_square
40
41             return hit_square_bitboard, piece_hit, list(zip(trajectory_list,
square_animation_states)), trajectory_bitboard, pieces_on_trajectory
42
43         current_square = next_square

```

```

44         previous_direction = direction
45
46     def calculate_next_square(self, square, piece, rotation, previous_direction):
47         match piece:
48             case Piece.SPHINX:
49                 if previous_direction != rotation:
50                     return EMPTY_BB, previous_direction, None
51
52                 next_square = self.next_square_bitboard(square, rotation)
53                 return next_square, previous_direction, Piece.SPHINX
54
55             case Piece.PYRAMID:
56                 if previous_direction in [rotation, rotation.get_clockwise()]:
57                     return EMPTY_BB, previous_direction, Piece.PYRAMID
58
59                 if previous_direction == rotation.get_anticlockwise():
60                     new_direction = previous_direction.get_clockwise()
61                 else:
62                     new_direction = previous_direction.get_anticlockwise()
63
64                 next_square = self.next_square_bitboard(square, new_direction)
65
66                 return next_square, new_direction, None
67
68             case Piece.ANUBIS:
69                 if previous_direction == rotation.get_clockwise().get_clockwise():
70                     return EMPTY_BB, previous_direction, None
71
72                 return EMPTY_BB, previous_direction, Piece.ANUBIS
73
74             case Piece.SCARAB:
75                 if previous_direction in [rotation.get_clockwise(), rotation.
76 get_anticlockwise()]:
77                     new_direction = previous_direction.get_anticlockwise()
78                 else:
79                     new_direction = previous_direction.get_clockwise()
80
81                 next_square = self.next_square_bitboard(square, new_direction)
82
83                 return next_square, new_direction, None
84
85             case Piece.PHAROAH:
86                 return EMPTY_BB, previous_direction, Piece.PHAROAH
87
88             case None:
89                 next_square = self.next_square_bitboard(square, previous_direction
90 )
91
92                 return next_square, previous_direction, None
93
94     def next_square_bitboard(self, src_bitboard, previous_direction):
95         match previous_direction:
96             case Rotation.UP:
97                 masked_src_bitboard = src_bitboard & EIGHT_RANK_MASK
98                 return masked_src_bitboard << 10
99             case Rotation.RIGHT:
100                 masked_src_bitboard = src_bitboard & J_FILE_MASK
101                 return masked_src_bitboard << 1
102             case Rotation.DOWN:
103                 masked_src_bitboard = src_bitboard & ONE_RANK_MASK
104                 return masked_src_bitboard >> 10
105             case Rotation.LEFT:

```

```

104         masked_src_bitboard = src_bitboard & A_FILE_MASK
105         return masked_src_bitboard >> 1

```

1.16.7 laser__draw.py

See Section ??.

1.16.8 move.py

```

1  from data.constants import MoveType, Colour, RotationDirection
2  from data.utils.bitboard_helpers import notation_to_bitboard, coords_to_bitboard,
   bitboard_to_coords, bitboard_to_notation, print_bitboard
3  import re
4  from data.managers.logs import initialise_logger
5
6  logger = initialise_logger(__name__)
7
8  class Move():
9      def __init__(self, move_type, src, dest=None, rotation_direction=None):
10         self.move_type = move_type
11         self.src = src
12         self.dest = dest
13         self.rotation_direction = rotation_direction
14
15     def to_notation(self, colour, piece, hit_square_bitboard):
16         hit_square = ''
17         if colour == Colour.BLUE:
18             piece = piece.upper()
19
20         if hit_square_bitboard:
21             hit_square = 'x' + bitboard_to_notation(hit_square_bitboard)
22
23         if self.move_type == MoveType.MOVE:
24             return 'M' + piece + bitboard_to_notation(self.src) +
bitboard_to_notation(self.dest) + hit_square
25         else:
26             return 'R' + piece + bitboard_to_notation(self.src) + self.
rotation_direction + hit_square
27
28     def __str__(self):
29         rotate_text = ''
30         coords_1 = '(' + chr(bitboard_to_coords(self.src)[0] + 65) + ',' + str(
bitboard_to_coords(self.src)[1] + 1) + ')'
31
32         if self.move_type == MoveType.ROTATE:
33             rotate_text = ' ' + self.rotation_direction.name
34             return f'{self.move_type.name}{rotate_text}: ON {coords_1}'
35
36         elif self.move_type == MoveType.MOVE:
37             coords_2 = '(' + chr(bitboard_to_coords(self.dest)[0] + 65) + ',' +
str(bitboard_to_coords(self.dest)[1] + 1) + ')'
38             return f'{self.move_type.name}{rotate_text}: FROM {coords_1} TO {
coords_2}'
39
40         # (Rotation: {self.rotation_direction})
41
42     @classmethod
43     def instance_from_notation(move_cls, notation):
44         try:
45             notation = notation.split('x')[0]
46             move_type = notation[0].lower()

```

```

47
48     moves = notation[2:]
49     letters = re.findall(r'[A-Za-z]+', moves)
50     numbers = re.findall(r'\d+', moves)
51
52     if move_type == MoveType.MOVE:
53         src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
54         dest_bitboard = notation_to_bitboard(letters[1] + numbers[1])
55
56         return move_cls(move_type, src_bitboard, dest_bitboard)
57
58     elif move_type == MoveType.ROTATE:
59         src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
60         rotation_direction = RotationDirection(letters[1])
61
62         return move_cls(move_type, src_bitboard, src_bitboard,
rotation_direction)
63     else:
64         raise ValueError('(Move.instance_from_notation) Invalid move type:
', move_type)
65
66     except Exception as error:
67         logger.info('(Move.instance_from_notation) Error occured while parsing
:', error)
68         raise error
69
70 @classmethod
71 def instance_from_input(move_cls, move_type, src, dest=None, rotation=None):
72     try:
73         if move_type == MoveType.MOVE:
74             src_bitboard = notation_to_bitboard(src)
75             dest_bitboard = notation_to_bitboard(dest)
76
77         elif move_type == MoveType.ROTATE:
78             src_bitboard = notation_to_bitboard(src)
79             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,
rotation_direction=None):
88     try:
89         src_bitboard = coords_to_bitboard(src_coords)
90         dest_bitboard = coords_to_bitboard(dest_coords)
91
92         return move_cls(move_type, src_bitboard, dest_bitboard,
rotation_direction)
93     except Exception as error:
94         logger.info('Error (Move.instance_from_coords):', error)
95         raise error
96
97 @classmethod
98 def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
None, rotation_direction=None):
99     try:
100         return move_cls(move_type, src_bitboard, dest_bitboard,
rotation_direction)
101     except Exception as error:

```

```

102         logger.info('Error (Move.instance_from_bitboards):', error)
103         raise error

```

1.16.9 overlay__draw.py

```

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

```

1.16.10 particles_draw.py

See Section ??.

1.16.11 piece_group.py

```

1  import pygame
2  from data.constants import EMPTY_BB, Colour, Piece
3  from data.states.game.components.piece_sprite import PieceSprite
4  from data.utils.board_helpers import coords_to_screen_pos
5  from data.utils import bitboard_helpers as bb_helpers
6
7  class PieceGroup(pygame.sprite.Group):
8      def __init__(self):
9          # self.square_list = []
10         # self.valid_square_list_positions = []
11         super().__init__()
12
13     def initialise_pieces(self, piece_list, board_position, board_size):
14         self.empty()
15
16         for index, piece_and_rotation in enumerate(piece_list):
17             x = index % 10
18             y = index // 10
19
20             if piece_and_rotation:
21                 if piece_and_rotation[0].isupper():
22                     colour = Colour.BLUE
23                 else:
24                     colour = Colour.RED
25
26             piece = PieceSprite(piece=Piece(piece_and_rotation[0].lower()),
colour=colour, rotation=piece_and_rotation[1])
27             piece.set_coors((x, y))
28             piece.set_geometry(board_position, board_size[0] / 10)
29             piece.set_image()
30             self.add(piece)
31
32     def set_geometry(self, board_position, board_size):
33         for sprite in self.sprites():

```

```

34         sprite.set_geometry(board_position, board_size[0] / 10)
35
36     def handle_resize(self, board_position, board_size):
37         self.set_geometry(board_position, board_size)
38
39         for sprite in self.sprites():
40             sprite.set_image()
41
42     def remove_piece(self, coords):
43         for sprite in self.sprites():
44             if sprite.coords == coords:
45                 sprite.kill()
46
47     # def handle_resize_end(self):
48     #     for sprite in self.sprites():
49     #         sprite.handle_resize_end()
50
51     # def clear_square(self, src_bitboard):
52     #     list_position = bb_helpers.bitboard_to_index(src_bitboard)
53     #     self.square_list[list_position].clear_piece()
54
55     # def update_squares_move(self, src, dest, new_piece_symbol, new_colour,
56     rotation):
57     #     self.square_list[src].clear_piece()
58     #     self.square_list[dest].clear_piece()
59     #     self.square_list[dest].set_piece(piece_symbol=new_piece_symbol, colour=
60 new_colour, rotation=rotation)
61
62     # def update_squares_rotate(self, src, piece_symbol, colour, new_rotation):
63     #     self.square_list[src].clear_piece()
64     #     self.square_list[src].set_piece(piece_symbol=piece_symbol, colour=colour
65 , rotation=new_rotation)
66
67     # def add_valid_square_overlays(self, valid_bitboard):
68     #     if valid_bitboard == EMPTY_BB:
69     #         return
70
71     #     list_positions = self.bitboard_to_list_positions(valid_bitboard)
72     #     self.valid_square_list_positions = list_positions
73
74     #     for square_position in list_positions:
75     #         square = self.square_list[square_position]
76     #         square.selected = True
77
78     # def remove_valid_square_overlays(self):
79     #     for square_position in self.valid_square_list_positions:
80     #         square = self.square_list[square_position]
81     #         square.selected = False
82     #         square.remove_overlay()
83
84     #     self.valid_square_list_positions = []
85
86     # def draw_valid_square_overlays(self):
87     #     for square_position in self.valid_square_list_positions:
88     #         square = self.square_list[square_position]
89     #         square.draw_overlay()
90
91     # def bitboard_to_list_positions(self, bitboard):
92     #     list_positions = []
93
94     #     for square in bb_helpers.occupied_squares(bitboard):
95     #         list_positions.append(bb_helpers.bitboard_to_index(square))

```



