

Chapter 1

Source Code

1.1 data

1.1.1 control.py

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

```

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

```

1.1.2 loading_screen.py

See Section ??.

1.1.3 main.py

See Section ??.

1.1.4 setup.py

```

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

```

1.1.5 windows_setup.py

```

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

```

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

```

```

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

```

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\helpers

1.6.1 asset_helpers.py

See Section ??.

1.6.2 bitboard_helpers.py

```

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

```

```

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

```

1.6.3 board_helpers.py

```

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

```

```

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

```

1.6.4 browser_helpers.py

```

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

```

1.6.5 database_helpers.py

See Section ??.

1.6.6 data_helpers.py

See Section ??.

1.6.7 font_helpers.py

```

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

```

```

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

```

1.6.8 input_helpers.py

```

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

```

1.6.9 load_helpers.py

```

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

```

```

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

```



```

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

```

```
127     return music_paths
```

1.6.10 widget_helpers.py

See Section ??.

1.7 data\managers

1.7.1 animation.py

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

1.7.2 audio.py

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

```

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

```

```

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

```

1.7.3 cursor.py

```

1  import pygame
2  from data.utils.enums import CursorMode
3  from data.utils.assets import GRAPHICS
4
5  class CursorManager:
6      def __init__(self):
7          self._mode = CursorMode.ARROW
8          self.set_mode(CursorMode.ARROW)
9
10     def set_mode(self, mode):
11         pygame.mouse.set_visible(True)
12
13         match mode:
14             case CursorMode.ARROW:
15                 pygame.mouse.set_cursor((7, 5), pygame.transform.scale(GRAPHICS['
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()

```

1.7.4 logs.py

```

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

```

1.7.5 shader.py

See Section ??.

1.7.6 theme.py

See Section ??.

1.7.7 window.py

```
1 import pygame
2 import moderngl
3 from data.utils.constants import ShaderType, SCREEN_SIZE, SHADER_MAP
4 from data.helpers.data_helpers import get_user_settings
5 from data.helpers.asset_helpers import draw_background
6 from data.managers.shader import ShaderManager
7
8 user_settings = get_user_settings()
9 is_opengl = user_settings['opengl']
10 is_fullscreen = user_settings['displayMode'] == 'fullscreen'
11
12 class WindowManager(pygame.Window):
13     def __init__(self, **kwargs):
14         super().__init__(**kwargs)
15         self._native_screen = self.get_surface() # Initialise convert format
16         self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
17
18         if is_opengl:
19             self._ctx = moderngl.create_context()
20             self._shader_manager = ShaderManager(self._ctx, screen_size=self.size)
21
22             self.shader_arguments = {
23                 ShaderType.BASE: {},
24                 ShaderType.SHAKE: {},
25                 ShaderType.BLOOM: {},
26                 ShaderType.GRAYSCALE: {},
27                 ShaderType.CRT: {},
28                 ShaderType.RAYS: {}
29             }
30
31             if (selected_shader := get_user_settings()['shader']) is not None:
32                 for shader_type in SHADER_MAP[selected_shader]:
33                     self.set_effect(shader_type)
34             else:
35                 from data.utils.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,
88                        fullscreen_desktop=is_fullscreen)

```

1.8 data\shaders

1.8.1 protocol.py

```

1 import pygame
2 import moderngl
3 from typing import Protocol, Optional
4 from data.utils.constants import ShaderType
5
6 class 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: ...

```

```

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

```

1.9 data\shaders\classes

1.9.1 base.py

```

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

```

```

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

```

1.9.2 blend.py

```

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

```

1.9.3 bloom.py

See Section ??.

1.9.4 blur.py

See Section ??.

1.9.5 chromatic_abbreviation.py

```
1 import pygame
2 from data.utils.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)
```

1.9.6 crop.py

```
1 from data.utils.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)
```

1.9.7 crt.py

```
1 from data.utils.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)
```

1.9.8 grayscale.py

```

1 from data.utils.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)

```

1.9.9 highlight_brightness.py

```

1 from data.utils.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: SMPProtocol):
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)

```

1.9.10 highlight_colour.py

```

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

```

1.9.11 lightmap.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
3 from data.shaders.classes.shadowmap import _Shadowmap
4
5 LIGHT_RESOLUTION = 256
6
7 class _Lightmap:
8     def __init__(self, shader_manager: SMPProtocol):
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
15         .size)
16         self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)
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,
21         resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
22         clamp, softShadow=softShadow)

```

1.9.12 occlusion.py

```

1 from data.utils.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
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))

```

1.9.13 rays.py

See Section ??.

1.9.14 shadowmap.py

```

1 import moderngl
2 from data.utils.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)

```

1.9.15 shake.py

```

1 from data.utils.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)

```

1.9.16 __init__.py

```

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

```

1.10 data\shaders\fragments

1.10.1 background_balatro.frag

```

1 # version 330 core
2
3 // Original by localthunk (https://www.playbalatro.com)
4
5 // Configuration (modify these values to change the effect)
6 #define SPIN_ROTATION -2.0
7 #define SPIN_SPEED 7.0
8 #define OFFSET vec2(0.0)
9 #define COLOUR_2 vec4(0.871, 0.267, 0.231, 1.0)
10 #define COLOUR_1 vec4(0.0, 0.42, 0.706, 1.0)
11 #define COLOUR_3 vec4(0.086, 0.137, 0.145, 1.0)
12 #define CONTRAST 3.5
13 #define 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;

```

```

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

```

1.10.2 background_gradient.frag

```

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

```

```

50
51     tuv.y *= 1./ratio;
52     tuv *= Rot(radians((degree-.5)*720.+180.));
53     tuv.y *= ratio;
54
55     // Wave warp with sin
56     float frequency = 5.;
57     float amplitude = 30.;
58     float speed = time * 2.;
59     tuv.x += sin(tuv.y*frequency+speed)/amplitude;
60     tuv.y += sin(tuv.x*frequency*1.5+speed)/(amplitude*.5);
61
62     // draw the image
63     vec3 colorYellow = vec3(.957, .804, .623);
64     vec3 colorDeepBlue = vec3(.192, .384, .933);
65     vec3 layer1 = mix(colorYellow, colorDeepBlue, S(-.3, .2, (tuv*Rot(radians(-5.)
66     ))).x));
67
68     vec3 colorRed = vec3(.910, .510, .8);
69     vec3 colorBlue = vec3(0.350, .71, .953);
70     vec3 layer2 = mix(colorRed, colorBlue, S(-.3, .2, (tuv*Rot(radians(-5.)))).x));
71
72     vec3 finalComp = mix(layer1, layer2, S(.5, -.3, tuv.y));
73
74     vec3 col = finalComp;
75
76     f_colour = vec4(col,1.0);
77 }

```

1.10.3 background_lasers.frag

```

1 // Modified from https://www.shadertoy.com/view/7tBSR1
2 // rand [0,1] https://www.shadertoy.com/view/4djSRW
3
4 #version 330 core
5
6 uniform float time;
7 uniform vec2 screenSize;
8
9 in vec2 uvs;
10 out vec4 f_colour;
11
12 float rand(vec2 p) {
13     p *= 500.0;
14     vec3 p3 = fract(vec3(p.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 lzh = 0.5+0.5*sn;
47     lzh = lzh*lzh*lzh*lzh*lzh;
48     float glow = pow(clamp(sn, 0.0, 1.0),100.0);
49     return lzh+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*1+uvs.y*uvs.y)*c;
74     f_colour = pow(col, vec4(0.75));
75 }

```

1.10.4 background_none.frag

```

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

```

1.10.5 background_waves.frag

```

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

```



```

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

```

1.10.6 base.frag

```

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

```

```

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

```

1.10.7 blend.frag

```

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

```

1.10.8 bloom.frag

```

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

```

```

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

```

1.10.9 bloom_old.frag

```

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

```

1.10.10 blur.frag

See Section ??.

