Chapter 1

Source Code

This appendix contains all source code not show in the technical solution chapter.

1.1 data

1.1.1 control.py

```
import pygame
from data.components.widget_group import WidgetGroup
from data.managers.logs import initialise_logger
from data.managers.cursor import CursorManager
from data.managers.animation import animation
from data.utils.assets import DEFAULT_FONT
from data.managers.window import window
from data.managers.audio import audio
from data.managers.theme import theme

logger = initialise_logger(__file__)

FPS = 60
SHOW_FPS = False
start_ticks = pygame.time.get_ticks()

# Control class for managing state machine
class Control:
    def __init__(self):
        self.done = False
        self._clock = pygame.time.Clock()

    def setup_states(self, state_dict, start_state):
        self.state_dict = state_dict
        self.state_name = start_state

        self.state = self.state_dict[self.state_name]
        self.state.startup()

    # Method to cleanup previous state and startup new state.
    def flip_state(self):
        self.state.done = False
        persist = self.state.cleanup()

        previous, self.state_name = self.state_name, self.state.next
```

```
36
37          self.state = self.state_dict[self.state_name]
38          self.state.previous = previous
39          self.state.startup(persist)
40
41      def update(self):
42          if self.state.quit:
43              self.done = True
44          elif self.state.done:
45              self.flip_state()
46
47          self._clock.tick(FPS)
48          animation.set_delta_time()
49
50          self.state.update()
51
52          if SHOW_FPS:
53              self.draw_fps()
54
55          window.update()
56
57      def main_game_loop(self):
58          while not self.done:
59              self.event_loop()
60              self.update()
61
62      def update_window(self, resize=False):
63          if resize:
64              self.update_native_window_size()
65              window.handle_resize()
66              self.state.handle_resize()
67
68          self.update()
69
70      # Debug method to render framerate.
71      def draw_fps(self):
72          fps = str(int(self._clock.get_fps()))
73          DEFAULT_FONT.strength = 0.1
74          DEFAULT_FONT.render_to(window.screen, (0, 0), fps, fgcolor=theme['
    textError'], size=15)
75
76      # Used to limit window dimensions when resizing application window
77      def update_native_window_size(self):
78          x, y = window.size
79
80          max_window_x = 100000
81          max_window_y = x / 1.4
82          min_window_x = 400
83          min_window_y = min_window_x / 1.4
84
85          # If aspect ratio is less than 1.4, stop allowing width rescaling
86          if x / y < 1.4:
87              min_window_x = x
88
89          window.minimum_size = (min_window_x, min_window_y)
90          window.maximum_size = (max_window_x, max_window_y)
91
92      def event_loop(self):
93          for event in pygame.event.get():
94              if event.type == pygame.QUIT:
95                  self.done = True
96
```

```
            # Only allow left-click for mouse presses
            if event.type == pygame.MOUSEBUTTONDOWN and event.button != 1:
                return

            self.state.get_event(event)

class _State:
    def __init__(self):
        self.next = None
        self.previous = None
        self.done = False
        self.quit = False
        self.persist = {}

        self._cursor = CursorManager()
        self._widget_group = None

    def startup(self, widgets=None, music=None):
        if widgets:
            self._widget_group = WidgetGroup(widgets)
            self._widget_group.handle_resize(window.size)

        if music:
            audio.play_music(music)

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

    def cleanup(self):
        logger.info(f'cleaning {self.__class__.__name__.lower()}.py')

    def draw(self):
        raise NotImplementedError

    def get_event(self, event):
        raise NotImplementedError

    def handle_resize(self):
        self._widget_group.handle_resize(window.size)

    def update(self, **kwargs):
        self.draw()
```

1.1.2 loading_screen.py

See Section ??.

1.1.3 main.py

See Section ??.

1.1.4 setup.py

```
import pygame

# Initialise Pygame
pygame.mixer.init()
pygame.init()

# Initialise OpenGL for Pygame with version 330
pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MAJOR_VERSION, 3)
```

```
pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MINOR_VERSION, 3)
pygame.display.gl_set_attribute(pygame.GL_CONTEXT_PROFILE_MASK, pygame.
    GL_CONTEXT_PROFILE_CORE)
pygame.display.gl_set_attribute(pygame.GL_CONTEXT_FORWARD_COMPATIBLE_FLAG, True)
```

1.1.5 windows_setup.py

```
import win32gui
import win32con
import ctypes
import sys
import os

def wndProc(oldWndProc, draw_callback, hWnd, message, wParam, lParam):
    # Run window update function whenever window is being resized
    if message == win32con.WM_SIZING or message == win32con.WM_TIMER:
        draw_callback(resize=True)
        win32gui.RedrawWindow(hWnd, None, None, win32con.RDW_INVALIDATE | win32con
    .RDW_ERASE)
    # Run window update function whenever window is being dragged
    elif message == win32con.WM_MOVE:
        draw_callback(resize=False)

    return win32gui.CallWindowProc(oldWndProc, hWnd, message, wParam, lParam)

def set_win_resize_func(resize_function):
    oldWndProc = win32gui.SetWindowLong(win32gui.GetForegroundWindow(), win32con.
    GWL_WNDPROC, lambda *args: wndProc(oldWndProc, resize_function, *args))

user32 = ctypes.windll.user32
user32.SetProcessDPIAware() # To deal with Windows High Text Size / Low Display
    Resolution Settings

# Only allow Windows Version >= 7
if os.name != 'nt' or sys.getwindowsversion()[0] < 6:
    raise NotImplementedError("Incompatible OS!")
```

1.2 data\app_data

1.2.1 default_settings.json

```
{
    "primaryBoardColour": "0xB98766",
    "secondaryBoardColour": "0xF3D8B8",
    "laserColourBlue": "0x0000ff",
    "laserColourRed": "0xff0000",
    "displayMode": "windowed",
    "musicVolume": 0.5,
    "sfxVolume": 0.5,
    "particles": true,
    "opengl": true,
    "shader": "default"
}
```

1.2.2 logs_config.json

```
{
    "version": 1,
    "disable_existing_loggers": false,
```

```
    "formatters": {
      "simple": {
        "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
        "datefmt": "%Y-%m-%d %H:%M:%S"
      }
    },

    "handlers": {
      "console": {
        "class": "logging.StreamHandler",
        "formatter": "simple",
        "stream": "ext://sys.stdout"
      }
    },

    "root": {
      "level": "INFO",
      "handlers": ["console"],
      "propagate": false
    }
  }
```

1.2.3 logs_config_prod.json

```
{
    "version": 1,
    "disable_existing_loggers": false,
    "formatters": {
      "simple": {
        "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
      }
    },

    "handlers": {
      "console": {
        "class": "logging.StreamHandler",
        "level": "DEBUG",
        "formatter": "simple",
        "stream": "ext://sys.stdout"
      },

      "info_file_handler": {
        "class": "logging.handlers.RotatingFileHandler",
        "level": "INFO",
        "formatter": "simple",
        "filename": "info.log",
        "maxBytes": 10485760,
        "backupCount": 20,
        "encoding": "utf8"
      },

      "error_file_handler": {
        "class": "logging.handlers.RotatingFileHandler",
        "level": "ERROR",
        "formatter": "simple",
        "filename": "errors.log",
        "maxBytes": 10485760,
        "backupCount": 20,
        "encoding": "utf8"
      }
    },

```

```
    "loggers": {
      "my_module": {
        "level": "ERROR",
        "handlers": ["console"],
        "propagate": false
      }
    },

    "root": {
      "level": "INFO",
      "handlers": ["console", "info_file_handler", "error_file_handler"]
    }
  }
```

1.2.4 themes.json

```
{
    "colours": {
        "text": {
            "primary": "0xdaf2e9",
            "secondary": "0xf14e52",
            "error": "0xf14e52"
        },
        "fill": {
            "primary": "0x1c2638",
            "secondary": "0xf14e52",
            "tertiary": "0xdaf2e9",
            "error": "0x9b222b"
        },
        "border": {
            "primary": "0x9b222b",
            "secondary": "0xdaf2e9"
        }
    },
    "dimensions": {
        "borderRadius": 3,
        "borderWidth": 5,
        "margin": 10
    }
}
```

1.2.5 user_settings.json

```
{
    "primaryBoardColour": "0xB98766",
    "secondaryBoardColour": "0xF3D8B8",
    "laserColourBlue": "0x0000ff",
    "laserColourRed": "0xff0000",
    "displayMode": "windowed",
    "musicVolume": 0.03,
    "sfxVolume": 0.08,
    "particles": true,
    "opengl": true,
    "shader": "default"
}
```

1.3 data\components

1.3.1 circular_linked_list.py

See Section ??.