```

93
94     #         return list_positions

```

1.16.12 piece_sprite.py

```

1  import pygame
2  from data.assets import GRAPHICS
3  from data.constants import Colour, Piece
4  from data.utils.asset_helpers import scale_and_cache
5  from data.utils.board_helpers import coords_to_screen_pos
6
7  class EmptyPiece(pygame.sprite.Sprite):
8      def __init__(self):
9          super().__init__()
10
11          self.image = pygame.Surface((1, 1))
12          self.rect = self.image.get_rect()
13          self.rect.topleft = (0, 0)
14
15      def set_image(self, type):
16          pass
17
18      def set_rect(self):
19          pass
20
21      def set_geometry(self, anchor_position, size):
22          pass
23
24  class PieceSprite(pygame.sprite.Sprite):
25      def __init__(self, piece, colour, rotation):
26          super().__init__()
27          self.colour = colour
28          self.rotation = rotation
29
30          self.type = piece
31          self.coords = None
32          self.size = None
33
34      @property
35      def image_name(self):
36          return Piece(self.type).name.lower() + '_' + str(self.colour) + '_' + self
37              .rotation
38
39      def set_image(self):
40          self.image = scale_and_cache(GRAPHICS[self.image_name], (self.size, self.
41              size))
42
43      def set_geometry(self, new_position, square_size):
44          self.size = square_size
45          self.rect = pygame.FRect((0, 0, square_size, square_size))
46
47          if self.coords:
48              self.rect.topleft = coords_to_screen_pos(self.coords, new_position,
49                  square_size)
50          else:
51              self.rect.topleft = new_position
52
53      def set_coords(self, new_coords):
54          self.coords = new_coords

```

1.16.13 psqt.py

```

1 from data.constants 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 }

```

1.17 data\states\game\cpu

1.17.1 arena.py

```

1 from data.states.game.cpu.engines import *

```

```

2 from data.states.game.components.board import Board
3 from data.constants import Colour, Miscellaneous
4 from data.managers.logs import initialise_logger
5
6 logger = initialise_logger(__name__)
7 # sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa
8   b
9 # scfaRa7/RaRaRaFa6/RaRaRa7/10/10/10/10/9Sa b
10 # scfa8/10/10/10/10/10/10/8FaSa b
11
12 def compare(cls1, cls2, depth, rounds):
13     wins = [0, 0]
14
15     board = Board()
16     def callback(move):
17         board.apply_move(move, add_hash=True)
18
19     cpu1 = cls1(callback=callback, max_depth=depth, verbose='compact')
20     cpu2 = cls2(callback=callback, max_depth=depth, verbose='compact')
21
22     for i in range(rounds):
23         board = Board(fen_string="scfa8/10/10/10/10/10/10/8FaSa b")
24         ply = 0
25
26         if i % 2 == 0:
27             players = { Colour.BLUE: cpu1, Colour.RED: cpu2, Miscellaneous.DRAW: '
28             DRAW' }
29         else:
30             players = { Colour.BLUE: cpu2, Colour.RED: cpu1, Miscellaneous.DRAW: '
31             DRAW' }
32
33         while (winner := board.check_win()) is None:
34             players[board.get_active_colour()].find_move(board, None)
35             ply += 1
36             logger.debug('PLY:', ply)
37
38         if winner == Miscellaneous.DRAW:
39             wins[0] += 0.5
40             wins[1] += 0.5
41         else:
42             if players[winner] == cpu1:
43                 wins[0] += 1
44             else:
45                 wins[1] += 1
46
47         logger.debug(f'ROUND {i + 1} | WINNER: {players[winner]} | PLY: {ply}')
48
49     logger.debug(f'{cpu1} SCORE: {wins[0]} | {cpu2} SCORE: {wins[1]}')
50
51 compare(TTNegamaxCPU, TTNegamaxCPU, 2, 1)

```

1.17.2 base.py

```

1 import time
2 from pprint import PrettyPrinter
3 from data.constants 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['
time_taken']), 3)
39         self._stats['ms_per_node'] = round(self._stats['time_taken'] / self._stats
['nodes'], 3)
40
41         # Prints stats across multiple lines
42         if self._verbose is True:
43             logger.info(f'\n\n'
44                 f'{self.__str__()} Search Results:\n'
45                 f'{printer.pformat(self._stats)}\n'
46                 f'Best score: {score}    Best move: {move}\n'
47                 )
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 Exception(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__

```

1.17.3 cpu_thread.py

See Section ??.

1.17.4 evaluator.py

See Section ??.

1.17.5 move_orderer.py

```

1 from data.states.game.cpu.evaluator import Evaluator
2 from data.constants import Colour
3 from data.utils.bitboard_helpers import print_bitboard, pop_count
4
5 class SimpleEvaluator:
6     def __init__(self):
7         self._evaluator = Evaluator(verbose=False)
8         self._cache = {}
9
10    def evaluate(self, board):
11        if (hashed := board.to_hash()) in self._cache:
12            return self._cache[hashed]
13
14        score = self._evaluator.evaluate_material(board, board.get_active_colour()
15    )
16        self._cache[hashed] = score
17
18        return score
19
20 class MoveOrderer:
21     def __init__(self):
22         self._evaluator = SimpleEvaluator()
23
24     # def get_eval(self, board, move):
25     #     laser_result = board.apply_move(move)
26     #     score = self._evaluator.evaluate(board)
27     #     board.undo_move(move, laser_result)
28     #     return score
29
30     # def score_moves(self, board, moves):
31     #     for i in range(len(moves)):
32         score = self.get_eval(board, moves[i])

```

```

32         #         moves[i] = (moves[i], score)
33
34         #         return moves
35
36     def best_move_to_front(self, moves, start_idx, hint):
37         for i in range(start_idx + 1, len(moves)):
38             if moves[i].src in hint:
39                 moves[i], moves[start_idx] = moves[start_idx], moves[i]
40             return
41
42     def get_moves(self, board, hint=None):
43         colour = board.get_active_colour()
44         moves = list(board.generate_all_moves(colour))
45
46         for i in range(len(moves)):
47             if hint:
48                 self.best_move_to_front(moves, i, hint)
49
50         yield moves[i]

```

1.17.6 temp.py

```

1  from data.constants import Score, Colour
2  from data.states.game.cpu.transposition_table import TranspositionTable
3  from data.states.game.cpu.base import BaseCPU
4  from pprint import pprint
5
6  class MinimaxCPU(BaseCPU):
7      def __init__(self, max_depth, callback, verbose):
8          super().__init__(callback, verbose)
9          self._max_depth = max_depth
10
11     def find_move(self, board, stop_event):
12         # No bit_length bug as None type returned, so Move __str__ called on
13         # NoneType I think (just deal with None being returned)
14         try:
15             best_move = self.search(board, self._max_depth, -Score.INFINITE, Score
16             .INFINITE, stop_event)
17
18             if self._verbose:
19                 print('\nCPU Search Results:')
20                 pprint(self._stats)
21                 print('Best move:', best_move, '\n')
22
23             self._callback(self._best_move)
24         except Exception as error:
25             print('(MinimaxBase.find_move) Error has occurred:')
26             raise error
27
28     def search(self, board, depth, alpha, beta, stop_event):
29         if stop_event.is_set():
30             raise Exception('Thread killed - stopping minimax function (CPU.
31             minimax)')
32
33         # cached_move, cached_score = self._transposition_table.get_entry(hash_key
34         # =board.bitboards.get_hash(), depth=depth, alpha=alpha, beta=beta)
35         # if cached_move or cached_score:
36         #     if depth == self._max_depth:
37         #         self._best_move = cached_move
38         #     return cached_score

```

```

37         if depth == 0:
38             return self.evaluate(board)
39
40         is_maximiser = board.get_active_colour() == Colour.BLUE
41
42         if is_maximiser:
43             score = -Score.INFINITE
44
45             for move in board.generate_all_moves(board.get_active_colour()):
46                 before, before_score = board.bitboards.get_rotation_string(), self
47                 .evaluate(board)
48
49                 laser_result = board.apply_move(move)
50                 new_score = self.minimax(board, depth - 1, alpha, beta, False,
51                 stop_event)
52
53                 if new_score >= score:
54                     score = new_score
55
56                     if depth == self._max_depth:
57                         self._best_move = move
58
59                     board.undo_move(move, laser_result)
60
61                     alpha = max(alpha, score)
62                     if depth == self._max_depth: # https://stackoverflow.com/questions/31429974/alphabeta-pruning-alpha-equals-or-greater-than-beta-why-equals
63                         if beta < alpha:
64                             break
65                     else:
66                         if beta <= alpha:
67                             break
68
69                     after, after_score = board.bitboards.get_rotation_string(), self.
70                     evaluate(board)
71                     if (before != after or before_score != after_score):
72                         print('shit\n\n')
73
74                     return score
75
76         else:
77             score = Score.INFINITE
78
79             for move in board.generate_all_moves(board.get_active_colour()):
80                 bef, before_score = board.bitboards.get_rotation_string(), self.
81                 evaluate(board)
82
83                 laser_result = board.apply_move(move)
84                 new_score = self.minimax(board, depth - 1, alpha, beta, False,
85                 stop_event)
86
87                 if new_score <= score:
88                     score = new_score
89                     if depth == self._max_depth:
90                         self._best_move = move
91
92                     board.undo_move(move, laser_result)
93
94                     beta = min(beta, score)
95                     if depth == self._max_depth:
96                         if beta < alpha:
97                             break

```