1.10.11 box_blur.frag

```

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

```

```

8 in vec2 uvs;
9 out vec4 f_colour;
10
11 vec2 textureSize = textureSize(image, 0);
12
13 void main() {
14     if (size <= 0) {
15         return;
16     }
17
18     float count = 0.0;
19
20     for (int i = -size ; i <= size ; ++i) {
21         for (int j = -size ; j <= size ; ++j) {
22             f_colour += texture(image, uvs + (vec2(i, j) * separation) /
23 textureSize).rgba;
24
25             count += 1.0;
26         }
27     }
28
29     f_colour.rgb /= count;
30 }

```

1.10.12 calibrate.frag

```

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

```

1.10.13 chromatic_abbreviation.frag

```

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

```

```

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

```

1.10.14 crop.frag

```

1  #version 330 core
2
3  uniform sampler2D image;
4  uniform vec2 relativeSize;
5  uniform vec2 relativePos;
6
7  in vec2 uvs;
8  out vec4 f_colour;
9
10 void main() {
11     vec2 sampleCoords = relativeSize.xy * uvs.xy + relativePos.xy;
12
13     float withinBounds = step(0.0, sampleCoords.x) * step(sampleCoords.x, 1.0) *
14     step(0.0, sampleCoords.y) * step(sampleCoords.y, 1.0);
15
16     vec3 colour = texture(image, sampleCoords).rgb * withinBounds;
17     colour.r += (1 - withinBounds);
18
19     f_colour = vec4(colour, 1.0);
20 }

```

1.10.15 crt.frag

```

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

```

```

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

```

1.10.16 flashlight.frag

```

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

```

1.10.17 grayscale.frag

```

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

```

```

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

```

1.10.18 highlight_brightness.frag

See Section ??.

1.10.19 highlight_colour.frag

```

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

```

1.10.20 lightmap.frag

See Section ??.

1.10.21 occlusion.frag

See Section ??.

1.10.22 rays.frag

```

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

```

1.10.23 shadowmap.frag

See Section ??.

1.10.24 shake.frag

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

1.11 data\shaders\vertex

1.11.1 base.vert

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

1.12 data\states

1.13 data\states\browser

1.13.1 browser.py

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



```

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

```

```

77         BROWSER_WIDGETS['help'].kill()
78
79     if widget_event is None:
80         return
81
82     match widget_event.type:
83         case BrowserEventType.MENU_CLICK:
84             self.next = 'menu'
85             self.done = True
86
87         case BrowserEventType.BROWSER_STRIP_CLICK:
88             self._selected_index = widget_event.selected_index
89
90         case BrowserEventType.COPY_CLICK:
91             if self._selected_index is None:
92                 return
93             logger.info('COPYING TO CLIPBOARD:', self._games_list[self.
94 _selected_index]['fen_string'])
95             pyperclip.copy(self._games_list[self._selected_index]['fen_string'
96 ])
97
98         case BrowserEventType.DELETE_CLICK:
99             if self._selected_index is None:
100                 return
101             delete_game(self._games_list[self._selected_index]['id'])
102             self.refresh_games_list()
103
104         case BrowserEventType.REVIEW_CLICK:
105             if self._selected_index is None:
106                 return
107             self.next = 'review'
108             self.done = True
109
110         case BrowserEventType.FILTER_COLUMN_CLICK:
111             selected_word = BROWSER_WIDGETS['filter_column_dropdown'].
112 get_selected_word()
113
114             if selected_word is None:
115                 return
116
117             self.refresh_games_list()
118
119         case BrowserEventType.FILTER_ASCEND_CLICK:
120             selected_word = BROWSER_WIDGETS['filter_ascend_dropdown'].
121 get_selected_word()
122
123             if selected_word is None:
124                 return
125
126             self.refresh_games_list()
127
128         case BrowserEventType.PAGE_CLICK:
129             self._page_number = widget_event.data
130
131             self.refresh_games_list()
132
133         case BrowserEventType.HELP_CLICK:
134             self._widget_group.add(BROWSER_WIDGETS['help'])
135             self._widget_group.handle_resize(window.size)
136
137     def draw(self):

```

```
135         self._widget_group.draw()
```

1.13.2 widget_dict.py

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

```

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

```

```

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

```

1.14 data\states\config

1.14.1 config.py

```

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

```

```

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

```

```

72         if pvc_enabled:
73             CONFIG_WIDGETS['pvc_button'].set_locked(True)
74             CONFIG_WIDGETS['pvp_button'].set_locked(False)
75         else:
76             CONFIG_WIDGETS['pvp_button'].set_locked(True)
77             CONFIG_WIDGETS['pvc_button'].set_locked(False)
78
79         self._config['CPU_ENABLED'] = pvc_enabled
80
81         if self._config['CPU_ENABLED']:
82             self.create_depth_picker()
83         else:
84             self.remove_depth_picker()
85
86     def set_fen_string(self, new_fen_string):
87         CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
88         self._config['FEN_STRING'] = new_fen_string
89
90         self.set_preset_overlay(new_fen_string)
91
92         try:
93             CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
94             CONFIG_WIDGETS['invalid_fen_string'].kill()
95
96             if new_fen_string[-1].lower() == 'r':
97                 self.set_active_colour(Colour.RED)
98             else:
99                 self.set_active_colour(Colour.BLUE)
100
101             self._valid_fen = True
102         except:
103             CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
104             self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])
105
106             window.set_effect(ShaderType.SHAKE)
107             animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE))
108
109         audio.play_sfx(SFX['error_1'])
110         audio.play_sfx(SFX['error_2'])
111
112         self._valid_fen = False
113
114     def get_event(self, event):
115         widget_event = self._widget_group.process_event(event)
116
117         if event.type in [pygame.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)
154
155         case ConfigEventType.PRESET_CLICK:
156             self.set_fen_string(widget_event.fen_string)
157
158         case ConfigEventType.SETUP_CLICK:
159             self.next = 'editor'
160             self.done = True
161
162         case ConfigEventType.COLOUR_CLICK:
163             self.set_active_colour(widget_event.data.get_flipped_colour())
164
165         case ConfigEventType.HELP_CLICK:
166             self._widget_group.add(CONFIG_WIDGETS['help'])
167             self._widget_group.handle_resize(window.size)
168
169     def set_preset_overlay(self, fen_string):
170         fen_string_widget_map = {
171             'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
PdNaFaNa3Sa b': 'preset_1',
172             'sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3/10/2
RaNaFaNa3Sa b': 'preset_2',
173             'sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3RaNaPa3Pc/4Fa5
/3PdNaPa3Sa b': 'preset_3'
174         }
175
176         if fen_string in fen_string_widget_map:
177             self._selected_preset = CONFIG_WIDGETS[fen_string_widget_map[
fen_string]]
178         else:
179             self._selected_preset = None
180
181     def set_active_colour(self, colour):
182         if self._config['COLOUR'] != colour:
183             CONFIG_WIDGETS['to_move_button'].set_next_icon()
184
185             self._config['COLOUR'] = colour
186
187             if colour == Colour.BLUE:
188                 CONFIG_WIDGETS['to_move_text'].set_text('BLUE TO MOVE')
189             elif colour == Colour.RED:
190                 CONFIG_WIDGETS['to_move_text'].set_text('RED TO MOVE')