1.3.2 cursor.py

```
import pygame

class Cursor(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.Surface((1, 1))
        self.image.fill((255, 0, 0))
        self.rect = self.image.get_rect()

    # def update(self):
    #     self.rect.center = pygame.mouse.get_pos()

    def get_sprite_collision(self, mouse_pos, square_group):
        self.rect.center = mouse_pos
        sprite = pygame.sprite.spritecollideany(self, square_group)

        return sprite
```

1.3.3 custom_event.py

See Section ??.

1.3.4 game_entry.py

```
from data.states.game.components.move import Move
from data.utils.enums import Colour

class GameEntry:
    def __init__(self, game_states, final_fen_string):
        self._game_states = game_states
        self._final_fen_string = final_fen_string

    # Debug method used to print GameEntry row
    def __str__(self):
        return f'''
<GameEntry> :>
    CPU_ENABLED: {self._game_states['CPU_ENABLED']}
    CPU_DEPTH: {self._game_states['CPU_DEPTH']},
    WINNER: {self._game_states['WINNER']},
    TIME_ENABLED: {self._game_states['TIME_ENABLED']},
    TIME: {self._game_states['TIME']},
    NUMBER_OF_PLY: {len(self._game_states['MOVES'])},
    MOVES: {self.convert_moves(self._game_states['MOVES'])}
    FINAL FEN_STRING: {self._final_fen_string}
    START FEN STRING: {self._game_states['START_FEN_STRING']}
</GameEntry>
        '''

    def convert_to_row(self):
        return (self._game_states['CPU_ENABLED'], self._game_states['CPU_DEPTH'],
    self._game_states['WINNER'], self._game_states['TIME_ENABLED'], self.
    _game_states['TIME'], len(self._game_states['MOVES']), self.convert_moves(self
    ._game_states['MOVES']), self._game_states['START_FEN_STRING'], self.
    _final_fen_string)

    # List comprehension used to format move dictionary into string
    def convert_moves(self, moves):
        return '|'.join([
```

```
            f'{round(move['time'][Colour.BLUE], 4)};{round(move['time'][Colour.RED], 4)};{move['move']}'
            for move in moves
        ])

    # Inverse method of convert_moves, converts string into dictionary of moves
    @staticmethod
    def parse_moves(move_str):
        moves = move_str.split('|')
        return [
            {
                'blue_time': move.split(';')[0],
                'red_time': move.split(';')[1],
                'move': Move.instance_from_notation(move.split(';')[2]),
                'unparsed_move': move.split(';')[2],
            } for move in moves if move != ''
        ]
```

1.3.5 widget_group.py

```
import pygame
from data.managers.window import window

# Overriding Pygame widget group to handle own widget system
class WidgetGroup(pygame.sprite.Group):
    def __init__(self, widget_dict):
        super().__init__()

        # Add widgets from WIDGET_DICT
        for value in widget_dict.values():
            if isinstance(value, list):
                for widget in value:
                    self.add(widget)
            elif isinstance(value, dict):
                for widget in value.values():
                    self.add(widget)
            else:
                self.add(value)

    def handle_resize(self, new_surface_size):
        for sprite in self.sprites():
            sprite.set_surface_size(new_surface_size)
            sprite.set_image()
            sprite.set_geometry()

    def process_event(self, event):
        for sprite in self.sprites():
            widget_event = sprite.process_event(event)

            if widget_event:
                return widget_event

        return None

    def draw(self):
        sprites = self.sprites()
        for spr in sprites:
            surface = spr._surface or window.screen
            self.spritedict[spr] = surface.blit(spr.image, spr.rect)
        self.lostsprites = []
        dirty = self.lostsprites

```

```
        return dirty

    # Returns True if mouse cursor is hovering over a widget
    def on_widget(self, mouse_pos):
        test_sprite = pygame.sprite.Sprite()
        test_sprite.rect = pygame.FRect(*mouse_pos, 1, 1)
        return pygame.sprite.spritecollideany(test_sprite, self)
```

1.4 data\database

1.5 data\database\migrations

1.5.1 add_created_dt_column27112024.py

```
import sqlite3
from pathlib import Path

database_path = (Path(__file__).parent / '../database.db').resolve()

# Upgrade function used to update games table schema
def upgrade():
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute('''
        ALTER TABLE games ADD COLUMN created_dt TIMESTAMP NOT NULL
    ''')

    connection.commit()
    connection.close()

# Downgrade function used to revert changes
def downgrade():
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute('''
        ALTER TABLE games DROP COLUMN created_dt
    ''')

    connection.commit()
    connection.close()

upgrade()
# downgrade()
```

1.5.2 add_fen_string_column_22112024.py

```
import sqlite3
from pathlib import Path

database_path = (Path(__file__).parent / '../database.db').resolve()

def upgrade():
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute('''
        ALTER TABLE games ADD COLUMN fen_string TEXT NOT NULL
```

```
    ''')

    connection.commit()
    connection.close()

def downgrade():
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute('''
        ALTER TABLE games DROP COLUMN fen_string
    ''')

    connection.commit()
    connection.close()

upgrade()
```

1.5.3 add_start_fen_string_column_23122024.py

```
import sqlite3
from pathlib import Path

database_path = (Path(__file__).parent / '../database.db').resolve()

def upgrade():
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute('''
        ALTER TABLE games ADD COLUMN start_fen_string TEXT NOT NULL
    ''')

    connection.commit()
    connection.close()

def downgrade():
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute('''
        ALTER TABLE games DROP COLUMN start_fen_string
    ''')

    connection.commit()
    connection.close()

upgrade()
# 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

```python
from data.managers.logs import initialise_logger
from data.utils.constants import EMPTY_BB
from data.utils.enums import Rank, File

logger = initialise_logger(__name__)

# Debug function to return string representation of bitboard
def print_bitboard(bitboard):
    if (bitboard >= (2 ** 80)):
        raise ValueError('Invalid bitboard: too many bits')

    characters = ''
    for rank in reversed(Rank):

        for file in File:
            mask = 1 << (rank * 10 + file)
            if (bitboard & mask) != 0:
                characters += '1  '
            else:
                characters += '.  '

        characters += '\n\n'

    logger.info('\n' + characters + '\n')

def is_occupied(bitboard, target_bitboard):
    return (target_bitboard & bitboard) != EMPTY_BB

def clear_square(bitboard, target_bitboard):
    return (~target_bitboard & bitboard)

def set_square(bitboard, target_bitboard):
    return (target_bitboard | bitboard)

def index_to_bitboard(index):
    return (1 << index)

def coords_to_bitboard(coords):
    index = coords[1] * 10 + coords[0]
    return index_to_bitboard(index)

# Converts bitboard square to algebraic board notation
def bitboard_to_notation(bitboard):
    index = bitboard_to_index(bitboard)
    x = index // 10
    y = index % 10

    return chr(y + 97) + str(x + 1)

def notation_to_bitboard(notation):
    index = (int(notation[1]) - 1) * 10 + int(ord(notation[0])) - 97

```