```

93         else:
94             if beta <= alpha:
95                 break
96
97         after, after_score = board.bitboards.get_rotation_string(), self.
evaluate(board)
98         if (bef != after or before_score != after_score):
99             print('shit\n\n')
100             raise ValueError
101
102         return score

```

1.17.7 transposition_table.py

See Section ??.

1.17.8 zobrist_hasher.py

See Section ??.

1.18 data\states\game\cpu\engines

1.18.1 alpha_beta.py

See Section ??.

1.18.2 iterative_deepening.py

```

1 from data.states.game.cpu.engines.transposition_table import
TranspositionTableMixin
2 from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU, ABNegamaxCPU
3 from data.constants import Score
4
5 class IterativeDeepeningMixin:
6     def find_move(self, board, stop_event):
7         best_move = None
8
9         for depth in range(1, self._max_depth + 1):
10             self.initialise_stats()
11             self._stats['ID_depth'] = depth
12
13             best_score, best_move = self.search(board, depth, -Score.INFINITE,
Score.INFINITE, stop_event)
14
15             if self._verbose:
16                 self.print_stats(best_score, best_move)
17
18             self._callback(best_move)
19
20 class IDMinimaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABMinimaxCPU)
:
21     def initialise_stats(self):
22         super().initialise_stats()
23         self._stats['cache_hits'] = 0
24
25     def print_stats(self, score, move):
26         self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
self._stats['nodes'], 3)

```



```

27         self._stats['cache_entries'] = len(self._table._table)
28         super().print_stats(score, move)
29
30 class IDNegamaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABNegamaxCPU)
31 :
32     def initialise_stats(self):
33         super().initialise_stats()
34         self._stats['cache_hits'] = 0
35
36     def print_stats(self, score, move):
37         self._stats['cache_hits_percentage'] = self._stats['cache_hits'] / self._stats['nodes']
38         self._stats['cache_entries'] = len(self._table._table)
39         super().print_stats(score, move)

```

1.18.3 minimax.py

See Section ??.

1.18.4 negamax.py

```

1 from data.constants import Score, Colour, Miscellaneous, MoveType
2 from data.states.game.cpu.base import BaseCPU
3 from data.utils.bitboard_helpers import print_bitboard, is_occupied
4 from random import choice, randint
5 from copy import deepcopy
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

```

1.18.5 simple.py

```

1 from data.states.game.cpu.base import BaseCPU
2 from data.constants 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.
search)')
20
21        active_colour = board.bitboards.active_colour
22        best_score = -Score.INFINITE if active_colour == Colour.BLUE else Score.
INFINITE
23        best_move = None
24
25        for move in board.generate_all_moves(active_colour):
26            laser_result = board.apply_move(move)
27
28            self._stats['nodes'] += 1
29
30            if winner := board.check_win() is not None:
31                self.process_win(winner)
32            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 (
active_colour == Colour.RED and score < best_score):
38                best_move = move
39                best_score = score
40
41            board.undo_move(move, laser_result)
42
43        return best_score, best_move

```

1.18.6 transposition_table.py

See Section ??.

1.18.7 __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, ABNegamaxCPU
5 from data.states.game.cpu.engines.iterative_deepening import IDMinimaxCPU,
  IDNegamaxCPU
6 from data.states.game.cpu.engines.transposition_table import TTMinimaxCPU,
  TTNegamaxCPU

```

1.19 data\states\game\mvc

1.19.1 game_controller.py

See Section ??.

1.19.2 game_model.py

See Section ??.

1.19.3 game_view.py

See Section ??.

1.19.4 pause_view.py

```

1 import pygame
2 from data.states.game.widget_dict import PAUSE_WIDGETS
3 from data.constants import GameEventType, PAUSE_COLOUR
4 from data.components.widget_group import WidgetGroup
5 from data.managers.window import window
6 from data.managers.audio import audio
7
8 class PauseView:
9     def __init__(self, model):
10         self._model = model
11
12         self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
13         self._screen_overlay.fill(PAUSE_COLOUR)
14
15         self._widget_group = WidgetGroup(PAUSE_WIDGETS)
16         self._widget_group.handle_resize(window.size)
17
18         self._model.register_listener(self.process_model_event, 'pause')
19
20         self._event_to_func_map = {
21             GameEventType.PAUSE_CLICK: self.handle_pause_click
22         }
23
24         self.states = {
25             'PAUSED': False
26         }
27
28     def handle_pause_click(self, event):
29         self.states['PAUSED'] = not self.states['PAUSED']
30
31         if self.states['PAUSED']:
32             audio.pause_sfx()
33         else:
34             audio.unpause_sfx()
35

```

```

36     def handle_resize(self):
37         self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
38         self._screen_overlay.fill(PAUSE_COLOUR)
39         self._widget_group.handle_resize(window.size)
40
41     def draw(self):
42         if self.states['PAUSED']:
43             window.screen.blit(self._screen_overlay, (0, 0))
44             self._widget_group.draw()
45
46     def process_model_event(self, event):
47         try:
48             self._event_to_func_map.get(event.type)(event)
49         except:
50             raise KeyError('Event type not recognized in Paused View (PauseView.
process_model_event)', event)
51
52     def convert_mouse_pos(self, event):
53         return self._widget_group.process_event(event)

```

1.19.5 win_view.py

```

1  from data.constants 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)

```

1.20 data\states\menu

1.20.1 menu.py

```

1  import pygame
2  import sys
3  from random import randint
4  from data.utils.asset_helpers import get_rotational_angle
5  from data.states.menu.widget_dict import MENU_WIDGETS
6  from data.constants import MenuEventType, ShaderType
7  from data.utils.asset_helpers import scale_and_cache
8  from data.managers.logs import initialise_logger
9  from data.managers.animation import animation
10 from data.assets import GRAPHICS, MUSIC, SFX
11 from data.managers.window import window
12 from data.managers.audio import audio
13 from data.control import _State
14
15 logger = initialise_logger(__file__)
16
17 class Menu(_State):
18     def __init__(self):
19         super().__init__()
20         self._fire_laser = False
21         self._bloom_mask = None
22         self._laser_mask = None
23
24     def cleanup(self):
25         super().cleanup()
26
27         window.clear_apply_arguments(ShaderType.BLOOM)
28         window.clear_apply_arguments(ShaderType.SHAKE)
29         window.clear_effect(ShaderType.CHROMATIC_ABBREVIATION)
30
31         return None
32
33     def startup(self, persist=None):
34         super().startup(MENU_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
35         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
36         window.set_effect(ShaderType.CHROMATIC_ABBREVIATION)
37
38         MENU_WIDGETS['credits'].kill()
39
40         self._fire_laser = False
41         self._bloom_mask = None
42         self._laser_mask = None
43
44         self.draw()

```

```

45         self.update_masks()
46
47     @property
48     def sphinx_center(self):
49         return (window.size[0] - self.sphinx_size[0] / 2, window.size[1] - self.
sphinx_size[1] / 2)
50
51     @property
52     def sphinx_size(self):
53         return (min(window.size) * 0.1, min(window.size) * 0.1)
54
55     @property
56     def sphinx_rotation(self):
57         mouse_pos = (pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1] + 0.01)
58         return -get_rotational_angle(mouse_pos, self.sphinx_center)
59
60     def get_event(self, event):
61         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
62             MENU_WIDGETS['credits'].kill()
63
64         if event.type == pygame.MOUSEBUTTONDOWN:
65             self._fire_laser = True
66             audio.play_sfx(SFX['menu_laser_windup'])
67             audio.play_sfx(SFX['menu_laser_loop'], loop=True)
68             animation.set_timer(SFX['menu_laser_loop'].get_length() * 1000 / 2,
lambda: audio.play_sfx(SFX['menu_laser_loop'], loop=True) if self._fire_laser
else ...) # OVERLAP TWO LOOPS TO HIDE TRANSITION
69
70         elif event.type == pygame.MOUSEBUTTONDOWN:
71             self._fire_laser = False
72
73         window.clear_effect(ShaderType.RAYS)
74         animation.set_timer(300, lambda: window.clear_effect(ShaderType.SHAKE)
)
75
76         audio.stop_sfx(1000)
77
78         widget_event = self._widget_group.process_event(event)
79
80         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):

```

```

103         sphinx_surface = scale_and_cache(GRAPHICS['sphinx_0_b'], self.sphinx_size)
104         sphinx_surface = pygame.transform.rotate(sphinx_surface, self.
sphinx_rotation)
105         sphinx_rect = pygame.Rect(0, 0, *self.sphinx_size)
106         sphinx_rect.center = self.sphinx_center
107
108         window.screen.blit(sphinx_surface, sphinx_rect)
109
110     def update_masks(self):
111         self.draw()
112
113         widget_mask = window.screen.copy()
114         laser_mask = pygame.mask.from_surface(widget_mask)
115         laser_mask = laser_mask.to_surface(setcolor=(255, 0, 0, 255), unsetcolor
=(0, 0, 0, 255))
116         pygame.draw.rect(laser_mask, (0, 0, 0), (window.screen.width - self.
sphinx_size[0], window.screen.height - self.sphinx_size[1], *self.sphinx_size)
)
117         pygame.draw.rect(widget_mask, (0, 0, 0, 255), (window.screen.width - 50,
0, 50, 50))
118
119         self._bloom_mask = widget_mask
120         self._laser_mask = laser_mask
121
122     def draw(self):
123         self._widget_group.draw()
124         self.draw_sphinx()
125
126         if self._fire_laser:
127             window.set_apply_arguments(ShaderType.RAYS, occlusion=self._laser_mask
, softShadow=0.1)
128
129             window.set_apply_arguments(ShaderType.BLOOM, highlight_surface=self.
_bloom_mask, surface_intensity=0.3, brightness_intensity=0.6)
130
131     def update(self, **kwargs):
132         random_offset = lambda: randint(-5, 5) / 40
133         if self._fire_laser:
134             window.clear_effect(ShaderType.RAYS)
135             window.set_effect(ShaderType.RAYS, lights=[[
136                 (self.sphinx_center[0] / window.size[0], self.sphinx_center[1] /
window.size[1]),
137                 2.2,
138                 (190, 190, 255),
139                 0.99,
140                 (self.sphinx_rotation - 2 + random_offset(), self.sphinx_rotation
+ 2 + random_offset())
141                 ]])
142
143             window.set_effect(ShaderType.SHAKE)
144             window.set_apply_arguments(ShaderType.SHAKE, intensity=1)
145             pygame.mouse.set_pos(pygame.mouse.get_pos()[0] + random_offset(),
pygame.mouse.get_pos()[1] + random_offset())
146
147         super().update(**kwargs)
148
149     def handle_resize(self):
150         super().handle_resize()
151         self.update_masks()