```



```

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

```

1.14.2 default_config.py

```

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

```

1.14.3 widget_dict.py

```

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

```

```

27
28 if default_config['TIME_ENABLED']:
29     time_enabled_icons = {True: GRAPHICS['timer'], False: get_highlighted_icon(
    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',
107             base_icon=GRAPHICS['help_base'],
108             hover_icon=GRAPHICS['help_hover'],
109             press_icon=GRAPHICS['help_press'],
110             event=CustomEvent(ConfigEventType.HELP_CLICK)
111         ),
112         TextInput(
113             parent=config_container,
114             relative_position=(0.3, 0.3),
115             relative_size=(0.65, 0.15),
116             fit_vertical=True,
117             placeholder='TIME CONTROL (DEFAULT 5)',
118             default=str(default_config['TIME']),
119             border_width=5,
120             margin=20,
121             validator=float_validator,
122             event=CustomEvent(ConfigEventType.TIME_TYPE)
123         ),
124         Text(
125             parent=config_container,
126             fit_vertical=False,
127             relative_position=(0.75, 0.3),
128             relative_size=(0.2, 0.15),
129             text='MINS',
130             border_width=0,
131             fill_colour=(0, 0, 0, 0)
132         ),
133         TextButton(
134             parent=preview_container,
135             relative_position=(0.3, 0),
136             relative_size=(0.15, 0.15),
137             text='CUSTOM',
138             anchor_y='bottom',
139             fit_vertical=False,
140             margin=10,
141             event=CustomEvent(ConfigEventType.SETUP_CLICK)
142         )
143     ],
144     'board_thumbnail':
145         board_thumbnail,
146     'fen_string_input':

```

```

147     TextInput(
148         parent=preview_container,
149         relative_position=(0, 0),
150         relative_size=(0.55, 0.15),
151         fit_vertical=False,
152         placeholder='ENTER FEN STRING',
153         default='sc3ncfcncpb2/2pc7/3Pd7/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
Pa2/2PdNaFaNa3Sa b',
154         border_width=5,
155         anchor_y='bottom',
156         anchor_x='right',
157         margin=20,
158         event=CustomEvent(ConfigEventType.FEN_STRING_TYPE)
159     ),
160     'start_button':
161     TextButton(
162         parent=config_container,
163         relative_position=(0, 0),
164         relative_size=(0.9, 0.3),
165         anchor_y='bottom',
166         anchor_x='center',
167         text='START NEW GAME',
168         strength=0.1,
169         text_colour=theme['textSecondary'],
170         margin=20,
171         fit_vertical=False,
172         event=CustomEvent(ConfigEventType.GAME_CLICK)
173     ),
174     'timer_button':
175     MultipleIconButton(
176         parent=config_container,
177         scale_mode='height',
178         relative_position=(0.05, 0.3),
179         relative_size=(0.15, 0.15),
180         margin=10,
181         border_width=5,
182         border_radius=5,
183         icons_dict=time_enabled_icons,
184         event=CustomEvent(ConfigEventType.TIME_CLICK)
185     ),
186     'pvp_button':
187     MultipleIconButton(
188         parent=config_container,
189         relative_position=(-0.225, 0.5),
190         relative_size=(0.45, 0.15),
191         margin=15,
192         anchor_x='center',
193         icons_dict=pvp_icons,
194         stretch=False,
195         event=CustomEvent(ConfigEventType.PVP_CLICK)
196     ),
197     'pvc_button':
198     MultipleIconButton(
199         parent=config_container,
200         relative_position=(0.225, 0.5),
201         relative_size=(0.45, 0.15),
202         anchor_x='center',
203         margin=15,
204         icons_dict=pvc_icons,
205         stretch=False,
206         event=CustomEvent(ConfigEventType.PVC_CLICK)
207     ),

```

```

208     'invalid_fen_string':
209     Text(
210         parent=board_thumbnail,
211         relative_position=(0, 0),
212         relative_size=(0.9, 0.1),
213         fit_vertical=False,
214         anchor_x='center',
215         anchor_y='center',
216         text='INVALID FEN STRING!',
217         margin=10,
218         fill_colour=theme['fillError'],
219         text_colour=theme['textError'],
220     ),
221     'preset_1':
222     BoardThumbnailButton(
223         parent=preview_container,
224         relative_width=0.25,
225         relative_position=(0, 0),
226         scale_mode='width',
227         fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3
228 /7Pa2/2PdNaFaNa3Sa b",
229         event=CustomEvent(ConfigEventType.PRESET_CLICK)
230     ),
231     'preset_2':
232     BoardThumbnailButton(
233         parent=preview_container,
234         relative_width=0.25,
235         relative_position=(0, 0.35),
236         scale_mode='width',
237         fen_string="sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3
238 /10/2RaNaFaNa3Sa b",
239         event=CustomEvent(ConfigEventType.PRESET_CLICK)
240     ),
241     'preset_3':
242     BoardThumbnailButton(
243         parent=preview_container,
244         relative_width=0.25,
245         relative_position=(0, 0.7),
246         scale_mode='width',
247         fen_string="sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3
248 RaNaPa3Pc/4Fa5/3PdNaPa3Sa b",
249         event=CustomEvent(ConfigEventType.PRESET_CLICK)
250     ),
251     'to_move_button':
252     MultipleIconButton(
253         parent=to_move_container,
254         scale_mode='height',
255         relative_position=(0, 0),
256         relative_size=(1, 1),
257         icons_dict=colour_icons,
258         anchor_x='left',
259         event=CustomEvent(ConfigEventType.COLOUR_CLICK)
260     ),
261     'to_move_text':
262     Text(
263         parent=to_move_container,
264         relative_position=(0, 0),
265         relative_size=(0.75, 1),
266         fit_vertical=False,
267         text='TO MOVE',
268         anchor_x='right'
269     ),