```
    return index_to_bitboard(index)

def bitboard_to_index(bitboard):
    return bitboard.bit_length() - 1

def bitboard_to_coords(bitboard):
    list_position = bitboard_to_index(bitboard)
    x = list_position % 10
    y = list_position // 10

    return x, y

# Converts every occupied bit in bitboard to tuple of integers in a list
def bitboard_to_coords_list(bitboard):
    list_positions = []

    for square in occupied_squares(bitboard):
        list_positions.append(bitboard_to_coords(square))

    return list_positions

# Yields all individual occupied squares in the form of a bitboard
def occupied_squares(bitboard):
    while bitboard:
        lsb_square = bitboard & -bitboard
        bitboard = bitboard ^ lsb_square

        yield lsb_square

# Returns number of occupied squares in bitboard
def pop_count(bitboard):
    count = 0
    while bitboard:
        count += 1
        # Find least significant occupied bit
        lsb_square = bitboard & -bitboard
        bitboard = bitboard ^ lsb_square

    return count

def loop_all_squares():
    for i in range(80):
        yield 1 << i
```

1.6.3 board_helpers.py

```
import pygame
from data.helpers.data_helpers import get_user_settings
from data.utils.assets import DEFAULT_FONT

user_settings = get_user_settings()

def create_board(board_size, primary_colour, secondary_colour, font=DEFAULT_FONT):
    square_size = board_size[0] / 10
    board_surface = pygame.Surface(board_size)

    for i in range(80):
        x = i % 10
        y = i // 10

        if (x + y) % 2 == 0:
            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,
    square_size + 1, square_size + 1)) # +1 to fill in black lines
24
25          if y == 7:
26              text_position = (square_x + square_size * 0.7, square_y + square_size
    * 0.55)
27              text_size = square_size / 3
28              font.render_to(board_surface, text_position, str(chr(x + 1 + 96)),
    fgcolor=(10, 10, 10, 175), size=text_size)
29          if x == 0:
30              text_position = (square_x + square_size * 0.1, square_y + square_size
    * 0.1)
31              text_size = square_size / 3
32              font.render_to(board_surface, text_position, str(7-y + 1), fgcolor
    =(10, 10, 10, 175), size=text_size)
33
34      return board_surface
35
36  def create_square_overlay(square_size, colour):
37      overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
38      overlay.fill(colour)
39
40      return overlay
41
42  def create_circle_overlay(square_size, colour):
43      overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
44      pygame.draw.circle(overlay, colour, (square_size / 2, square_size / 2),
    square_size / 4)
45
46      return overlay
47
48  def coords_to_screen_pos(coords, board_position, square_size):
49      x = board_position[0] + (coords[0] * square_size)
50      y = board_position[1] + ((7 - coords[1]) * square_size)
51
52      return (x, y)
53
54  def screen_pos_to_coords(mouse_position, board_position, board_size):
55      if (board_position[0] <= mouse_position[0] <= board_position[0] + board_size
    [0]) and (board_position[1] <= mouse_position[1] <= board_position[1] +
    board_size[1]):
56          x = (mouse_position[0] - board_position[0]) // (board_size[0] / 10)
57          y = (board_size[1] - (mouse_position[1] - board_position[1])) // (
    board_size[0] / 10)
58          return (int(x), int(y))
59
60      return None
```

1.6.4 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:
```

```
        return 'DRAW'
    else:
        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

```
def height_to_font_size(font, target_height):
    test_size = 1
    while True:
        glyph_metrics = font.get_metrics('j', size=test_size)
        descender = font.get_sized_descender(test_size)
        test_height = abs(glyph_metrics[0][3] - glyph_metrics[0][2]) - descender
        if test_height > target_height:
            return test_size - 1

        test_size += 1

def width_to_font_size(font, target_width):
    test_size = 1
    while True:
        glyph_metrics = font.get_metrics(' ', size=test_size)

        if (glyph_metrics[0][4] * 8) > target_width:
            return (test_size - 1)

        test_size += 1

def text_width_to_font_size(text, font, target_width):
    test_size = 1
    if len(text) == 0:
        # print('(text_width_to_font_size) Text must have length greater than 1!')
        text = " "

    while True:
        text_rect = font.get_rect(text, size=test_size)

        if text_rect.width > target_width:
            return (test_size - 1)

        test_size += 1

def text_height_to_font_size(text, font, target_height):
    test_size = 1

    if ('(' in text) or (')' in text):
        text = text.replace('(', 'j') # Pygame freetype thinks '(' or ')' is
taller for some reason
        text = text.replace(')', 'j')

    if len(text) == 0:
        # print('(text_height_to_font_size) Text must have length greater than
1!')
```

```
        text = "j"

    while True:
        text_rect = font.get_rect(text, size=test_size)

        if text_rect.height > target_height:
            return (test_size - 1)

        test_size += 1

def get_font_height(font, font_size):
    glyph_metrics = font.get_metrics('j', size=font_size)
    descender = font.get_sized_descender(font_size)
    return abs(glyph_metrics[0][3] - glyph_metrics[0][2]) - descender
```

1.6.8 input_helpers.py

```
from data.utils.enums import MoveType, Rotation

def parse_move_type(move_type):
    if move_type.isalpha() is False:
        raise ValueError('Invalid move type - move type must be a string!')
    if move_type.lower() not in MoveType:
        raise ValueError('Invalid move - type - move type must be m or r!')

    return MoveType(move_type.lower())

def parse_notation(notation):
    if (notation[0].isalpha() is False) or (notation[1].isnumeric() is False):
        raise ValueError('Invalid notation - invalid notation input types!')
    if not (97 <= ord(notation[0]) <= 106):
        raise ValueError('Invalid notation - file is out of range!')
    elif not (0 <= int(notation[1]) <= 10):
        raise ValueError('Invalid notation - rank is out of range!')

    return notation

def parse_rotation(rotation):
    if rotation == '':
        return None
    if rotation.isalpha() is False:
        raise ValueError('Invalid rotation - rotation must be a string!')
    if rotation.lower() not in Rotation:
        raise ValueError('Invalid rotation - rotation is invalid!')

    return Rotation(rotation.lower())
```

1.6.9 load_helpers.py

```
import pygame
import pygame.freetype
from pathlib import Path
from data.helpers.asset_helpers import gif_to_frames, pil_image_to_surface

def convert_gfx_alpha(image, colorkey=(0, 0, 0)):
    # if image.get_alpha():
        return image.convert_alpha()
    # else:
    #     image = image.convert_alpha()
    #     image.set_colorkey(colorkey)

```

```
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',
         'pyramid_0', 'scarab_0', 'anubis_0', 'pharaoh_0', 'sphinx_0'],
53                      ['_a', '_b', '_c', '_d'])
54
55                  graphics = graphics | data
56                  continue
57
58              data = load_gfx(path, colorkey, accept)
59
60              if isinstance(data, list):
61                  graphics[name] = data[0]
62                  graphics[f'{name}_lq'] = data[1]
63              else:
64                  graphics[name] = data
65
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 *
    sprite_size[1], *sprite_size))
81              sprite_dict[name] = surface
82
83      return sprite_dict
84
85  def load_all_fonts(directory, accept=(".ttf", ".otf")):
86      fonts = {}
87
88      for file in Path(directory).rglob('*'):
89          name, extension = file.stem, file.suffix
90          path = Path(directory / file)
91
92          if extension.lower() in accept:
93              font = pygame.freetype.Font(path)
94              fonts[name] = font
95
96      return fonts
97
98  def load_all_sfx(directory, accept=(".mp3", ".wav", ".ogg")):
99      sound_effects = {}
100
101     for file in Path(directory).rglob('*'):
102         name, extension = file.stem, file.suffix
103         path = Path(directory / file)
104
105         if extension.lower() in accept and 'old' not in name:
106             sound_effects[name] = load_sfx(path)
107
108     return sound_effects
109
110 def load_sfx(path, accept=(".mp3", ".wav", ".ogg")):
111     file_path = Path(path)
112     name, extension = file_path.stem, file_path.suffix
113
114     if extension.lower() in accept:
115         sfx = pygame.mixer.Sound(path)
116         return sfx
117
118 def load_all_music(directory, accept=(".mp3", ".wav", ".ogg")):
119     music_paths = {}
120     for file in Path(directory).rglob('*'):
121         name, extension = file.stem, file.suffix
122         path = Path(directory / file)
123
124         if extension.lower() in accept:
125             music_paths[name] = path
126
127     return music_paths
```

1.6.10 widget_helpers.py

See Section ??.

1.7 data\managers

1.7.1 animation.py