```

1.20.2 widget_dict.py

```

1 from data.components.custom_event import CustomEvent
2 from data.constants import MenuEventType
3 from data.managers.theme import theme
4 from data.assets import GRAPHICS
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 }

```

1.21 data\states\review

1.21.1 review.py

See Section ??.

1.21.2 widget_dict.py

```

1 from data.widgets import *
2 from data.components.custom_event import CustomEvent
3 from data.constants import ReviewEventType, Colour
4 from data.assets import GRAPHICS
5
6 MOVE_LIST_WIDTH = 0.2
7
8 right_container = Rectangle(
9     relative_position=(0.05, 0),
10    relative_size=(0.2, 0.7),
11    anchor_y='center',
12    anchor_x='right'
13 )

```

```

14
15 info_container = Rectangle(
16     parent=right_container,
17     relative_position=(0, 0.5),
18     relative_size=(1, 0.5),
19     visible=True
20 )
21
22 arrow_container = Rectangle(
23     relative_position=(0, 0.05),
24     relative_size=(0.4, 0.1),
25     anchor_x='center',
26     anchor_y='bottom'
27 )
28
29 move_list = MoveList(
30     parent=right_container,
31     relative_position=(0, 0),
32     relative_width=1,
33     minimum_height=300,
34     move_list=[]
35 )
36
37 top_right_container = Rectangle(
38     relative_position=(0, 0),
39     relative_size=(0.15, 0.075),
40     fixed_position=(5, 5),
41     anchor_x='right',
42     scale_mode='height'
43 )
44
45 REVIEW_WIDGETS = {
46     'help':
47         Icon(
48             relative_position=(0, 0),
49             relative_size=(1.02, 1.02),
50             icon=GRAPHICS['review_help'],
51             anchor_x='center',
52             anchor_y='center',
53             border_width=0,
54             fill_colour=(0, 0, 0, 0)
55         ),
56     'default': [
57         arrow_container,
58         right_container,
59         info_container,
60         top_right_container,
61         ReactiveIconButton(
62             parent=top_right_container,
63             relative_position=(0, 0),
64             relative_size=(1, 1),
65             anchor_x='right',
66             scale_mode='height',
67             base_icon=GRAPHICS['home_base'],
68             hover_icon=GRAPHICS['home_hover'],
69             press_icon=GRAPHICS['home_press'],
70             event=CustomEvent(ReviewEventType.MENU_CLICK)
71         ),
72         ReactiveIconButton(
73             parent=top_right_container,
74             relative_position=(0, 0),
75             relative_size=(1, 1),

```

```

76         scale_mode='height',
77         base_icon=GRAPHICS['help_base'],
78         hover_icon=GRAPHICS['help_hover'],
79         press_icon=GRAPHICS['help_press'],
80         event=CustomEvent(ReviewEventType.HELP_CLICK)
81     ),
82     ReactiveIconButton(
83         parent=arrow_container,
84         relative_position=(0, 0),
85         relative_size=(1, 1),
86         scale_mode='height',
87         base_icon=GRAPHICS['left_arrow_filled_base'],
88         hover_icon=GRAPHICS['left_arrow_filled_hover'],
89         press_icon=GRAPHICS['left_arrow_filled_press'],
90         event=CustomEvent(ReviewEventType.PREVIOUS_CLICK)
91     ),
92     ReactiveIconButton(
93         parent=arrow_container,
94         relative_position=(0, 0),
95         relative_size=(1, 1),
96         scale_mode='height',
97         anchor_x='right',
98         base_icon=GRAPHICS['right_arrow_filled_base'],
99         hover_icon=GRAPHICS['right_arrow_filled_hover'],
100        press_icon=GRAPHICS['right_arrow_filled_press'],
101        event=CustomEvent(ReviewEventType.NEXT_CLICK)
102    ),
103 ],
104 'move_list':
105     move_list,
106 'scroll_area':
107     ScrollArea(
108         parent=right_container,
109         relative_position=(0, 0),
110         relative_size=(1, 0.5),
111         vertical=True,
112         widget=move_list
113     ),
114 'chessboard':
115     Chessboard(
116         relative_position=(0, 0),
117         relative_width=0.4,
118         scale_mode='width',
119         anchor_x='center',
120         anchor_y='center'
121     ),
122 'move_number_text':
123     Text(
124         parent=info_container,
125         relative_position=(0, 0),
126         relative_size=(1, 0.3),
127         anchor_y='bottom',
128         text='MOVE NO:',
129         fit_vertical=False,
130         margin=10,
131         border_width=0,
132         fill_colour=(0, 0, 0, 0),
133     ),
134 'move_colour_text':
135     Text(
136         parent=info_container,
137         relative_size=(1, 0.3),

```

```

138         relative_position=(0, 0),
139         anchor_y='center',
140         text='TO MOVE',
141         fit_vertical=False,
142         margin=10,
143         border_width=0,
144         fill_colour=(0, 0, 0, 0),
145     ),
146     'winner_text':
147     Text(
148         parent=info_container,
149         relative_size=(1, 0.3),
150         relative_position=(0, 0),
151         text='WINNER:',
152         fit_vertical=False,
153         margin=10,
154         border_width=0,
155         fill_colour=(0, 0, 0, 0),
156     ),
157     'blue_timer':
158     Timer(
159         relative_position=(0.05, 0.05),
160         anchor_y='center',
161         relative_size=(0.1, 0.1),
162         active_colour=Colour.BLUE,
163     ),
164     'red_timer':
165     Timer(
166         relative_position=(0.05, -0.05),
167         anchor_y='center',
168         relative_size=(0.1, 0.1),
169         active_colour=Colour.RED
170     ),
171     'timer_disabled_text':
172     Text(
173         relative_size=(0.2, 0.1),
174         relative_position=(0.05, 0),
175         anchor_y='center',
176         fit_vertical=False,
177         text='TIMER DISABLED',
178     ),
179     'blue_piece_display':
180     PieceDisplay(
181         relative_position=(0.05, 0.05),
182         relative_size=(0.2, 0.1),
183         anchor_y='bottom',
184         active_colour=Colour.BLUE
185     ),
186     'red_piece_display':
187     PieceDisplay(
188         relative_position=(0.05, 0.05),
189         relative_size=(0.2, 0.1),
190         active_colour=Colour.RED
191     ),
192 }

```

1.22 data\states\settings

1.22.1 settings.py

```

1 import pygame

```

```

2 from random import randint
3 from data.utils.data_helpers import get_default_settings, get_user_settings,
  update_user_settings
4 from data.constants import SettingsEventType, WidgetState, ShaderType, SHADER_MAP
5 from data.states.settings.widget_dict import SETTINGS_WIDGETS
6 from data.managers.logs import initialise_logger
7 from data.managers.window import window
8 from data.managers.audio import audio
9 from data.widgets import ColourPicker
10 from data.control import _State
11 from data.assets import MUSIC
12
13 logger = initialise_logger(__name__)
14
15 class Settings(_State):
16     def __init__(self):
17         super().__init__()
18
19         self._colour_picker = None
20         self._settings = None
21
22     def cleanup(self):
23         super().cleanup()
24
25         update_user_settings(self._settings)
26
27         return None
28
29     def startup(self, persist=None):
30         super().startup(SETTINGS_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
31
32         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
33         self._settings = get_user_settings()
34         self.reload_settings()
35
36         self.draw()
37
38     def create_colour_picker(self, mouse_pos, button_type):
39         if button_type == SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK:
40             selected_colour = self._settings['primaryBoardColour']
41             event_type = SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK
42         else:
43             selected_colour = self._settings['secondaryBoardColour']
44             event_type = SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK
45
46         self._colour_picker = ColourPicker(
47             relative_position=(mouse_pos[0] / window.size[0], mouse_pos[1] /
window.size[1]),
48             relative_width=0.15,
49             selected_colour=selected_colour,
50             event_type=event_type
51         )
52         self._widget_group.add(self._colour_picker)
53
54     def remove_colour_picker(self):
55         self._colour_picker.kill()
56
57     def reload_display_mode(self):
58         relative_mouse_pos = (pygame.mouse.get_pos()[0] / window.size[0], pygame.
mouse.get_pos()[1] / window.size[1])
59