```

```

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 }

```

1.15 data\states\editor

1.15.1 editor.py

```

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

```

```

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

```

```

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

```



```

132         logger.info(f'COPYING TO CLIPBOARD: {encode_fen_string(self.
_bitboards)}')
133         pyperclip.copy(encode_fen_string(self._bitboards))
134
135         case EditorEventType.BLUE_START_CLICK:
136             self.set_starting_colour(Colour.BLUE)
137
138         case EditorEventType.RED_START_CLICK:
139             self.set_starting_colour(Colour.RED)
140
141         case EditorEventType.START_CLICK:
142             self.next = 'config'
143             self.done = True
144
145         case EditorEventType.CONFIG_CLICK:
146             self.reset_board()
147             self.next = 'config'
148             self.done = True
149
150         case EditorEventType.ERASE_CLICK:
151             if self._selected_tool == 'ERASE':
152                 self.deselect_tool()
153             else:
154                 self.select_tool('ERASE', None)
155
156         case EditorEventType.MOVE_CLICK:
157             if self._selected_tool == 'MOVE':
158                 self.deselect_tool()
159             else:
160                 self.select_tool('MOVE', None)
161
162         case EditorEventType.HELP_CLICK:
163             self._widget_group.add(EDITOR_WIDGETS['help'])
164             self._widget_group.handle_resize(window.size)
165
166     def reset_board(self):
167         self._bitboards = BitboardCollection(self._initial_fen_string)
168         self.refresh_pieces()
169
170     def refresh_pieces(self):
171         self._piece_group.initialise_pieces(self._bitboards.convert_to_piece_list
(), 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.
BLUE) or self._bitboards.get_piece_on(bitboard_under_mouse, Colour.RED)

```

```

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

```

```

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

```

1.15.2 widget_dict.py

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

```

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

```

```

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

```

```

181         box_colours=BLUE_BUTTON_COLOURS,
182         icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
183         ['finger'])},
184         event=CustomEvent(EditorEventType.MOVE_CLICK),
185     ),
186     'chessboard':
187     Chessboard(
188         relative_position=(0, 0),
189         relative_width=0.4,
190         scale_mode='width',
191         anchor_x='center',
192         anchor_y='center'
193     ),
194     'blue_start_button':
195     MultipleIconButton(
196         parent=bottom_actions_container,
197         relative_position=(0, 0),
198         relative_size=(1, 1),
199         scale_mode='height',
200         anchor_x='right',
201         box_colours=BLUE_BUTTON_COLOURS,
202         icons_dict={False: get_highlighted_icon(GRAPHICS['pharoah_0_a']), True:
203         GRAPHICS['pharoah_0_a']},
204         event=CustomEvent(EditorEventType.BLUE_START_CLICK)
205     ),
206     'red_start_button':
207     MultipleIconButton(
208         parent=bottom_actions_container,
209         relative_position=(0, 0),
210         relative_size=(1, 1),
211         scale_mode='height',
212         icons_dict={True: GRAPHICS['pharoah_1_a'], False: get_highlighted_icon(
213         GRAPHICS['pharoah_1_a'])},
214         event=CustomEvent(EditorEventType.RED_START_CLICK)
215     )
216 }
217
218 for index, piece in enumerate([piece for piece in Piece if piece != Piece.SPHINX])
219 :
220     blue_icon = GRAPHICS[f'{piece.name.lower()}_0_a']
221     dimmed_blue_icon = get_highlighted_icon(blue_icon)
222
223     EDITOR_WIDGETS['blue_piece_buttons'][piece] = MultipleIconButton(
224         parent=blue_pieces_container,
225         relative_position=(0, (index + 1) / 5),
226         relative_size=(0.2, 0.2),
227         scale_mode='height',
228         box_colours=BLUE_BUTTON_COLOURS,
229         icons_dict={True: blue_icon, False: dimmed_blue_icon},
230         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
231         active_colour=Colour.BLUE)
232     )
233
234     red_icon = GRAPHICS[f'{piece.name.lower()}_1_a']
235     dimmed_red_icon = get_highlighted_icon(red_icon)
236
237     EDITOR_WIDGETS['red_piece_buttons'][piece] = MultipleIconButton(
238         parent=red_pieces_container,
239         relative_position=(0, (index + 1) / 5),
240         relative_size=(0.2, 0.2),
241         scale_mode='height',

```

```

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

```

1.16 data\states\game

1.16.1 game.py

```

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

```



```

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

```

1.16.2 widget_dict.py

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

1.17 data\states\game\components

1.17.1 bitboard_collection.py

See Section ??.

1.17.2 board.py

See Section ??.

1.17.3 capture_draw.py

```

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

```

```

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

```

1.17.4 father.py

```

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

```



```

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

```

1.17.5 fen_parser.py

```

1 from data.helpers.bitboard_helpers import occupied_squares, bitboard_to_index
2 from data.utils.enums import Colour, RotationIndex, Rotation, Piece
3 from data.utils.constants import EMPTY_BB
4
5 def parse_fen_string(fen_string):
6     #sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
PdNaFaNa3Sa b
7     piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char: EMPTY_BB for
char in Piece}]
8     rotation_bitboards = [EMPTY_BB, EMPTY_BB]
9     combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
10    combined_all_bitboard = 0
11    part_1, part_2 = fen_string.split(' ')
12
13    rank = 7
14    file = 0
15
16    piece_count = {char.lower(): 0 for char in Piece} | {char.upper(): 0 for char
in Piece}
17
18    for index, character in enumerate(part_1):
19        square = rank * 10 + file
20
21        if character.lower() in Piece:
22            piece_count[character] += 1
23            if character.isupper():
24                piece_bitboards[Colour.BLUE][character.lower()] |= 1 << square
25
26        else:
27            piece_bitboards[Colour.RED][character.lower()] |= 1 << square
28
29        rotation = part_1[index + 1]
30        match rotation:
31            case Rotation.UP:
32                pass
33            case Rotation.RIGHT:
34                rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
35            case Rotation.DOWN:
36                rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square

```

```

37         case Rotation.LEFT:
38             rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
39             rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
40         case _:
41             raise ValueError('Invalid FEN String - piece character not
followed by rotational character')
42
43         file += 1
44         elif character in '0123456789':
45             if character == '1' and fen_string[index + 1] == '0':
46                 file += 10
47                 continue
48
49             file += int(character)
50         elif character == '/':
51             rank = rank - 1
52             file = 0
53         elif character in Rotation:
54             continue
55         else:
56             raise ValueError('Invalid FEN String - invalid character found:',
character)
57
58     if piece_count['s'] != 1 or piece_count['S'] != 1:
59         raise ValueError('Invalid FEN string - invalid number of Sphinx pieces')
60     # COMMENTED OUT AS NO PHAROAH PIECES IS OKAY IF PARSING FEN STRING FOR
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
118     fen_string = fen_string[1:]
119
120     if bitboard_collection.active_colour == Colour.BLUE:
121         colour = 'b'
122     else:
123         colour = 'r'
124
125     return fen_string + ' ' + colour

```

1.17.6 laser.py

```

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

```

```

21
22         while current_square:
23             current_piece = self._bitboards.get_piece_on(current_square, Colour.
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.
get_anticlockwise()]:
76                 new_direction = previous_direction.get_anticlockwise()
77             else:
78                 new_direction = previous_direction.get_clockwise()

```

```

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

```

1.17.7 laser_draw.py

See Section ??.