```
import pygame
from data.helpers.asset_helpers import scale_and_cache

FPS = 60

class AnimationManager:
    def __init__(self):
        self._current_ms = 0
        self._timers = []

    def set_delta_time(self):
        self._current_ms = pygame.time.get_ticks()

        for timer in self._timers:
            start_ms, target_ms, callback = timer
            if self._current_ms - start_ms >= target_ms:
                callback()
                self._timers.remove(timer)

    def calculate_frame_index(self, start_index, end_index, fps):
        ms_per_frame = int(1000 / fps)
        return start_index + ((self._current_ms // ms_per_frame) % (end_index -
    start_index))

    def draw_animation(self, screen, animation, position, size, fps=8):
        frame_index = self.calculate_frame_index(0, len(animation), fps)
        scaled_animation = scale_and_cache(animation[frame_index], size)
        screen.blit(scaled_animation, position)

    def draw_image(self, screen, image, position, size):
        scaled_background = scale_and_cache(image, size)
        screen.blit(scaled_background, position)

    def set_timer(self, target_ms, callback):
        self._timers.append((self._current_ms, target_ms, callback))

animation = AnimationManager()
```

1.7.2 audio.py

```
import pygame
from data.helpers.data_helpers import get_user_settings
from data.managers.logs import initialise_logger

logger = initialise_logger(__name__)
user_settings = get_user_settings()

class AudioManager:
    def __init__(self, num_channels=16):
        pygame.mixer.set_num_channels(num_channels)

        self._music_volume = user_settings['musicVolume']
        self._sfx_volume = user_settings['sfxVolume']

        self._current_song = None
        self._current_channels = []

```

```
18      def set_sfx_volume(self, volume):
19          self._sfx_volume = volume
20
21          for channel in self._current_channels:
22              channel.set_volume(self._sfx_volume)
23
24      def set_music_volume(self, volume):
25          self._music_volume = volume
26
27          pygame.mixer.music.set_volume(self._music_volume)
28
29      def pause_sfx(self):
30          pygame.mixer.pause()
31
32      def unpause_sfx(self):
33          pygame.mixer.unpause()
34
35      def stop_sfx(self, fadeout=0):
36          pygame.mixer.fadeout(fadeout)
37
38      def remove_unused_channels(self):
39          unused_channels = []
40          for channel in self._current_channels:
41              if channel.get_busy() is False:
42                  unused_channels.append(channel)
43
44          return unused_channels
45
46      def play_sfx(self, sfx, loop=False):
47          unused_channels = self.remove_unused_channels()
48
49          if len(unused_channels) == 0:
50              channel = pygame.mixer.find_channel()
51          else:
52              channel = unused_channels.pop(0)
53
54          if channel is None:
55              logger.warning('No available channel for SFX')
56              return
57
58          self._current_channels.append(channel)
59          channel.set_volume(self._sfx_volume)
60
61          if loop:
62              channel.play(sfx, loops=-1)
63          else:
64              channel.play(sfx)
65
66      def play_music(self, music_path):
67          if 'menu' in str(music_path) and 'menu' in str(self._current_song):
68              return
69
70          if music_path == self._current_song:
71              return
72
73          pygame.mixer.music.stop()
74          pygame.mixer.music.unload()
75          pygame.mixer.music.load(music_path)
76          pygame.mixer.music.set_volume(self._music_volume)
77          pygame.mixer.music.play(loops=-1)
78
79          self._current_song = music_path
```

```

audio = AudioManager()
```

1.7.3 cursor.py

```
import pygame
from data.utils.enums import CursorMode
from data.utils.assets import GRAPHICS

# Manager to change mouse cursor iconss
class CursorManager:
    def __init__(self):
        self._mode = CursorMode.ARROW
        self.set_mode(CursorMode.ARROW)

    def set_mode(self, mode):
        pygame.mouse.set_visible(True)

        match mode:
            case CursorMode.ARROW:
                pygame.mouse.set_cursor((7, 5), pygame.transform.scale(GRAPHICS['arrow'], (32, 32)))
            case CursorMode.IBEAM:
                pygame.mouse.set_cursor((15, 5), pygame.transform.scale(GRAPHICS['ibeam'], (32, 32)))
            case CursorMode.OPENHAND:
                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['hand_open'], (32, 32)))
            case CursorMode.CLOSEDHAND:
                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['hand_closed'], (32, 32)))
            case CursorMode.NO:
                pygame.mouse.set_visible(False)

        self._mode = mode

    def get_mode(self):
        return self._mode

cursor = CursorManager()
```

1.7.4 logs.py

```
import logging.config
from data.helpers.data_helpers import load_json
from pathlib import Path
import logging

config_path = (Path(__file__).parent / '../app_data/logs_config.json').resolve()
config = load_json(config_path)
logging.config.dictConfig(config)

def initialise_logger(file_path):
    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

```
import pygame
import moderngl
from data.utils.constants import ShaderType, SCREEN_SIZE, SHADER_MAP
from data.helpers.data_helpers import get_user_settings
from data.helpers.asset_helpers import draw_background
from data.managers.shader import ShaderManager

user_settings = get_user_settings()
is_opengl = user_settings['opengl']
is_fullscreen = user_settings['displayMode'] == 'fullscreen'

class WindowManager(pygame.Window):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self._native_screen = self.get_surface() # Initialise convert format
        self.screen = pygame.Surface(self.size, pygame.SRCALPHA)

        if is_opengl:
            self._ctx = moderngl.create_context()
            self._shader_manager = ShaderManager(self._ctx, screen_size=self.size)

            # Each ShaderType contains a dictionary of kwargs, used as arguments when running the apply method on the corresponding shader class
            self.shader_arguments = {
                ShaderType.BASE: {},
                ShaderType.SHAKE: {},
                ShaderType.BLOOM: {},
                ShaderType.GRAYSCALE: {},
                ShaderType.CRT: {},
                ShaderType.RAYS: {}
            }

            # For the secret settings option in the settings menu, apply shaders for the selected option
            if (selected_shader := get_user_settings()['shader']) is not None:
                for shader_type in SHADER_MAP[selected_shader]:
                    self.set_effect(shader_type)
        else:
            # If shaders disabled, use temporary image as background
            from data.utils.assets import GRAPHICS
            self._background_image = GRAPHICS['temp_background']

    def set_effect(self, effect, **kwargs):
        if is_opengl:
            self._shader_manager.apply_shader(effect, **kwargs)

    def set_apply_arguments(self, effect, **kwargs):
        if is_opengl:
            self.shader_arguments[effect] = kwargs

    def clear_apply_arguments(self, effect):
        if is_opengl:
            self.shader_arguments[effect] = {}

    def clear_effect(self, effect):
```

```
        if is_opengl:
            self._shader_manager.remove_shader(effect)
            self.clear_apply_arguments(effect)

    def clear_all_effects(self, clear_arguments=False):
        if is_opengl:
            self._shader_manager.clear_shaders()

            if clear_arguments:
                for shader_type in self.shader_arguments:
                    self.shader_arguments[shader_type] = {}

    def draw(self):
        if is_opengl:
            self._shader_manager.draw(self.screen, self.shader_arguments)
        else:
            self._native_screen.blit(self.screen, (0, 0))

        self.flip()

        if is_opengl:
            self.screen.fill((0, 0, 0, 0))
        else:
            self.screen.fill((0, 0, 0))
            draw_background(self.screen, self._background_image)

    def update(self):
        self.draw()

    def handle_resize(self):
        self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
        if is_opengl:
            self._shader_manager.handle_resize(self.size)
        else:
            draw_background(self.screen, self._background_image)

window = WindowManager(size=SCREEN_SIZE, resizable=True, opengl=is_opengl,
    fullscreen_desktop=is_fullscreen)
```