```

```

60         if self._settings['displayMode'] == 'fullscreen':
61             window.set_fullscreen(desktop=True)
62             window.handle_resize()
63
64         elif self._settings['displayMode'] == 'windowed':
65             window.set_windowed()
66             window.handle_resize()
67             window.restore()
68
69         self._widget_group.handle_resize(window.size)
70
71         new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
72                          relative_mouse_pos[1] * window.size[1])
73         pygame.mouse.set_pos(new_mouse_pos)
74
75     def reload_shaders(self):
76         window.clear_all_effects()
77
78         for shader_type in SHADER_MAP[self._settings['shader']]:
79             window.set_effect(shader_type)
80
81     def reload_settings(self):
82         SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self._settings['primaryBoardColour'])
83         SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self._settings['secondaryBoardColour'])
84         SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.BASE)
85         SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.BASE)
86         SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['musicVolume'])
87         SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume'])
88         SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings['displayMode'])
89         SETTINGS_WIDGETS['shader_carousel'].set_to_key(self._settings['shader'])
90         SETTINGS_WIDGETS['particles_switch'].set_toggle_state(self._settings['particles'])
91         SETTINGS_WIDGETS['opengl_switch'].set_toggle_state(self._settings['opengl'])
92
93     self.reload_shaders()
94     self.reload_display_mode()
95
96     def get_event(self, event):
97         widget_event = self._widget_group.process_event(event)
98
99         if widget_event is None:
100             if event.type == pygame.MOUSEBUTTONDOWN and self._colour_picker:
101                 self.remove_colour_picker()
102             return
103
104         match widget_event.type:
105             case SettingsEventType.VOLUME_SLIDER_SLIDE:
106                 return
107
108             case SettingsEventType.VOLUME_SLIDER_CLICK:
109                 if widget_event.volume_type == 'music':
110                     audio.set_music_volume(widget_event.volume)
111                     self._settings['musicVolume'] = widget_event.volume
112                 elif widget_event.volume_type == 'sfx':

```

```

112         audio.set_sfx_volume(widget_event.volume)
113         self._settings['sfxVolume'] = widget_event.volume
114
115     case SettingsEventType.DROPDOWN_CLICK:
116         selected_word = SETTINGS_WIDGETS['display_mode_dropdown'].
get_selected_word()
117
118         if selected_word is None or selected_word == self._settings['
displayMode']:
119             return
120
121         self._settings['displayMode'] = selected_word
122
123         self.reload_display_mode()
124
125     case SettingsEventType.MENU_CLICK:
126         self.next = 'menu'
127         self.done = True
128
129     case SettingsEventType.RESET_DEFAULT:
130         self._settings = get_default_settings()
131         self.reload_settings()
132
133     case SettingsEventType.RESET_USER:
134         self._settings = get_user_settings()
135         self.reload_settings()
136
137     case SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK | SettingsEventType
.SECONDARY_COLOUR_BUTTON_CLICK:
138         if self._colour_picker:
139             self.remove_colour_picker()
140
141         self.create_colour_picker(event.pos, widget_event.type)
142
143     case SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK | SettingsEventType
.SECONDARY_COLOUR_PICKER_CLICK:
144         if widget_event.colour:
145             r, g, b = widget_event.colour.rgb
146             hex_colour = f'0x{hex(r)[2:].zfill(2)}{hex(g)[2:].zfill(2)}{
hex(b)[2:].zfill(2)}'
147
148             if widget_event.type == SettingsEventType.
PRIMARY_COLOUR_PICKER_CLICK:
149                 SETTINGS_WIDGETS['primary_colour_button'].
initialise_new_colours(widget_event.colour)
150                 SETTINGS_WIDGETS['primary_colour_button'].set_state_colour
(WidgetState.BASE)
151                 self._settings['primaryBoardColour'] = hex_colour
152             elif widget_event.type == SettingsEventType.
SECONDARY_COLOUR_PICKER_CLICK:
153                 SETTINGS_WIDGETS['secondary_colour_button'].
initialise_new_colours(widget_event.colour)
154                 SETTINGS_WIDGETS['secondary_colour_button'].
set_state_colour(WidgetState.BASE)
155                 self._settings['secondaryBoardColour'] = hex_colour
156
157     case SettingsEventType.SHADER_PICKER_CLICK:
158         self._settings['shader'] = widget_event.data
159         self.reload_shaders()
160
161     case SettingsEventType.OPENGL_CLICK:
162         self._settings['opengl'] = widget_event.toggled

```



```

163         self.reload_shaders()
164
165         case SettingsEventType.PARTICLES_CLICK:
166             self._settings['particles'] = widget_event.toggled
167
168     def draw(self):
169         self._widget_group.draw()

```

1.22.2 widget_dict.py

```

1 from data.widgets import *
2 from data.components.custom_event import CustomEvent
3 from data.constants import SettingsEventType, SHADER_MAP
4 from data.utils.data_helpers import get_user_settings
5 from data.assets import GRAPHICS, DEFAULT_FONT
6 from data.managers.theme import theme
7 from data.utils.font_helpers import text_width_to_font_size
8 from data.managers.window import window
9
10 user_settings = get_user_settings()
11 # font_size = text_width_to_font_size('Shaders (OPENGL GPU REQUIRED)',
12     DEFAULT_FONT, 0.4 * window.screen.width)
13 FONT_SIZE = 21
14
15 carousel_widgets = {
16     key: Text(
17         relative_position=(0, 0),
18         relative_size=(0.25, 0.04),
19         margin=0,
20         text=key.replace('_', ' ').upper(),
21         fit_vertical=True,
22         border_width=0,
23         fill_colour=(0, 0, 0, 0),
24     ) for key in SHADER_MAP.keys()
25 }
26
27 reset_container = Rectangle(
28     relative_size=(0.2, 0.2),
29     relative_position=(0, 0),
30     fixed_position=(5, 5),
31     anchor_x='right',
32     anchor_y='bottom',
33 )
34
35 SETTINGS_WIDGETS = {
36     'default': [
37         reset_container,
38         ReactiveIconButton(
39             relative_position=(0, 0),
40             relative_size=(0.075, 0.075),
41             anchor_x='right',
42             scale_mode='height',
43             base_icon=GRAPHICS['home_base'],
44             hover_icon=GRAPHICS['home_hover'],
45             press_icon=GRAPHICS['home_press'],
46             fixed_position=(5, 5),
47             event=CustomEvent(SettingsEventType.MENU_CLICK)
48         ),
49         Text(
50             relative_position=(0.01, 0.1),
51             text='Display mode',
52             relative_size=(0.4, 0.04),

```

```

52         center=False,
53         border_width=0,
54         margin=0,
55         font_size=21,
56         fill_colour=(0, 0, 0, 0)
57     ),
58     Text(
59         relative_position=(0.01, 0.2),
60         text='Music',
61         relative_size=(0.4, 0.04),
62         center=False,
63         border_width=0,
64         margin=0,
65         font_size=21,
66         fill_colour=(0, 0, 0, 0)
67     ),
68     Text(
69         relative_position=(0.01, 0.3),
70         text='SFX',
71         relative_size=(0.4, 0.04),
72         center=False,
73         border_width=0,
74         margin=0,
75         font_size=21,
76         fill_colour=(0, 0, 0, 0)
77     ),
78     Text(
79         relative_position=(0.01, 0.4),
80         text='Primary board colour',
81         relative_size=(0.4, 0.04),
82         center=False,
83         border_width=0,
84         margin=0,
85         font_size=21,
86         fill_colour=(0, 0, 0, 0)
87     ),
88     Text(
89         relative_position=(0.01, 0.5),
90         text='Secondary board colour',
91         relative_size=(0.4, 0.04),
92         center=False,
93         border_width=0,
94         margin=0,
95         font_size=21,
96         fill_colour=(0, 0, 0, 0)
97     ),
98     Text(
99         relative_position=(0.01, 0.6),
100        text='Particles',
101        relative_size=(0.4, 0.04),
102        center=False,
103        border_width=0,
104        margin=0,
105        font_size=21,
106        fill_colour=(0, 0, 0, 0)
107    ),
108    Text(
109        relative_position=(0.01, 0.7),
110        text='Shaders (OPENGL GPU REQUIRED)',
111        relative_size=(0.4, 0.04),
112        center=False,
113        border_width=0,

```

```

114         margin=0,
115         font_size=21,
116         fill_colour=(0, 0, 0, 0)
117     ),
118     Text(
119         relative_position=(0.01, 0.8),
120         text='Super Secret Settings',
121         relative_size=(0.4, 0.04),
122         center=False,
123         border_width=0,
124         margin=0,
125         font_size=21,
126         fill_colour=(0, 0, 0, 0)
127     ),
128     TextButton(
129         parent=reset_container,
130         relative_position=(0, 0),
131         relative_size=(1, 0.5),
132         fit_vertical=False,
133         margin=10,
134         text='DISCARD CHANGES',
135         text_colour=theme['textSecondary'],
136         event=CustomEvent(SettingsEventType.RESET_USER)
137     ),
138     TextButton(
139         parent=reset_container,
140         relative_position=(0, 0.5),
141         relative_size=(1, 0.5),
142         fit_vertical=False,
143         margin=10,
144         text='RESET TO DEFAULT',
145         text_colour=theme['textSecondary'],
146         event=CustomEvent(SettingsEventType.RESET_DEFAULT)
147     )
148 ],
149 'display_mode_dropdown':
150 Dropdown(
151     relative_position=(0.4, 0.1),
152     relative_width=0.2,
153     word_list=['fullscreen', 'windowed'],
154     fill_colour=(255, 100, 100),
155     event=CustomEvent(SettingsEventType.DROPDOWN_CLICK)
156 ),
157 'primary_colour_button':
158 ColourButton(
159     relative_position=(0.4, 0.4),
160     relative_size=(0.08, 0.05),
161     fill_colour=user_settings['primaryBoardColour'],
162     border_width=5,
163     event=CustomEvent(SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK)
164 ),
165 'secondary_colour_button':
166 ColourButton(
167     relative_position=(0.4, 0.5),
168     relative_size=(0.08, 0.05),
169     fill_colour=user_settings['secondaryBoardColour'],
170     border_width=5,
171     event=CustomEvent(SettingsEventType.SECONDARY_COLOUR_BUTTON_CLICK)
172 ),
173 'music_volume_slider':
174 VolumeSlider(
175     relative_position=(0.4, 0.2),