1.17.8 move.py

```

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

```

```

25         else:
26             return 'R' + piece + bitboard_to_notation(self.src) + self.
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,
88                             rotation_direction=None):
89         try:
90             src_bitboard = coords_to_bitboard(src_coords)
91             dest_bitboard = coords_to_bitboard(dest_coords)
92
93             return move_cls(move_type, src_bitboard, dest_bitboard,
94                             rotation_direction)
95         except Exception as error:
96             logger.info('Error (Move.instance_from_coords):', error)
97             raise error
98
99     @classmethod
100     def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
101                                None, rotation_direction=None):
102         try:
103             return move_cls(move_type, src_bitboard, dest_bitboard,
104                             rotation_direction)
105         except Exception as error:
106             logger.info('Error (Move.instance_from_bitboards):', error)
107             raise error

```

1.17.9 overlay_draw.py

```

1 import pygame
2 from data.utils.constants import OVERLAY_COLOUR_LIGHT, OVERLAY_COLOUR_DARK
3 from data.helpers.board_helpers import coords_to_screen_pos, screen_pos_to_coords,
4   create_square_overlay, create_circle_overlay
5
6 class OverlayDraw:
7     def __init__(self, board_position, board_size, limit_hover=True):
8         self._board_position = board_position
9         self._board_size = board_size
10
11         self._hovered_coords = None
12         self._selected_coords = None
13         self._available_coords = None
14
15         self._limit_hover = limit_hover
16
17         self._selected_overlay = None
18         self._hovered_overlay = None
19         self._available_overlay = None
20
21         self.initialise_overlay_surfaces()
22
23     @property
24     def square_size(self):
25         return self._board_size[0] / 10
26
27     def initialise_overlay_surfaces(self):
28         self._selected_overlay = create_square_overlay(self.square_size,
29                                                         OVERLAY_COLOUR_DARK)
30         self._hovered_overlay = create_square_overlay(self.square_size,
31                                                         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()

```

1.17.10 particles_draw.py

See Section ??.

1.17.11 piece_group.py

```

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

```



```

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

```

1.17.12 piece_sprite.py

```

1 import pygame
2 from data.helpers.board_helpers import coords_to_screen_pos
3 from data.helpers.asset_helpers import scale_and_cache
4 from data.utils.assets import GRAPHICS
5 from data.utils.enums import Piece
6
7 class PieceSprite(pygame.sprite.Sprite):
8     def __init__(self, piece, colour, rotation):
9         super().__init__()
10         self.colour = colour
11         self.rotation = rotation
12
13         self.type = piece
14         self.coords = None
15         self.size = None
16
17     @property
18     def image_name(self):
19         return Piece(self.type).name.lower() + '_' + str(self.colour) + '_' + self
20         .rotation
21
22     def set_image(self):
23         self.image = scale_and_cache(GRAPHICS[self.image_name], (self.size, self
24         .size))
25
26     def set_geometry(self, new_position, square_size):

```

```

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

```

1.17.13 psqt.py

```

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

```

```

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

```

1.18 data\states\game\cpu

1.18.1 base.py

```

1  import time
2  from pprint import PrettyPrinter
3  from data.utils.enums import Colour, Score, Miscellaneous
4  from data.states.game.cpu.evaluator import Evaluator
5  from data.managers.logs import initialise_logger
6
7  logger = initialise_logger(__name__)
8  printer = PrettyPrinter(indent=2, sort_dicts=False)
9
10 class BaseCPU:
11     def __init__(self, callback, verbose=True):
12         self._evaluator = Evaluator(verbose=False)
13         self._verbose = verbose
14         self._callback = callback
15         self._stats = {}
16
17     def initialise_stats(self):
18         self._stats = {
19             'nodes': 0,
20             'leaf_nodes': 0,
21             'draws': 0,
22             'mates': 0,
23             'ms_per_node': 0,
24             'time_taken': time.time()
25         }
26
27     def print_stats(self, score, move):
28         """
29         Prints statistics after traversing tree.
30
31         Args:
32             score (int): Final score obtained after traversal.
33             move (Move): Best move obtained after traversal.
34         """
35         if self._verbose is False:
36             return
37
38         self._stats['time_taken'] = round(1000 * (time.time() - self._stats['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 TimeoutError(f'Thread killed - stopping minimax function ({self.
__str__}.search)')
60
61         self._stats['nodes'] += 1
62
63         if (winner := board.check_win()) is not None:
64             self._stats['leaf_nodes'] += 1
65             return self.process_win(winner, depth, absolute)
66
67         if depth == 0:
68             self._stats['leaf_nodes'] += 1
69             return self._evaluator.evaluate(board, absolute), None
70
71     def process_win(self, winner, depth, absolute):
72         self._stats['leaf_nodes'] += 1
73
74         if winner == Miscellaneous.DRAW:
75             self._stats['draws'] += 1
76             return 0, None
77         elif winner == Colour.BLUE or absolute:
78             self._stats['mates'] += 1
79             return Score.CHECKMATE + depth, None
80         elif winner == Colour.RED:
81             self._stats['mates'] += 1
82             return -Score.CHECKMATE - depth, None
83
84     def __str__(self):
85         return self.__class__.__name__

```

1.18.2 cpu_thread.py

See Section ??.

1.18.3 evaluator.py

See Section ??.

1.18.4 move_orderer.py

```

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

```

```

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

```

1.18.5 temp.py

```

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

```

```

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

```

```

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

```

1.18.6 transposition_table.py

See Section ??.

1.18.7 zobrist_hasher.py

See Section ??.

1.19 data\states\game\cpu\engines

1.19.1 alpha_beta.py

See Section ??.

1.19.2 iterative_deepening.py

See Section ??.

1.19.3 minimax.py

See Section ??.

1.19.4 negamax.py

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



```

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

```

```

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

```

1.19.5 simple.py

```

1 from data.states.game.cpu.base import BaseCPU
2 from data.utils.enums import Colour, Score
3
4 class SimpleCPU(BaseCPU):
5     def __init__(self, callback, verbose=True):
6         super().__init__(callback, verbose)
7
8     def find_move(self, board, stop_event=None):
9         self.initialise_stats()
10         best_score, best_move = self.search(board, stop_event)
11
12         if self._verbose:
13             self.print_stats(best_score, best_move)
14
15         self._callback(best_move)
16
17     def search(self, board, stop_event):
18         if stop_event and stop_event.is_set():
19             raise Exception('Thread killed - stopping simple function (SimpleCPU.
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 (
38             active_colour == Colour.RED and score < best_score):
39             best_move = move
40             best_score = score
41
42         board.undo_move(move, laser_result)
43
44         return best_score, best_move

```

1.19.6 transposition_table.py

See Section ??.

1.19.7 __init__.py

```

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

```

1.20 data\states\game\mvc

1.20.1 game_controller.py

See Section ??.

1.20.2 game_model.py

See Section ??.

1.20.3 game_view.py

See Section ??.