1.8 data\shaders

1.8.1 protocol.py

```
import pygame
import moderngl
from typing import Protocol, Optional
from data.utils.constants import ShaderType

class SMProtocol(Protocol):
    def load_shader(self, shader_type: ShaderType, **kwargs) -> None: ...
    def clear_shaders(self) -> None: ...
    def create_vao(self, shader_type: ShaderType) -> None: ...
    def create_framebuffer(self, shader_type: ShaderType, size: Optional[tuple[int
    ]]=None, filter: Optional[int]=moderngl.NEAREST) -> None: ...
    def render_to_fbo(self, shader_type: ShaderType, texture: moderngl.Texture,
    output_fbo: Optional[moderngl.Framebuffer] = None, program_type: Optional[
    ShaderType] = None, use_image: Optional[bool] = True, **kwargs) -> None: ...
    def apply_shader(self, shader_type: ShaderType, **kwargs) -> None: ...
    def remove_shader(self, shader_type: ShaderType) -> None: ...
    def render_output(self, texture: moderngl.Texture) -> None: ...
    def get_fbo_texture(self, shader_type: ShaderType) -> moderngl.Texture: ...
```

```
    def calibrate_pygame_surface(self, pygame_surface: pygame.Surface) -> moderngl
    .Texture: ...
    def draw(self, surface: pygame.Surface, arguments: dict) -> None: ...
    def __del__(self) -> None: ...
    def cleanup(self) -> None: ...
    def handle_resize(self, new_screen_size: tuple[int]) -> None: ...

    _ctx: moderngl.Context
    _screen_size: tuple[int]
    _opengl_buffer: moderngl.Buffer
    _pygame_buffer: moderngl.Buffer
    _shader_stack: list[ShaderType]

    _vert_shaders: dict
    _frag_shaders: dict
    _programs: dict
    _vaos: dict
    _textures: dict
    _shader_passes: dict
    framebuffers: dict
```

1.9 data\shaders\classes

1.9.1 base.py

```
import pygame
from data.shaders.protocol import SMProtocol
from data.utils.constants import ShaderType

class Base:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        self._shader_manager.create_framebuffer(ShaderType.BASE)
        self._shader_manager.create_vao(ShaderType.BACKGROUND_WAVES)
        self._shader_manager.create_vao(ShaderType.BACKGROUND_BALATRO)
        self._shader_manager.create_vao(ShaderType.BACKGROUND_LASERS)
        self._shader_manager.create_vao(ShaderType.BACKGROUND_GRADIENT)
        self._shader_manager.create_vao(ShaderType.BACKGROUND_NONE)

    def apply(self, texture, background_type=None):
        base_texture = self._shader_manager.get_fbo_texture(ShaderType.BASE)

        # Draws background to ShaderType.BASE framebuffer
        match background_type:
            case ShaderType.BACKGROUND_WAVES:
                self._shader_manager.render_to_fbo(
                    ShaderType.BASE,
                    texture=base_texture,
                    program_type=ShaderType.BACKGROUND_WAVES,
                    use_image=False,
                    time=pygame.time.get_ticks() / 1000
                )
            case ShaderType.BACKGROUND_BALATRO:
                self._shader_manager.render_to_fbo(
                    ShaderType.BASE,
                    texture=base_texture,
                    program_type=ShaderType.BACKGROUND_BALATRO,
                    use_image=False,
                    time=pygame.time.get_ticks() / 1000,
                    screenSize=base_texture.size
```

```
                )
            case ShaderType.BACKGROUND_LASERS:
                self._shader_manager.render_to_fbo(
                    ShaderType.BASE,
                    texture=base_texture,
                    program_type=ShaderType.BACKGROUND_LASERS,
                    use_image=False,
                    time=pygame.time.get_ticks() / 1000,
                   screenSize=base_texture.size
                )
            case ShaderType.BACKGROUND_GRADIENT:
                self._shader_manager.render_to_fbo(
                    ShaderType.BASE,
                    texture=base_texture,
                    program_type=ShaderType.BACKGROUND_GRADIENT,
                    use_image=False,
                    time=pygame.time.get_ticks() / 1000,
                   screenSize=base_texture.size
                )
            case None:
                self._shader_manager.render_to_fbo(
                    ShaderType.BASE,
                    texture=base_texture,
                    program_type=ShaderType.BACKGROUND_NONE,
                    use_image=False,
                )
            case _:
                raise ValueError('(shader.py) Unknown background type:', background_type)

        # Draws background using texture in ShaderType.BASE framebuffer, on pixels in the Pygame texture that have no alpha
        self._shader_manager.get_fbo_texture(ShaderType.BASE).use(1)
        self._shader_manager.render_to_fbo(ShaderType.BASE, texture, background=1)
```

1.9.2 blend.py

```
import moderngl
from data.shaders.protocol import SMProtocol
from data.utils.constants import ShaderType

class _Blend:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        self._shader_manager.create_framebuffer(ShaderType._BLEND)

    # Blend two textures, while positioning textures relative to each other if not the same size
    def apply(self, texture, texture_2, texture_2_pos):
        self._shader_manager._ctx.blend_func = (moderngl.SRC_ALPHA, moderngl.ONE)

        relative_size = (texture_2.size[0] / texture.size[0], texture_2.size[1] / texture.size[1])
        # Convert position of smaller texture within big texture into OpenGL coordinates
        opengl_pos = (texture_2_pos[0], 1 - texture_2_pos[1] - relative_size[1])

        texture_2.use(1)
        self._shader_manager.render_to_fbo(ShaderType._BLEND, texture, image2=1, image2Pos=opengl_pos, relativeSize=relative_size)
        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

```
import pygame
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol

CHROMATIC_ABBREVIATION_INTENSITY = 2.0

class ChromaticAbbreviation:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        self._shader_manager.create_framebuffer(ShaderType.CHROMATIC_ABBREVIATION)

    def apply(self, texture):
        mouse_pos = (pygame.mouse.get_pos()[0] / texture.size[0], pygame.mouse.get_pos()[1] / texture.size[1])
        self._shader_manager.render_to_fbo(ShaderType.CHROMATIC_ABBREVIATION, texture, mouseFocusPoint=mouse_pos, enabled=pygame.mouse.get_pressed()[0], intensity=CHROMATIC_ABBREVIATION_INTENSITY)
```

1.9.6 crop.py

```
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol

class _Crop:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

    def apply(self, texture, relative_pos, relative_size):
        opengl_pos = (relative_pos[0], 1 - relative_pos[1] - relative_size[1])
        pixel_size = (int(relative_size[0] * texture.size[0]), int(relative_size[1] * texture.size[1]))

        self._shader_manager.create_framebuffer(ShaderType._CROP, size=pixel_size)

        self._shader_manager.render_to_fbo(ShaderType._CROP, texture, relativePos=opengl_pos, relativeSize=relative_size)
```

1.9.7 crt.py

```
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol

class CRT:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        shader_manager.create_framebuffer(ShaderType.CRT)

```

```
    def apply(self, texture):
        self._shader_manager.render_to_fbo(ShaderType.CRT, texture)
```

1.9.8 grayscale.py