```

```

176         relative_length=(0.5),
177         default_volume=user_settings['musicVolume'],
178         border_width=5,
179         volume_type='music'
180     ),
181     'sfx_volume_slider':
182     VolumeSlider(
183         relative_position=(0.4, 0.3),
184         relative_length=(0.5),
185         default_volume=user_settings['sfxVolume'],
186         border_width=5,
187         volume_type='sfx'
188     ),
189     'shader_carousel':
190     Carousel(
191         relative_position = (0.4, 0.8),
192         margin=5,
193         border_width=0,
194         fill_colour=(0, 0, 0, 0),
195         widgets_dict=carousel_widgets,
196         event=CustomEvent(SettingsEventType.SHADER_PICKER_CLICK),
197     ),
198     'particles_switch':
199     Switch(
200         relative_position=(0.4, 0.6),
201         relative_height=0.04,
202         event=CustomEvent(SettingsEventType.PARTICLES_CLICK)
203     ),
204     'opengl_switch':
205     Switch(
206         relative_position=(0.4, 0.7),
207         relative_height=0.04,
208         event=CustomEvent(SettingsEventType.OPENGL_CLICK)
209     ),
210 }

```

1.23 data\utils

1.23.1 asset_helpers.py

See Section ??.

1.23.2 bitboard_helpers.py

```

1  from data.constants import Rank, File, EMPTY_BB
2  from data.managers.logs import initialise_logger
3
4  logger = initialise_logger(__name__)
5
6  def print_bitboard(bitboard):
7      if (bitboard >= (2 ** 80)):
8          raise ValueError('Invalid bitboard: too many bits')
9
10     characters = ''
11     for rank in reversed(Rank):
12
13         for file in File:
14             mask = 1 << (rank * 10 + file)
15             if (bitboard & mask) != 0:
16                 characters += '1 '

```

```

17         else:
18             characters += ' '
19
20         characters += '\n\n'
21
22     logger.info('\n' + characters + '\n')
23
24 def is_occupied(bitboard, target_bitboard):
25     return (target_bitboard & bitboard) != EMPTY_BB
26
27 def clear_square(bitboard, target_bitboard):
28     return (~target_bitboard & bitboard)
29
30 def set_square(bitboard, target_bitboard):
31     return (target_bitboard | bitboard)
32
33 def index_to_bitboard(index):
34     return (1 << index)
35
36 def coords_to_bitboard(coords):
37     index = coords[1] * 10 + coords[0]
38     return index_to_bitboard(index)
39
40 def bitboard_to_notation(bitboard):
41     index = bitboard_to_index(bitboard)
42     x = index // 10
43     y = index % 10
44
45     return chr(y + 97) + str(x + 1)
46
47 def notation_to_bitboard(notation):
48     index = (int(notation[1]) - 1) * 10 + int(ord(notation[0])) - 97
49
50     return index_to_bitboard(index)
51
52 def bitboard_to_index(bitboard):
53     return bitboard.bit_length() - 1
54
55 def bitboard_to_coords(bitboard):
56     list_position = bitboard_to_index(bitboard)
57     x = list_position % 10
58     y = list_position // 10
59
60     return x, y
61
62 def bitboard_to_coords_list(bitboard):
63     list_positions = []
64
65     for square in occupied_squares(bitboard):
66         list_positions.append(bitboard_to_coords(square))
67
68     return list_positions
69
70 def occupied_squares(bitboard):
71     while bitboard:
72         lsb_square = bitboard & -bitboard
73         bitboard = bitboard ^ lsb_square
74
75         yield lsb_square
76
77 def pop_count(bitboard):
78     count = 0

```

```

79     while bitboard:
80         count += 1
81         lsb_square = bitboard & -bitboard
82         bitboard = bitboard ^ lsb_square
83
84     return count
85
86 # def pop_count(bitboard):
87 #     count = 0
88 #     while bitboard:
89 #         count += 1
90 #         bitboard &= bitboard - 1
91
92 #     return count
93
94 def loop_all_squares():
95     for i in range(80):
96         yield 1 << i
97
98 #Solar
99 def get_LSB_value(bitboard: int):
100     return bitboard & -bitboard
101
102 def pop_count_2(bitboard):
103     count = 0
104     while bitboard > 0:
105         lsb_value = get_LSB_value(bitboard)
106         count += 1
107         bitboard ^= lsb_value
108
109     return count

```

1.23.3 board_helpers.py

```

1 import pygame
2 from data.utils.data_helpers import get_user_settings
3 from data.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)

```

```

27         text_size = square_size / 3
28         font.render_to(board_surface, text_position, str(chr(x + 1 + 96)),
fgcolor=(10, 10, 10, 175), size=text_size)
29         if x == 0:
30             text_position = (square_x + square_size * 0.1, square_y + square_size
* 0.1)
31             text_size = square_size / 3
32             font.render_to(board_surface, text_position, str(7-y + 1), fgcolor
=(10, 10, 10, 175), size=text_size)
33
34     return board_surface
35
36 def create_square_overlay(square_size, colour):
37     overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
38     overlay.fill(colour)
39
40     return overlay
41
42 def create_circle_overlay(square_size, colour):
43     overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
44     pygame.draw.circle(overlay, colour, (square_size / 2, square_size / 2),
square_size / 4)
45
46     return overlay
47
48 def coords_to_screen_pos(coords, board_position, square_size):
49     x = board_position[0] + (coords[0] * square_size)
50     y = board_position[1] + ((7 - coords[1]) * square_size)
51
52     return (x, y)
53
54 def screen_pos_to_coords(mouse_position, board_position, board_size):
55     if (board_position[0] <= mouse_position[0] <= board_position[0] + board_size
[0]) and (board_position[1] <= mouse_position[1] <= board_position[1] +
board_size[1]):
56         x = (mouse_position[0] - board_position[0]) // (board_size[0] / 10)
57         y = (board_size[1] - (mouse_position[1] - board_position[1])) // (
board_size[0] / 10)
58         return (int(x), int(y))
59
60     return None

```

1.23.4 browser_helpers.py

```

1 from data.constants 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

```

1.23.5 database_helpers.py

See Section ??.

1.23.6 data_helpers.py

See Section ??.

1.23.7 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        test_size += 1
56
57 def get_font_height(font, font_size):
58     glyph_metrics = font.get_metrics('j', size=font_size)
59     descender = font.get_sized_descender(font_size)

```



```
58     return abs(glyph_metrics[0][3] - glyph_metrics[0][2]) - descender
```

1.23.8 input_helpers.py

```
1 from data.constants 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())
```

1.23.9 load_helpers.py

```
1 import pygame
2 from pathlib import Path
3
4 import pygame.freetype
5 from data.utils.asset_helpers import gif_to_frames, pil_image_to_surface
6
7 def convert_gfx_alpha(image, colorkey=(0, 0, 0)):
8     # if image.get_alpha():
9         return image.convert_alpha()
10    # else:
11    #     image = image.convert_alpha()
12    #     image.set_colorkey(colorkey)
13
14    #     return image
15
16 def load_gfx(path, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
17     file_path = Path(path)
18     name, extension = file_path.stem, file_path.suffix
19
20     if extension.lower() in accept:
21         if extension.lower() == '.gif':
22             frames_list = []
23
24             for frame in gif_to_frames(path):
25                 image_surface = pil_image_to_surface(frame)
```

```

26         frames_list.append(image_surface)
27
28     return frames_list
29
30     if extension.lower() == '.svg':
31         low_quality_image = pygame.image.load_sized_svg(path, (200, 200))
32         image = pygame.image.load(path)
33         image = convert_gfx_alpha(image, colorkey)
34
35         return [image, low_quality_image]
36
37     else:
38         image = pygame.image.load(path)
39         return convert_gfx_alpha(image, colorkey)
40
41 def load_all_gfx(directory, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".
gif")):
42     graphics = {}
43
44     for file in Path(directory).rglob('*'):
45         name, extension = file.stem, file.suffix
46         path = Path(directory / file)
47
48         if extension.lower() in accept and 'old' not in name:
49             if name == 'piece_spritesheet':
50                 data = load_spritesheet(
51                     path,
52                     (16, 16),
53                     ['pyramid_1', 'scarab_1', 'anubis_1', 'pharoah_1', 'sphinx_1',
'pyramid_0', 'scarab_0', 'anubis_0', 'pharoah_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
67     return graphics
68
69 def load_spritesheet(path, sprite_size, col_names, row_names):
70     spritesheet = load_gfx(path)
71     col_count = int(spritesheet.width / sprite_size[0])
72     row_count = int(spritesheet.height / sprite_size[1])
73
74     sprite_dict = {}
75
76     for column in range(col_count):
77         for row in range(row_count):
78             surface = pygame.Surface(sprite_size, pygame.SRCALPHA)
79             name = col_names[column] + row_names[row]
80
81             surface.blit(spritesheet, (0, 0), (column * sprite_size[0], row *
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     for file in Path(directory).rglob('*'):
122         name, extension = file.stem, file.suffix
123         path = Path(directory / file)
124
125         if extension.lower() in accept:
126             music_paths[name] = path
127
128     return music_paths

```

1.23.10 widget_helpers.py

See Section ??.

1.24 data\widgets

1.24.1 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

```

1.24.2 board_thumbnail_button.py

```

1 import pygame
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.board_thumbnail import BoardThumbnail
4 from data.constants import WidgetState
5 from data.components.custom_event import CustomEvent
6
7 class BoardThumbnailButton(_Pressable, BoardThumbnail):
8     def __init__(self, event, **kwargs):
9         _Pressable.__init__(
10             self,