1.20.4 pause_view.py

```

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

```

```

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

```

1.20.5 win_view.py

```

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

```

```

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

```

1.21 data\states\menu

1.21.1 menu.py

```

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

```

```

24
25     def cleanup(self):
26         super().cleanup()
27
28         window.clear_apply_arguments(ShaderType.BLOOM)
29         window.clear_apply_arguments(ShaderType.SHAKE)
30         window.clear_effect(ShaderType.CHROMATIC_ABBREVIATION)
31
32         return None
33
34     def startup(self, persist=None):
35         super().startup(MENU_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
36         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
37         window.set_effect(ShaderType.CHROMATIC_ABBREVIATION)
38
39         MENU_WIDGETS['credits'].kill()
40
41         self._fire_laser = False
42         self._bloom_mask = None
43         self._laser_mask = None
44
45         self.draw()
46         self.update_masks()
47
48     @property
49     def sphinx_center(self):
50         return (window.size[0] - self.sphinx_size[0] / 2, window.size[1] - self.
sphinx_size[1] / 2)
51
52     @property
53     def sphinx_size(self):
54         return (min(window.size) * 0.1, min(window.size) * 0.1)
55
56     @property
57     def sphinx_rotation(self):
58         mouse_pos = (pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1] + 0.01)
59         return -get_rotational_angle(mouse_pos, self.sphinx_center)
60
61     def get_event(self, event):
62         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
63             MENU_WIDGETS['credits'].kill()
64
65             if event.type == pygame.MOUSEBUTTONDOWN:
66                 self._fire_laser = True
67                 audio.play_sfx(SFX['menu_laser_windup'])
68                 audio.play_sfx(SFX['menu_laser_loop'], loop=True)
69                 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
70
71             elif event.type == pygame.MOUSEBUTTONUP:
72                 self._fire_laser = False
73
74                 window.clear_effect(ShaderType.RAYS)
75                 animation.set_timer(300, lambda: window.clear_effect(ShaderType.SHAKE)
)
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):
104         sphinx_surface = scale_and_cache(GRAPHICS['sphinx_0_b'], self.sphinx_size)
105         sphinx_surface = pygame.transform.rotate(sphinx_surface, self.
sphinx_rotation)
106         sphinx_rect = pygame.Rect(0, 0, *self.sphinx_size)
107         sphinx_rect.center = self.sphinx_center
108
109         window.screen.blit(sphinx_surface, sphinx_rect)
110
111     def update_masks(self):
112         self.draw()
113
114         widget_mask = window.screen.copy()
115         laser_mask = pygame.mask.from_surface(widget_mask)
116         laser_mask = laser_mask.to_surface(setcolor=(255, 0, 0, 255), unsetcolor
=(0, 0, 0, 255))
117         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)
)
118         pygame.draw.rect(widget_mask, (0, 0, 0, 255), (window.screen.width - 50,
0, 50, 50))
119
120         self._bloom_mask = widget_mask
121         self._laser_mask = laser_mask
122
123     def draw(self):
124         self._widget_group.draw()
125         self.draw_sphinx()
126
127         if self._fire_laser:
128             window.set_apply_arguments(ShaderType.RAYS, occlusion=self._laser_mask
, softShadow=0.1)
129
130             window.set_apply_arguments(ShaderType.BLOOM, highlight_surface=self.
_bloom_mask, surface_intensity=0.3, brightness_intensity=0.6)
131
132     def update(self, **kwargs):
133         random_offset = lambda: randint(-5, 5) / 40
134         if self._fire_laser:
135             window.clear_effect(ShaderType.RAYS)

```

```

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

```

1.21.2 widget_dict.py

```

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

```



```

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

```

```

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

```

1.22 data\states\review

1.22.1 review.py

See Section ??.

1.22.2 widget_dict.py

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

```

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

```

```

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

```

```

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

```

1.23 data\states\settings

1.23.1 settings.py

```

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

```

```

38
39 def create_colour_picker(self, mouse_pos, button_type):
40     if button_type == SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK:
41         selected_colour = self._settings['primaryBoardColour']
42         event_type = SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK
43     else:
44         selected_colour = self._settings['secondaryBoardColour']
45         event_type = SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK
46
47     self._colour_picker = ColourPicker(
48         relative_position=(mouse_pos[0] / window.size[0], mouse_pos[1] /
49 window.size[1]),
49         relative_width=0.15,
50         selected_colour=selected_colour,
51         event_type=event_type
52     )
53     self._widget_group.add(self._colour_picker)
54
55 def remove_colour_picker(self):
56     self._colour_picker.kill()
57
58 def reload_display_mode(self):
59     relative_mouse_pos = (pygame.mouse.get_pos()[0] / window.size[0], pygame.
60 mouse.get_pos()[1] / window.size[1])
61
62     if self._settings['displayMode'] == 'fullscreen':
63         window.set_fullscreen(desktop=True)
64         window.handle_resize()
65
66     elif self._settings['displayMode'] == 'windowed':
67         window.set_windowed()
68         window.handle_resize()
69         window.restore()
70
71     self._widget_group.handle_resize(window.size)
72
73     new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
74 relative_mouse_pos[1] * window.size[1])
75     pygame.mouse.set_pos(new_mouse_pos)
76
77 def reload_shaders(self):
78     window.clear_all_effects()
79
80     for shader_type in SHADER_MAP[self._settings['shader']]:
81         window.set_effect(shader_type)
82
83 def reload_settings(self):
84     SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self.
85 _settings['primaryBoardColour'])
86     SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self.
87 _settings['secondaryBoardColour'])
88     SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.
89 BASE)
90     SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.
91 BASE)
92     SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['
93 musicVolume'])
94     SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume
95 '])
96     SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings
97 ['displayMode'])
98     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':
113                     audio.set_sfx_volume(widget_event.volume)
114                     self._settings['sfxVolume'] = widget_event.volume
115
116             case SettingsEventType.DROPDOWN_CLICK:
117                 selected_word = SETTINGS_WIDGETS['display_mode_dropdown'].
get_selected_word()
118
119                 if selected_word is None or selected_word == self._settings['
displayMode']:
120                     return
121
122                 self._settings['displayMode'] = selected_word
123
124                 self.reload_display_mode()
125
126             case SettingsEventType.MENU_CLICK:
127                 self.next = 'menu'
128                 self.done = True
129
130             case SettingsEventType.RESET_DEFAULT:
131                 self._settings = get_default_settings()
132                 self.reload_settings()
133
134             case SettingsEventType.RESET_USER:
135                 self._settings = get_user_settings()
136                 self.reload_settings()
137
138             case SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK | SettingsEventType
.SECONDARY_COLOUR_BUTTON_CLICK:
139                 if self._colour_picker:
140                     self.remove_colour_picker()
141
142                 self.create_colour_picker(event.pos, widget_event.type)
143
144             case SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK | SettingsEventType
.SECONDARY_COLOUR_PICKER_CLICK:
145                 if widget_event.colour:

```



```

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

```

1.23.2 widget_dict.py

```

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