```
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol

class Grayscale:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        shader_manager.create_framebuffer(ShaderType.GRAYSCALE)

    def apply(self, texture):
        self._shader_manager.render_to_fbo(ShaderType.GRAYSCALE, texture)
```

1.9.9 highlight_brightness.py

```
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol

HIGHLIGHT_THRESHOLD = 0.9

class _HighlightBrightness:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_BRIGHTNESS)

    def apply(self, texture, intensity):
        self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_BRIGHTNESS,
    texture, threshold=HIGHLIGHT_THRESHOLD, intensity=intensity)
```

1.9.10 highlight_colour.py

```
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol

class _HighlightColour:
    def __init__(self, shader_manager: SMProtocol):
        self._shader_manager = shader_manager

        shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_COLOUR)

    def apply(self, texture, old_highlight, colour, intensity):
        old_highlight.use(1)
        self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_COLOUR, texture,
    highlight=1, colour=colour, threshold=0.1, intensity=intensity)
```

1.9.11 lightmap.py

```
from data.utils.constants import ShaderType
from data.shaders.protocol import SMProtocol
from data.shaders.classes.shadowmap import _Shadowmap

LIGHT_RESOLUTION = 256

class _Lightmap:
    def __init__(self, shader_manager: SMProtocol):
```

```
9           self._shader_manager = shader_manager
10
11          shader_manager.load_shader(ShaderType._SHADOWMAP)
12
13      def apply(self, texture, colour, softShadow, occlusion=None, falloff=0.0,
    clamp=(-180, 180)):
14          self._shader_manager.create_framebuffer(ShaderType._LIGHTMAP, size=texture
    .size)
15          self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)
16
17          _Shadowmap(self._shader_manager).apply(texture, occlusion)
18          shadow_map = self._shader_manager.get_fbo_texture(ShaderType._SHADOWMAP)
19
20          self._shader_manager.render_to_fbo(ShaderType._LIGHTMAP, shadow_map,
    resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
    clamp, softShadow=softShadow)
21
22          self._shader_manager._ctx.disable(self._shader_manager._ctx.BLEND)
```

1.9.12 occlusion.py

```
1  from data.utils.constants import ShaderType
2  from data.shaders.protocol import SMProtocol
3
4  class _Occlusion:
5      def __init__(self, shader_manager: SMProtocol):
6          self._shader_manager = shader_manager
7
8      def apply(self, texture, occlusion_colour=(255, 0, 0)):
9          self._shader_manager.create_framebuffer(ShaderType._OCCLUSION, size=
    texture.size)
10         self._shader_manager.render_to_fbo(ShaderType._OCCLUSION, texture,
    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 SMProtocol
4  from data.shaders.classes.occlusion import _Occlusion
5
6  LIGHT_RESOLUTION = 256
7
8  class _Shadowmap:
9      def __init__(self, shader_manager: SMProtocol):
10         self._shader_manager = shader_manager
11
12         shader_manager.load_shader(ShaderType._OCCLUSION)
13
14     def apply(self, texture, occlusion_texture=None):
15         self._shader_manager.create_framebuffer(ShaderType._SHADOWMAP, size=(
    texture.size[0], 1), filter=moderngl.LINEAR)
16
17         if occlusion_texture is None:
18             _Occlusion(self._shader_manager).apply(texture)
19             occlusion_texture = self._shader_manager.get_fbo_texture(ShaderType.
    _OCCLUSION)
```

```
20
21          self._shader_manager.render_to_fbo(ShaderType._SHADOWMAP,
    occlusion_texture, resolution=LIGHT_RESOLUTION)
```

1.9.15 shake.py

```
1  from data.utils.constants import ShaderType
2  from data.shaders.protocol import SMProtocol
3  from random import randint
4
5  SHAKE_INTENSITY = 3
6
7  class Shake:
8      def __init__(self, shader_manager: SMProtocol):
9          self._shader_manager = shader_manager
10
11         self._shader_manager.create_framebuffer(ShaderType.SHAKE)
12
13     def apply(self, texture, intensity=SHAKE_INTENSITY):
14         displacement = (randint(-intensity, intensity) / 1000, randint(-intensity,
     intensity) / 1000)
15         self._shader_manager.render_to_fbo(ShaderType.SHAKE, texture, displacement
    =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
17 from data.utils.constants import ShaderType
18
19 shader_pass_lookup = {
20     ShaderType.CHROMATIC_ABBREVIATION: ChromaticAbbreviation,
21     ShaderType.GRAYSCALE: Grayscale,
22     ShaderType.SHAKE: Shake,
23     ShaderType.BLOOM: Bloom,
24     ShaderType.BASE: Base,
25     ShaderType.RAYS: Rays,
26     ShaderType.CRT: CRT,
27
28     ShaderType._HIGHLIGHT_BRIGHTNESS: _HighlightBrightness,
29     ShaderType._HIGHLIGHT_COLOUR: _HighlightColour,
30     ShaderType._CALIBRATE: lambda *args: None,
31     ShaderType._OCCLUSION: _Occlusion,
32     ShaderType._SHADOWMAP: _Shadowmap,
33     ShaderType._LIGHTMAP: _Lightmap,
34     ShaderType._BLEND: _Blend,
35     ShaderType._BLUR: _Blur,
```

```
    ShaderType._CROP: _Crop,
}
```

1.10 data\shaders\fragments

1.10.1 background_balatro.frag

```
// Original by localthunk (https://www.playbalatro.com)
// Modified from https://godotshaders.com/shader/balatro-background-shader/

# version 330 core

// Configuration (modify these values to change the effect)
#define SPIN_ROTATION -2.0
#define SPIN_SPEED 7.0
#define OFFSET vec2(0.0)
#define COLOUR_2 vec4(0.871, 0.267, 0.231, 1.0)
#define COLOUR_1 vec4(0.0, 0.42, 0.706, 1.0)
#define COLOUR_3 vec4(0.086, 0.137, 0.145, 1.0)
#define CONTRAST 3.5
#define LIGTHING 0.4
#define SPIN_AMOUNT 0.25
#define PIXEL_FILTER 745.0
#define SPIN_EASE 1.0
#define PI 3.14159265359
#define IS_ROTATE false

uniform float time;
uniform vec2 screenSize;

in vec2 uvs;
out vec4 f_colour;

vec4 effect(vec2 screenSize, vec2 screen_coords) {
    float pixel_size = length(screenSize.xy) / PIXEL_FILTER;
    vec2 uv = (floor(screen_coords.xy*(1./pixel_size))*pixel_size - 0.5*screenSize
    .xy)/length(screenSize.xy) - OFFSET;
    float uv_len = length(uv);

    float speed = (SPIN_ROTATION*SPIN_EASE*0.2);
    if(IS_ROTATE){
       speed = time * speed;
    }
    speed += 302.2;
    float new_pixel_angle = atan(uv.y, uv.x) + speed - SPIN_EASE*20.*(1.*
    SPIN_AMOUNT*uv_len + (1. - 1.*SPIN_AMOUNT));
    vec2 mid = (screenSize.xy/length(screenSize.xy))/2.;
    uv = (vec2((uv_len * cos(new_pixel_angle) + mid.x), (uv_len * sin(
    new_pixel_angle) + mid.y)) - mid);

    uv *= 30.;
    speed = time*(SPIN_SPEED);
    vec2 uv2 = vec2(uv.x+uv.y);

    for(int i=0; i < 5; i++) {
        uv2 += sin(max(uv.x, uv.y)) + uv;
        uv  += 0.5*vec2(cos(5.1123314 + 0.353*uv2.y + speed*0.131121),sin(uv2.x -
    0.113*speed));
        uv  -= 1.0*cos(uv.x + uv.y) - 1.0*sin(uv.x*0.711 - uv.y);
    }

```