```

```

11         event=CustomEvent(**vars(event), fen_string=kwargs.get('fen_string')),
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     BoardThumbnail.__init__(self, **kwargs)
17
18     self.initialise_new_colours(self._fill_colour)
19     self.set_state_colour(WidgetState.BASE)

```

1.24.3 browser_item.py

```

1  import pygame
2  from data.utils.font_helpers import text_width_to_font_size
3  from data.utils.browser_helpers import get_winner_string
4  from data.widgets.board_thumbnail import BoardThumbnail
5  from data.utils.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
26         self.set_image()
27         self.set_geometry()
28
29     def get_text_to_render(self):
30         depth_to_text = {
31             2: 'EASY',
32             3: 'MEDIUM',
33             4: 'HARD'
34         }
35
36         format_moves = lambda no_of_moves: int(no_of_moves / 2) if (no_of_moves /
37 2 % 1 == 0) else round(no_of_moves / 2, 1)
38
39         if self._game['cpu_enabled'] == 1:
40             depth_text = depth_to_text[self._game['cpu_depth']]
41             cpu_text = f'PVC ({depth_text})'
42         else:
43             cpu_text = 'PVP'
44
45         return [
46             cpu_text,
47             self._game['created_dt'].strftime('%Y-%m-%d %H:%M:%S'),
48             f'WINNER: {get_winner_string(self._game['winner'])}',
49             f'NO. MOVES: {format_moves(self._game['number_of_ply'])}'
50         ]

```

```

48
49     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

```

1.24.4 browser_strip.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.browser_item import BrowserItem
4 from data.constants import BrowserEventType
5 from data.components.custom_event import CustomEvent
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

```

```

41         self._selected_index = None
42
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)

```

1.24.5 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.constants import Miscellaneous
7 from data.assets import GRAPHICS, SFX
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)

```



```

59     @property
60     def size(self):
61         return ((self.arrow_size[0] + self.margin) * 2 + self.max_widget_size[0],
62                 self.max_widget_size[1])
63
64     @property
65     def left_arrow_position(self):
66         return (0, (self.size[1] - self.arrow_size[1]) / 2)
67
68     @property
69     def right_arrow_position(self):
70         return (self.size[0] - self.arrow_size[0], (self.size[1] - self.arrow_size
71 [1]) / 2)
72
73     def set_image(self):
74         self.image = pygame.transform.scale(self._empty_surface, self.size)
75         self.image.fill(self._fill_colour)
76
77         if self.border_width:
78             pygame.draw.rect(self.image, self._border_colour, (0, 0, *self.size),
79                             width=int(self.border_width), border_radius=int(self.border_radius))
80
81         self._left_arrow.set_image()
82         self.image.blit(self._left_arrow.image, self.left_arrow_position)
83
84         self.current_item.set_image()
85         self.image.blit(self.current_item.image, ((self.size[0] - self.
86 current_item.rect.size[0]) / 2, (self.size[1] - self.current_item.rect.size
87 [1]) / 2))
88
89         self._right_arrow.set_image()
90         self.image.blit(self._right_arrow.image, self.right_arrow_position)
91
92     def set_geometry(self):
93         super().set_geometry()
94
95         self.current_item.set_geometry()
96         self._left_arrow.set_geometry()
97         self._right_arrow.set_geometry()
98
99         self.current_item.rect.center = self.rect.center
100         self._left_arrow.rect.topleft = (self.position[0] + self.
101 left_arrow_position[0], self.position[1] + self.left_arrow_position[1])
102         self._right_arrow.rect.topleft = (self.position[0] + self.
103 right_arrow_position[0], self.position[1] + self.right_arrow_position[1])
104
105     def set_surface_size(self, new_surface_size):
106         super().set_surface_size(new_surface_size)
107         self._left_arrow.set_surface_size(new_surface_size)
108         self._right_arrow.set_surface_size(new_surface_size)
109
110         for item in self._items_dict.values():
111             item.set_surface_size(new_surface_size)
112
113     def process_event(self, event):
114         self.current_item.process_event(event)
115         left_arrow_event = self._left_arrow.process_event(event)
116         right_arrow_event = self._right_arrow.process_event(event)
117
118         if left_arrow_event:
119             self.set_previous_item()
120             self.current_item.set_surface_size(self._raw_surface_size)

```

```

114
115         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.
MOUSEMOTION]:
126         self.set_image()
127         self.set_geometry()

```

1.24.6 chessboard.py

```

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

```

```

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

```

1.24.7 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.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))

```

1.24.8 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

```

1.24.9 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
48               [1])
49
50     def set_image(self):
51         self.image = pygame.Surface(self.size)
52         self.image.fill(self._fill_colour)
53
54         self._square.set_image()
55         self._square.set_geometry()
56         self.image.blit(self._square.image, self.global_to_relative_pos(self.
57                               _square.position))
58
59         self._slider.set_image()
```

```

57         self._slider.set_geometry()
58         self.image.blit(self._slider.image, self.global_to_relative_pos(self._slider.position))
59
60         self._display.set_image()
61         self._display.set_geometry()
62         self.image.blit(self._display.image, self.global_to_relative_pos(self._display.position))
63
64         pygame.draw.rect(self.image, self._border_colour, (0, 0, self.size[0], 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.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONDOWN, pygame.MOUSEMOTION] and self.rect.collidepoint(event.pos):
88             return CustomEvent(self._event_type, colour=square_colour)

```

1.24.10 colour_slider.py

See Section ??.

1.24.11 colour_square.py

```

1  import pygame
2  from data.widgets.bases.widget import _Widget
3  from data.utils.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)
9
10         self._colour = None
11
12     def set_colour(self, new_colour):
13         self._colour = pygame.Color(new_colour)
14
15     def get_colour(self):
16         return self._colour

```

```

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

```

1.24.12 dropdown.py

```

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

```

```

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.FRect(0, 0, max_word_rect.size[0], (max_word_rect.
size[1] * len(self._word_list)) + (self.margin * (len(self._word_list) - 1)))
48         all_words_rect = all_words_rect.inflate(2 * self.margin, 2 * self.margin)
49         return (all_words_rect.size[0] + max_word_rect.size[1], all_words_rect.
size[1])
50
51     def get_selected_word(self):
52         return self._word_list[0].lower()
53
54     def toggle_expanded(self):
55         if self._expanded:
56             self._word_list = [self._word_list_copy[0]]
57         else:
58             self._word_list = [*self._word_list_copy]
59
60         self._expanded = not(self._expanded)
61
62     def hover_func(self):
63         mouse_position = pygame.mouse.get_pos()
64         relative_position = (mouse_position[0] - self.position[0], mouse_position
[1] - self.position[1])
65         self._hovered_index = self.calculate_hovered_index(relative_position)
66         self.set_state_colour(WidgetState.HOVER)
67
68     def set_selected_word(self, word):
69         index = self._word_list_copy.index(word.capitalize())
70         selected_word = self._word_list_copy.pop(index)
71         self._word_list_copy.insert(0, selected_word)
72
73         if self._expanded:
74             self._word_list.pop(index)
75             self._word_list.insert(0, selected_word)
76         else:
77             self._word_list = [selected_word]
78
79         self.set_image()
80
81     def up_func(self):
82         if self.get_widget_state() == WidgetState.PRESS:
83             if self._expanded and self._hovered_index is not None:
84                 self.set_selected_word(self._word_list_copy[self._hovered_index])
85
86                 self.toggle_expanded()
87
88             self._hovered_index = None
89
90         self.set_state_colour(WidgetState.BASE)
91         self.set_geometry()
92

```

```

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.Rect(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
117         if self._expanded:
118             self.image.blit(closed_arrow_surface, arrow_position)
119         else:
120             self.image.blit(open_arrow_surface, arrow_position)
121
122         for index, word in enumerate(self._word_list):
123             word_position = (self.margin, self.margin + (word_box_height + self.
124                 margin) * index)
125             self._font.render_to(self.image, word_position, word, fgcolor=self.
126                 _text_colour, size=self.font_size)
127
128             if self._hovered_index is not None:
129                 overlay_surface = pygame.Surface((self.size[0], word_box_height + 2 *
130                     self.margin), pygame.SRCALPHA)
131                 overlay_surface.fill((*self._fill_colour.rgb, 128))
132                 overlay_position = (0, (word_box_height + self.margin) * self.
133                     _hovered_index)
134                 self.image.blit(overlay_surface, overlay_position)

```

1.24.13 icon.py

```

1  import pygame
2  from data.widgets.bases.widget import _Widget
3  from data.utils.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.
65 BLEND_RGBA_MULT)
66             else:
67                 self.image.blit(scaled_icon, icon_position)
68
69             if self._box_colours is None and self.border_width:
70                 pygame.draw.rect(self.image, self._border_colour, self.image.get_rect
71 (), width=int(self.border_width), border_radius=int(self.border_radius))
72
73     def process_event(self, event):
74         pass

```

1.24.14 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.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)

```

1.24.15 move_list.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.utils.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 ))
59             else:
60                 text_position = (self.size[0] * 4 / 7, row_gap * (1 + 2 * (index
61 // 2)))
62
63             self._font.render_to(self.image, text_position, text=move, size=self.
64 font_size, fgcolor=self._text_colour)
65
66             move_number = (index // 2) + 1
67             move_number_position = (self.size[0] / 14, row_gap * (1 + 2 * (index
68 // 2)))
69
70             self._font.render_to(self.image, move_number_position, text=str(
71 move_number), size=self.font_size, fgcolor=self._text_colour)
72
73     def process_event(self, event, scrolled_pos=None):
74         pass

```

1.24.16 multiple_icon_button.py

```

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

```

1.24.17 piece_display.py