```

```

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

```

```

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

```

```

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

```

1.24.1 assets.py

1.24.2 constants.py

109

```

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

```

1.24.3 enums.py

```

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

```

```

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

```

```

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

```



```

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

```

1.24.4 event_types.py

```

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

```

```

41     HELP_CLICK = auto()
42     TUTORIAL_CLICK = auto()
43     RESIGN_CLICK = auto()
44     DRAW_CLICK = auto()
45     REVIEW_CLICK = auto()
46     PIECE_DROP = auto()
47     UPDATE_PIECES = auto()
48     ROTATE_PIECE = auto()
49     SET_LASER = auto()
50     TIMER_END = auto()
51
52     class MenuEventType(StrEnum):
53         CONFIG_CLICK = auto()
54         SETTINGS_CLICK = auto()
55         BROWSER_CLICK = auto()
56         QUIT_CLICK = auto()
57         CREDITS_CLICK = auto()
58
59     class SettingsEventType(StrEnum):
60         RESET_DEFAULT = auto()
61         RESET_USER = auto()
62         MENU_CLICK = auto()
63         COLOUR_SLIDER_SLIDE = auto()
64         COLOUR_SLIDER_CLICK = auto()
65         COLOUR_PICKER_HOVER = auto()
66         PRIMARY_COLOUR_PICKER_CLICK = auto()
67         SECONDARY_COLOUR_PICKER_CLICK = auto()
68         PRIMARY_COLOUR_BUTTON_CLICK = auto()
69         SECONDARY_COLOUR_BUTTON_CLICK = auto()
70         VOLUME_SLIDER_SLIDE = auto()
71         VOLUME_SLIDER_CLICK = auto()
72         SHADER_PICKER_CLICK = auto()
73         OPENGL_CLICK = auto()
74         DROPDOWN_CLICK = auto()
75         PARTICLES_CLICK = auto()
76
77     class ConfigEventType(StrEnum):
78         GAME_CLICK = auto()
79         MENU_CLICK = auto()
80         FEN_STRING_TYPE = auto()
81         TIME_TYPE = auto()
82         TIME_CLICK = auto()
83         PVP_CLICK = auto()
84         PVC_CLICK = auto()
85         CPU_DEPTH_CLICK = auto()
86         PRESET_CLICK = auto()
87         SETUP_CLICK = auto()
88         COLOUR_CLICK = auto()
89         HELP_CLICK = auto()

```

1.25 data\widgets

1.25.1 board_thumbnail.py

```

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

```

```

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

```

1.25.2 board_thumbnail_button.py

```

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

```

```

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

```

1.25.3 browser_item.py

```

1  import pygame
2  from data.helpers.font_helpers import text_width_to_font_size
3  from data.helpers.browser_helpers import get_winner_string
4  from data.widgets.board_thumbnail import BoardThumbnail
5  from data.helpers.asset_helpers import scale_and_cache
6  from data.widgets.bases.widget import _Widget
7
8  FONT_DIVISION = 7
9
10 class BrowserItem(_Widget):
11     def __init__(self, relative_width, game, **kwargs):
12         super().__init__(relative_size=(relative_width, relative_width * 2),
13                          scale_mode='height', **kwargs)
14
15         self._relative_font_size = text_width_to_font_size('YYYY-MM-DD HH:MM:SS',
16 self._font, self.size[0]) / self.surface_size[1]
17
18         self._game = game
19         self._board_thumbnail = BoardThumbnail(
20             relative_position=(0, 0),
21             scale_mode='height',
22             relative_width=relative_width,
23             fen_string=self._game['final_fen_string']
24         )
25
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         ]
51
52     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
[0], self.size[0] * 0.8))
52         self.image.blit(resized_board, (0, 0))
53
54         get_line_y = lambda line: (self.size[0] * 0.8) + ((self.size[0] * 0.8) /
FONT_DIVISION) * (line + 0.5)
55
56         text_to_render = self.get_text_to_render()
57
58         for index, text in enumerate(text_to_render):
59             self._font.render_to(self.image, (0, get_line_y(index)), text, fgcolor
=self._text_colour, size=self.font_size)
60
61     def process_event(self, event):
62         pass

```

1.25.4 browser_strip.py

```

1  import pygame
2  from data.components.custom_event import CustomEvent
3  from data.utils.event_types import BrowserEventType
4  from data.widgets.browser_item import BrowserItem
5  from data.widgets.bases.widget import _Widget
6
7  WIDTH_FACTOR = 0.3
8
9  class BrowserStrip(_Widget):
10     def __init__(self, relative_height, games_list, **kwargs):
11         super().__init__(relative_size=None, **kwargs)
12         self._relative_item_width = relative_height / 2
13         self._get_rect = None
14
15         self._games_list = []
16         self._items_list = []
17         self._selected_index = None
18
19         self.initialise_games_list(games_list)
20
21     @property
22     def item_width(self):
23         return self._relative_item_width * self.surface_size[1]
24
25     @property
26     def size(self):
27         if self._get_rect:
28             height = self._get_rect().height
29         else:
30             height = 0
31         width = max(0, len(self._games_list) * (self.item_width + self.margin) +
self.margin)
32
33         return (width, height)
34
35     def register_get_rect(self, get_rect_func):
36         self._get_rect = get_rect_func
37
38     def initialise_games_list(self, games_list):
39         self._items_list = []
40         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,
relative_width=self._relative_item_width)
45             self._items_list.append(browser_item)
46
47             self.set_image()
48             self.set_geometry()
49
50     def set_image(self):
51         self.image = pygame.Surface(self.size, pygame.SRCALPHA)
52         browser_list = []
53
54         for index, item in enumerate(self._items_list):
55             item.set_image()
56             browser_list.append((item.image, (index * (self.item_width + self.
margin) + self.margin, self.margin)))
57
58         self.image.blit(browser_list)
59
60         if self._selected_index is not None:
61             border_position = (self._selected_index * (self.item_width + self.
margin), 0)
62             border_size = (self.item_width + 2 * self.margin, self.size[1])
63             pygame.draw.rect(self.image, (255, 255, 255), (*border_position, *
border_size), width=int(self.item_width / 20))
64
65     def set_geometry(self):
66         super().set_geometry()
67         for item in self._items_list:
68             item.set_geometry()
69
70     def set_surface_size(self, new_surface_size):
71         super().set_surface_size(new_surface_size)
72
73         for item in self._items_list:
74             item.set_surface_size(new_surface_size)
75
76     def process_event(self, event, scrolled_pos):
77         parent_pos = self._get_rect().topleft
78         self.rect.topleft = parent_pos
79
80         if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
81             self._selected_index = None
82             self.set_image()
83             return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
selected_index=None)
84
85         if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.
pos):
86             relative_mouse_pos = (event.pos[0] - parent_pos[0], event.pos[1] -
parent_pos[1])
87             self._selected_index = int(max(0, (relative_mouse_pos[0] - self.margin
) // (self.item_width + self.margin)))
88             self.set_image()
89             return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
selected_index=self._selected_index)