```
51      float contrast_mod = (0.25*CONTRAST + 0.5*SPIN_AMOUNT + 1.2);
52      float paint_res = min(2., max(0.,length(uv)*(0.035)*contrast_mod));
53      float c1p = max(0.,1. - contrast_mod*abs(1.-paint_res));
54      float c2p = max(0.,1. - contrast_mod*abs(paint_res));
55      float c3p = 1. - min(1., c1p + c2p);
56      float light = (LIGTHING - 0.2)*max(c1p*5. - 4., 0.) + LIGTHING*max(c2p*5. -
        4., 0.);
57      return (0.3/CONTRAST)*COLOUR_1 + (1. - 0.3/CONTRAST)*(COLOUR_1*c1p + COLOUR_2*
        c2p + vec4(c3p*COLOUR_3.rgb, c3p*COLOUR_1.a)) + light;
58  }
59
60  void main() {
61      f_colour = effect(screenSize.xy, uvs* screenSize.xy);
62  }
```

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
        License.
22  vec2 hash( vec2 p )
23  {
24      p = vec2( dot(p,vec2(2127.1,81.17)), dot(p,vec2(1269.5,283.37)) );
25    return fract(sin(p)*43758.5453);
26  }
27
28  float noise( in vec2 p )
29  {
30      vec2 i = floor( p );
31      vec2 f = fract( p );
32
33    vec2 u = f*f*(3.0-2.0*f);
34
35      float n = mix( mix( dot( -1.0+2.0*hash( i + vec2(0.0,0.0) ), f - vec2(0.0,0.0)
         ),
36                          dot( -1.0+2.0*hash( i + vec2(1.0,0.0) ), f - vec2(1.0,0.0)
         ), u.x),
37                     mix( dot( -1.0+2.0*hash( i + vec2(0.0,1.0) ), f - vec2(0.0,1.0)
         ),
38                          dot( -1.0+2.0*hash( i + vec2(1.0,1.0) ), f - vec2(1.0,1.0)
         ), u.x), u.y);
39    return 0.5 + 0.5*n;
40  }
```

```

void main() {
    float ratio = screenSize.x / screenSize.y;

    vec2 tuv = uvs;
    tuv -= .5;

    // rotate with Noise
    float degree = noise(vec2(time*.1, tuv.x*tuv.y));

    tuv.y *= 1./ratio;
    tuv *= Rot(radians((degree-.5)*720.+180.));
  tuv.y *= ratio;

    // Wave warp with sin
    float frequency = 5.;
    float amplitude = 30.;
    float speed = time * 2.;
    tuv.x += sin(tuv.y*frequency+speed)/amplitude;
    tuv.y += sin(tuv.x*frequency*1.5+speed)/(amplitude*.5);

    // draw the image
    vec3 colorYellow = vec3(.957, .804, .623);
    vec3 colorDeepBlue = vec3(.192, .384, .933);
    vec3 layer1 = mix(colorYellow, colorDeepBlue, S(-.3, .2, (tuv*Rot(radians(-5.)
    )).x));

    vec3 colorRed = vec3(.910, .510, .8);
    vec3 colorBlue = vec3(0.350, .71, .953);
    vec3 layer2 = mix(colorRed, colorBlue, S(-.3, .2, (tuv*Rot(radians(-5.))).x));

    vec3 finalComp = mix(layer1, layer2, S(.5, -.3, tuv.y));

    vec3 col = finalComp;

    f_colour = vec4(col,1.0);
}
```

1.10.3 background_lasers.frag

```

// Modified from https://www.shadertoy.com/view/7tBSR1
// rand [0,1] https://www.shadertoy.com/view/4djSRW

#version 330 core

uniform float time;
uniform vec2 screenSize;

in vec2 uvs;
out vec4 f_colour;

float rand(vec2 p) {
    p *= 500.0;
  vec3 p3  = fract(vec3(p.xyx) * .1031);
    p3 += dot(p3, p3.yzx + 33.33);
    return fract((p3.x + p3.y) * p3.z);
}

// value noise
float noise(vec2 p) {
  vec2 f = smoothstep(0.0, 1.0, fract(p));
  vec2 i = floor(p);
```

```
  float a = rand(i);
  float b = rand(i+vec2(1.0,0.0));
  float c = rand(i+vec2(0.0,1.0));
  float d = rand(i+vec2(1.0,1.0));
  return mix(mix(a, b, f.x), mix(c, d, f.x), f.y);
}

// fractal noise
float fbm(vec2 p) {
    float a = 0.5;
    float r = 0.0;
    for (int i = 0; i < 8; i++) {
        r += a*noise(p);
        a *= 0.5;
        p *= 2.0;
    }
    return r;
}

// lasers originating from a central point
float laser(vec2 p, int num) {
  float r = atan(p.x, p.y);
  float sn = sin(r*float(num)+time);
    float lzr = 0.5+0.5*sn;
    lzr = lzr*lzr*lzr*lzr*lzr;
    float glow = pow(clamp(sn, 0.0, 1.0),100.0);
  return lzr+glow;
}

// mix of fractal noises to simulate fog
float clouds(vec2 uv) {
    vec2 t = vec2(0,time);
  float c1 = fbm(fbm(uv*3.0)*0.75+uv*3.0+t/3.0);
  float c2 = fbm(fbm(uv*2.0)*0.5+uv*7.0+t/3.0);
  float c3 = fbm(fbm(uv*10.0-t)*0.75+uv*5.0+t/6.0);
    float r = mix(c1, c2, c3*c3);
  return r*r;
}

void main() {
    vec2 hs = screenSize.xy/screenSize.y*0.5;
    vec2 uvc = uvs-hs;
  float l = (1.0 + 3.0*noise(vec2(15.0-time)))
        * laser(vec2(uvs.x+0.5, uvs.y*(0.5 + 10.0*noise(vec2(time/5.0))) + 0.1),
    15);
  l += fbm(vec2(2.0*time))
        * laser(vec2(hs.x-uvc.x-0.2, uvs.y+0.1), 25);
  l += noise(vec2(time-73.0))
        * laser(vec2(uvc.x, 1.0-uvs.y+0.5), 30);
    float c = clouds(uvs);
    vec4 col = vec4(uvs.x, 0.0, 1-uvs.x, 1.0)*(uvs.y*l+uvs.y*uvs.y)*c;

    f_colour = pow(col, vec4(0.75));
}
```

1.10.4 background_none.frag

```
# version 330 core

in vec2 uvs;
out vec4 f_colour;

```

```
void main() {
    f_colour = vec4(vec3(0.0 + uvs.x * 0.001), 1.0);
}
```

1.10.5 background_waves.frag

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

# version 330 core

uniform float wave_amp=1.0;
uniform float wave_size=4.0;
uniform float wave_time_mul=0.2;

uniform int total_phases=20;

uniform vec4 bottom_color=vec4(0.608, 0.133, 0.167, 1.0);
uniform vec4 top_color=vec4(0.110, 0.149, 0.220, 1.0);

// uniform vec4 bottom_color=vec4(0.38, 0.04, 0.71, 1.0);
// uniform vec4 top_color=vec4(0.15, 0.02, 0.49, 1.0);

uniform float time;

in vec2 uvs;
out vec4 f_colour;