```

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

```

```

31     def reset_piece_list(self):
32         self._piece_list = []
33         self.initialise_piece_surface()
34
35     def initialise_piece_surface(self):
36         self._piece_surface = pygame.Surface((self.size[0] - 2 * self.margin, self
37 .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
44 .margin) / len(self._piece_list))
45         piece_list = []
46
47         for index, piece in enumerate(self._piece_list):
48             piece_instance = PieceSprite(piece, self._active_colour.
49 get_flipped_colour(), Rotation.UP)
50             piece_instance.set_geometry((0, 0), piece_width)
51             piece_instance.set_image()
52             piece_list.append((piece_instance.image, (piece_width * index, (self.
53 _piece_surface.height - piece_width) / 2)))
54
55         self._piece_surface.fblits(piece_list)
56
57         self.set_image()
58
59     def set_image(self):
60         self.image = create_text_box(self.size, self.border_width, self.
61 _box_colours)
62
63         resized_piece_surface = scale_and_cache(self._piece_surface, (self.size[0]
64 - 2 * self.margin, self.size[1] - 2 * self.margin))
65         self.image.blit(resized_piece_surface, (self.margin, self.margin))
66
67     def process_event(self, event):
68         pass

```

1.24.18 reactive_button.py

See Section ??.

1.24.19 reactive_icon_button.py

See Section ??.

1.24.20 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
25     def process_event(self, event):
26         pass

```

1.24.21 scrollbar.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.constants import WidgetState, Miscellaneous
5
6 # self.set_state_colour(WidgetState.HOVER)
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

```

1.24.22 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)
85         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.
scrollbar_size[0] + 0.001)
86         else:
87             max_scoll_width = widget_width - self.size[0]
88             current_scroll_width = self._scroll_percentage * max_scoll_width
89             self._scroll_percentage = (current_scroll_width + offset) /
max_scoll_width
90
91         return min(1, max(0, self._scroll_percentage))
92
93     def calculate_widget_rect(self):
94         widget_position = self.calculate_widget_position()
95         return pygame.Rect(widget_position[0] - self.position[0], self.position
[1] + widget_position[1], self.size[0], self.size[1])
96
97     def calculate_widget_position(self):
98         if self._vertical:
99             return (0, -self._scroll_percentage * (self._widget.rect.height - self
.size[1]))
100         else:
101             return (-self._scroll_percentage * (self._widget.rect.width - self.
size[0]), 0)
102
103     def calculate_relative_scrollbar_position(self):
104         if self._vertical:
105             vertical_offset = (self.size[1] - self.scrollbar_size[1]) * self.
_scroll_percentage
106             scrollbar_position = (0, vertical_offset)
107         else:
108             horizontal_offset = (self.size[0] - self.scrollbar_size[0]) * self.
_scroll_percentage
109             scrollbar_position = (horizontal_offset, 0)
110
111         return (scrollbar_position[0] / self.size[0], scrollbar_position[1] / self
.size[1])
112
113     def set_widget(self, new_widget):
114         self._widget = new_widget
115         self.set_image()
116         self.set_geometry()
117
118     def set_image(self):
119         self.image = pygame.transform.scale(self._empty_surface, self.size)
120         self.image.fill(theme['fillPrimary'])
121
122         self._widget.set_image()
123         self.image.blit(self._widget.image, self.calculate_widget_position())
124
125         self._scrollbar.set_relative_position(self.
calculate_relative_scrollbar_position()) # WRONG USING RELATIVE
126         self._scrollbar.set_relative_size((self.scrollbar_size[0] / self.size[1],
self.scrollbar_size[1] / self.size[1]))
127         self._scrollbar.set_image()
128         relative_scrollbar_position = (self._scrollbar.rect.left - self.position
[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(-
159                                     self.scroll_factor)
160                             self.set_image()
161                             return
162                         elif event.button == 5:
163                             if self._scroll_percentage == 100:
164                                 return
165                             self._scroll_percentage = self.calculate_scroll_percentage(
166                                     self.scroll_factor)
167                             self.set_image()
168                             return
169
170                 widget_event = self._widget.process_event(event, scrolled_pos=self.
171                                     calculate_widget_position())
172                 if widget_event is not None:
173                     self.set_image()
174                     return widget_event

```

1.24.23 slider_thumb.py

```

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

```

```

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

```

1.24.24 switch.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.constants import WidgetState
5 from data.utils.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] /
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,
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)

```

1.24.25 text.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.constants import WidgetState
4 from data.utils.font_helpers import text_width_to_font_size,
text_height_to_font_size, height_to_font_size
5 from data.utils.widget_helpers import create_text_box
6 from data.assets import GRAPHICS

```

```

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

```

```

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

```

1.24.26 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.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], **
kwargs)
17
18         self.initialise_new_colours(self._fill_colour)
19         self.set_state_colour(WidgetState.BASE)

```

1.24.27 text_input.py

See Section ??.

1.24.28 timer.py

```

1 import pygame
2 from data.constants import WidgetState, Colour, BLUE_BUTTON_COLOURS,
RED_BUTTON_COLOURS
3 from data.components.custom_event import CustomEvent
4 from data.managers.animation import animation
5 from data.widgets.text import Text
6
7 class Timer(Text):

```

```

8     def __init__(self, active_colour, event=None, start_mins=60, **kwargs):
9         box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour ==
Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]
10
11         self._current_ms = float(start_mins) * 60 * 1000
12         self._active_colour = active_colour
13         self._active = False
14         self._timer_running = False
15         self._event = event
16
17         super().__init__(text=self.format_to_text(), fit_vertical=False,
box_colours=box_colours, **kwargs)
18
19     def set_active(self, is_active):
20         if self._active == is_active:
21             return
22
23         if is_active and self._timer_running is False:
24             self._timer_running = True
25             animation.set_timer(1000, self.decrement_second)
26
27         self._active = is_active
28
29     def set_time(self, milliseconds):
30         self._current_ms = milliseconds
31         self._text = self.format_to_text()
32         self.set_image()
33         self.set_geometry()
34
35     def get_time(self):
36         return self._current_ms / (1000 * 60)
37
38     def decrement_second(self):
39         if self._active:
40             self.set_time(self._current_ms - 1000)
41
42             if self._current_ms <= 0:
43                 self._active = False
44                 self._timer_running = False
45                 self.set_time(0)
46                 pygame.event.post(pygame.event.Event(pygame.MOUSEMOTION, pos=
pygame.mouse.get_pos())) # RANDOM EVENT TO TRIGGER process_event
47             else:
48                 animation.set_timer(1000, self.decrement_second)
49         else:
50             self._timer_running = False
51
52     def format_to_text(self):
53         raw_seconds = self._current_ms / 1000
54         minutes, seconds = divmod(raw_seconds, 60)
55         return f'{str(int(minutes)).zfill(2)}:{str(int(seconds)).zfill(2)}'
56
57     def process_event(self, event):
58         if self._current_ms <= 0:
59             return CustomEvent(**vars(self._event), active_colour=self.
_active_colour)

```

1.24.29 volume_slider.py

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.slider_thumb import _SliderThumb

```

```

4 from data.components.custom_event import CustomEvent
5 from data.constants import SettingsEventType
6 from data.constants import WidgetState
7 from data.utils.widget_helpers import create_slider
8 from data.utils.asset_helpers import scale_and_cache
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))

```



```

58         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.
border_width)
62         self._thumb.set_position(self.relative_to_global_position((thumb_position
[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.
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)

```

1.24.30 __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']

```

1.25 data\widgets\bases

1.25.1 box.py

```

1 from data.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)

```

1.25.2 circular.py

See Section ??.

1.25.3 pressable.py

```

1 import pygame
2 from data.constants import WidgetState
3 from data.managers.audio import audio
4 from data.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

```

```

16
17     self._widget_state = WidgetState.BASE
18
19     self._colours = {}
20
21     def set_state_colour(self, state):
22         self._fill_colour = self._colours[state]
23
24         self.set_image()
25
26     def initialise_new_colours(self, colour):
27         r, g, b, a = pygame.Color(colour).rgba
28
29         self._colours = {
30             WidgetState.BASE: pygame.Color(r, g, b, a),
31             WidgetState.HOVER: pygame.Color(min(r + 25, 255), min(g + 25, 255),
min(b + 25, 255), a),
32             WidgetState.PRESS: pygame.Color(min(r + 50, 255), min(g + 50, 255),
min(b + 50, 255), a)
33         }
34
35     def get_widget_state(self):
36         return self._widget_state
37
38     def process_event(self, event):
39         match event.type:
40             case pygame.MOUSEBUTTONDOWN:
41                 if self.rect.collidepoint(event.pos):
42                     self._down_func()
43                     self._widget_state = WidgetState.PRESS
44
45             case pygame.MOUSEBUTTONUP:
46                 if self.rect.collidepoint(event.pos):
47                     if self._widget_state == WidgetState.PRESS:
48                         if self._sfx:
49                             audio.play_sfx(self._sfx)
50
51                         self._up_func()
52                         self._widget_state = WidgetState.HOVER
53                         return self._event
54
55                     elif self._widget_state == WidgetState.BASE:
56                         self._hover_func()
57
58                     elif self._prolonged and self._widget_state == WidgetState.PRESS:
59                         if self._sfx:
60                             audio.play_sfx(self._sfx)
61                         self._up_func()
62                         self._widget_state = WidgetState.BASE
63                         return self._event
64
65             case pygame.MOUSEMOTION:
66                 if self.rect.collidepoint(event.pos):
67                     if self._widget_state == WidgetState.PRESS:
68                         return
69                     elif self._widget_state == WidgetState.BASE:
70                         self._hover_func()
71                         self._widget_state = WidgetState.HOVER
72                     elif self._widget_state == WidgetState.HOVER:
73                         self._hover_func()
74                 else:
75                     if self._prolonged is False:

```

```

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

```

1.25.4 widget.py

See Section ??.