```

1.25.5 carousel.py

```

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

```

```

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

```

```

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

```

1.25.6 chessboard.py

```

1  import pygame
2  from data.helpers.data_helpers import get_user_settings
3  from data.helpers.board_helpers import create_board
4  from data.widgets.bases.widget import _Widget
5  from data.utils.enums import CursorMode
6  from data.managers.cursor import cursor
7
8  class Chessboard(_Widget):
9      def __init__(self, relative_width, change_cursor=True, **kwargs):
10         super().__init__(relative_size=(relative_width, relative_width * 0.8), **
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)

```

1.25.7 colour_button.py

```

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

```

1.25.8 colour_display.py

```

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

```

1.25.9 colour_picker.py

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

```

1.25.10 colour_slider.py

See Section ??.

1.25.11 colour_square.py

```

1  import pygame
2  from data.widgets.bases.widget import _Widget
3  from data.helpers.widget_helpers import create_square_gradient
4
5  class _ColourSquare(_Widget):
6      def __init__(self, relative_width, **kwargs):
7          super().__init__(relative_size=(relative_width, relative_width),
8                          scale_mode='width', **kwargs)
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

```

1.25.12 dropdown.py

```

1  import pygame
2  from data.widgets.bases.widget import _Widget
3  from data.widgets.bases.pressable import _Pressable
4  from data.utils.constants import WidgetState
5  from data.helpers.data_helpers import get_user_settings
6  from data.helpers.font_helpers import text_width_to_font_size,
        text_height_to_font_size
7  from data.utils.assets import GRAPHICS
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)

```

1.25.13 icon.py

```

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

```

```

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

```

1.25.14 icon_button.py


```

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

```

1.25.15 move_list.py

```

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

```

```

39         self._move_list.append(new_move)
40         self.set_image()
41         self.set_geometry()
42
43     def pop_from_move_list(self):
44         self._move_list.pop()
45         self.set_image()
46         self.set_geometry()
47
48     def set_image(self):
49         self.image = pygame.transform.scale(self._empty_surface, self.size)
50         self.image.fill(self._fill_colour)
51
52         font_metrics = self._font.get_metrics('j', size=self.font_size)
53         row_gap = font_metrics[0][3] - font_metrics[0][2]
54
55         for index, move in enumerate(self._move_list):
56             if index % 2 == 0:
57                 text_position = (self.size[0] / 7, row_gap * (1 + 2 * (index // 2)
58 ))
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             self._font.render_to(self.image, move_number_position, text=str(
70 move_number), size=self.font_size, fgcolor=self._text_colour)
71
72     def process_event(self, event, scrolled_pos=None):
73         pass

```

1.25.16 multiple_icon_button.py

```

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

```

1.25.17 piece_display.py

```

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

```

```

31
32     def reset_piece_list(self):
33         self._piece_list = []
34         self.initialise_piece_surface()
35
36     def initialise_piece_surface(self):
37         self._piece_surface = pygame.Surface((self.size[0] - 2 * self.margin, self
38 .size[1] - 2 * self.margin), pygame.SRCALPHA)
39
40         if (len(self._piece_list) == 0):
41             self.set_image()
42             return
43
44         piece_width = min(self.size[1] - 2 * self.margin, (self.size[0] - 2 * self
45 .margin) / len(self._piece_list))
46         piece_list = []
47
48         for index, piece in enumerate(self._piece_list):
49             piece_instance = PieceSprite(piece, self._active_colour.
50 get_flipped_colour(), Rotation.UP)
51             piece_instance.set_geometry((0, 0), piece_width)
52             piece_instance.set_image()
53             piece_list.append((piece_instance.image, (piece_width * index, (self.
54 _piece_surface.height - piece_width) / 2)))
55
56         self._piece_surface.blit(piece_list)
57
58         self.set_image()
59
60     def set_image(self):
61         self.image = create_text_box(self.size, self.border_width, self.
62 _box_colours)
63
64         resized_piece_surface = scale_and_cache(self._piece_surface, (self.size[0]
65 - 2 * self.margin, self.size[1] - 2 * self.margin))
66         self.image.blit(resized_piece_surface, (self.margin, self.margin))
67
68     def process_event(self, event):
69         pass

```

1.25.18 reactive_button.py

See Section ??.

1.25.19 reactive_icon_button.py

See Section ??.

1.25.20 rectangle.py

```

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

```

```

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

```

1.25.21 scrollbar.py

```

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

```

```

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

```

1.25.22 scroll_area.py

```

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

```

```

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

```

```

78         else:
79             widget_width = self._widget.rect.width
80
81             if widget_width < self.size[0]:
82                 return 0
83
84             if scrollbar:
85                 self._scroll_percentage += offset / (self.size[0] - self.
scrollbar_size[0] + 0.001)
86             else:
87                 max_scroll_width = widget_width - self.size[0]
88                 current_scroll_width = self._scroll_percentage * max_scroll_width
89                 self._scroll_percentage = (current_scroll_width + offset) /
max_scroll_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
166                             self._scroll_percentage = self.calculate_scroll_percentage(
167                             self.scroll_factor)
168                             self.set_image()
169                             return
170
171                 widget_event = self._widget.process_event(event, scrolled_pos=self.
172                 calculate_widget_position())
173                 if widget_event is not None:
174                     self.set_image()
175                 return widget_event

```

1.25.23 slider_thumb.py

```

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

```

```

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

```

1.25.24 switch.py

```

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

```

```

22         self._background_colour = None
23
24         self._is_toggled = None
25         self.set_toggle_state(False)
26
27         self.initialise_new_colours(self._fill_colour)
28         self.set_state_colour(WidgetState.BASE)
29
30         self.set_image()
31         self.set_geometry()
32
33     def hover_func(self):
34         self.set_state_colour(WidgetState.HOVER)
35
36     def set_toggle_state(self, is_toggled):
37         self._is_toggled = is_toggled
38         if is_toggled:
39             self._background_colour = self._on_colour
40         else:
41             self._background_colour = self._off_colour
42
43         self.set_image()
44
45     def up_func(self):
46         if self.get_widget_state() == WidgetState.PRESS:
47             toggle_state = not(self._is_toggled)
48             self.set_toggle_state(toggle_state)
49
50         self.set_state_colour(WidgetState.BASE)
51
52     def draw_thumb(self):
53         margin = self.size[1] * 0.1
54         thumb_radius = (self.size[1] / 2) - margin
55
56         if self._is_toggled:
57             thumb_center = (self.size[0] - margin - thumb_radius, self.size[1] /
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)

```

1.25.25 text.py

```

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

```

```

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

```

```

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

```

1.25.26 text_button.py

```

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

```

1.25.27 text_input.py

See Section ??.

1.25.28 timer.py

```

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

```

```

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

```

1.25.29 volume_slider.py

```

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

```

```

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

```

```

59
60         thumb_position = self.calculate_thumb_position()
61         self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
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)

```

1.25.30 __init__.py

```

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

```



```

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

```

1.26 data\widgets\bases

1.26.1 box.py

```

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

```

1.26.2 circular.py

See Section ??.

1.26.3 pressable.py

```

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

```

```

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()

```

1.26.4 widget.py

See Section ??.