#define PI 3.14159

float rand (float n) {
    return fract(sin(n) * 43758.5453123);
}
float noise (float p){
  float fl = floor(p);
    float fc = fract(p);
  return mix(rand(fl), rand(fl + 1.0), fc);
}
float fmod(float x, float y) {
  return x - floor(x / y) * y;
}
vec4 lerp(vec4 a, vec4 b, float w) {
  return a + w * (b - a);
}

void main() {
  float t = float(total_phases);
  float effective_wave_amp = min(wave_amp, 0.5 / t);
  float d = fmod(uvs.y, 1.0 / t);
  float i = floor(uvs.y * t);
  float vi = floor(uvs.y * t + t * effective_wave_amp);
  float s = effective_wave_amp * sin((uvs.x + time * max(1.0 / t, noise(vi)) *
    wave_time_mul * vi / t) * 2.0 * PI * wave_size);

  if (d < s) i--;
  if (d > s + 1.0 / t) i++;
  i = clamp(i, 0.0, t - 1.0);

  f_colour = lerp(top_color, bottom_color, i / (t - 1.0));
}
```

1.10.6 base.frag

```
#version 330 core

uniform sampler2D image;
uniform sampler2D background;

in vec2 uvs;
out vec4 f_colour;

void main() {
    vec4 colour = texture(image, uvs);

  if (colour.a == 1.0) {
    f_colour = colour;
  } else {
    f_colour = texture(background, uvs);
  }
}
```

1.10.7 blend.frag

```
#version 330 core

uniform sampler2D image;
uniform sampler2D image2;
uniform vec2 relativeSize;
uniform vec2 image2Pos;

in vec2 uvs;
out vec4 f_colour;

void main() {
    vec3 colour = texture(image, uvs).rgb;

    vec2 image2Coords = vec2((uvs.x - image2Pos.x) / relativeSize.x, (uvs.y -
    image2Pos.y) / relativeSize.y);

    float withinBounds = step(image2Pos.x, uvs.x) * step(uvs.x, (image2Pos.x +
    relativeSize.x)) * step(image2Pos.y, uvs.y) * step(uvs.y, (image2Pos.y +
    relativeSize.y));

    f_colour = vec4(colour + (texture(image2, image2Coords).rgb * withinBounds),
    1.0);
}
```

1.10.8 bloom.frag

```
#version 330 core

in vec2 uvs;
out vec4 f_colour;

uniform sampler2D image;
uniform sampler2D blurredImage;
uniform float intensity;

void main() {
    vec3 baseColour = texture(image, uvs).rgb;
    vec3 bloomColor = texture(blurredImage, uvs).rgb;

    baseColour += bloomColor * intensity;
    f_colour = vec4(baseColour, 1.0);
}
```

1.10.9 bloom_old.frag

```
#version 330 core

in vec2 uvs;
out vec4 f_colour;

uniform sampler2D image;
uniform float bloom_spread = 0.1;
uniform float bloom_intensity = 0.5;

void main() {
  ivec2 size = textureSize(image, 0);

    float uv_x = uvs.x * size.x;
    float uv_y = uvs.y * size.y;

    vec4 sum = vec4(0.0);

    for (int n = 0; n < 9; ++n) {
        uv_y = (uvs.y * size.y) + (bloom_spread * float(n - 4));
        vec4 h_sum = vec4(0.0);
        h_sum += texelFetch(image, ivec2(uv_x - (4.0 * bloom_spread), uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x - (3.0 * bloom_spread), uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x - (2.0 * bloom_spread), uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x - bloom_spread, uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x, uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x + bloom_spread, uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x + (2.0 * bloom_spread), uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x + (3.0 * bloom_spread), uv_y), 0);
        h_sum += texelFetch(image, ivec2(uv_x + (4.0 * bloom_spread), uv_y), 0);
        sum += h_sum / 9.0;
    }

    f_colour = texture(image, uvs) + ((sum / 9.0) * bloom_intensity);
}
```

1.10.10 blur.frag

See Section ??.

1.10.11 box_blur.frag

```
# version 330 core

uniform sampler2D image;

uniform int size=1;
uniform int separation=1;

in vec2 uvs;
out vec4 f_colour;

vec2 textureSize = textureSize(image, 0);

void main() {
    if (size <= 0) {
        return;
    }

```

```
    float count = 0.0;

    for (int i = -size ; i <= size ; ++i) {
        for (int j = -size ; j <= size ; ++j) {
            f_colour += texture(image, uvs + (vec2(i, j) * separation) /
    textureSize).rgba;

            count += 1.0;
        }
    }

    f_colour.rgb /= count;
}
```

1.10.12 calibrate.frag

```
#version 330 core

uniform sampler2D image;

in vec2 uvs;
out vec4 f_colour;

void main() {
    f_colour = vec4(texture(image, uvs).rgba);
}
```

1.10.13 chromatic_abbreviation.frag

```
#version 330 core

in vec2 uvs;
out vec4 f_colour;

uniform sampler2D image;

uniform bool enabled;
uniform vec2 mouseFocusPoint;
uniform float intensity;

void main() {
  if (!enabled) {
    f_colour = texture(image, uvs);
    return;
  }

  float redOffset = 0.009 * intensity;
  float greenOffset = 0.006 * intensity;
  float blueOffset = -0.006 * intensity;

  vec2 texSize = textureSize(image, 0).xy;
  vec2 direction = uvs - mouseFocusPoint;

  f_colour = texture(image, uvs);

  f_colour.r = texture(image, uvs + (direction * vec2(redOffset))).r;
  f_colour.g = texture(image, uvs + (direction * vec2(greenOffset))).g;
  f_colour.b = texture(image, uvs + (direction * vec2(blueOffset))).b;
}
```

1.10.14 crop.frag

```
#version 330 core

uniform sampler2D image;
uniform vec2 relativeSize;
uniform vec2 relativePos;

in vec2 uvs;
out vec4 f_colour;

void main() {
    vec2 sampleCoords = relativeSize.xy * uvs.xy + relativePos.xy;

    float withinBounds = step(0.0, sampleCoords.x) * step(sampleCoords.x, 1.0) *
    step(0.0, sampleCoords.y) * step(sampleCoords.y, 1.0);

    vec3 colour = texture(image, sampleCoords).rgb * withinBounds;
    colour.r += (1 - withinBounds);

    f_colour = vec4(colour, 1.0);
}
```

1.10.15 crt.frag

```
#version 330 core

uniform sampler2D image;
uniform int mode = 1;

in vec2 uvs;
out vec4 f_colour;

void main() {
  if (mode == 0){
    f_colour = vec4(texture(image, uvs).rgb, 1.0);
  } else {
    float flatness = 1.0;

    if (mode == 1) flatness = 5.0;
    else if (mode == 2) flatness = 10.0;

    vec2 center = vec2(0.5, 0.5);
    vec2 off_center = uvs - center;

    // Calculate offset of bulged pixels, increases with distance from center
    off_center *= 1.0 + 0.8 * pow(abs(off_center.yx), vec2(flatness));

    vec2 uvs_2 = center + off_center;

    if (uvs_2.x > 1.0 || uvs_2.x < 0.0 || uvs_2.y > 1.0 || uvs_2.y < 0.0) {
      f_colour = vec4(0.0, 0.0, 0.0, 1.0);
    } else {
      f_colour = vec4(texture(image, uvs_2).rgb, 1.0);

      // Draw horizontal lines
      float fv = fract(uvs_2.y * float(textureSize(image, 0).y));
      fv = min(1.0, 0.8 + 0.5 * min(fv, 1.0 - fv));
      f_colour.rgb *= fv;
    }
  }
}
```

1.10.16 flashlight.frag

```
#version 330 core

uniform sampler2D image;
uniform vec2 center;

in vec2 uvs;
out vec4 f_colour;

vec2 resolution = textureSize(image, 0);
float radius = 100.0; // radius in pixel

float getDistance(vec2 pixelCoord, vec2 playerCoord) {
    return distance(pixelCoord*resolution, playerCoord);
}

void main() {
    float distance = getDistance(uvs, center);
    float a = 0;
    float b = 1;

    // if (distance < radius)
    float factor = 1.0 / (pow((distance / 100), 2) + 1);
    float isLit = step(distance, 10000);

    f_colour = vec4(texture(image, uvs).rgb + factor * isLit, 1.0);

    // if (distance < 10000) {
    //     float factor = 1.0 / (pow((distance / 100), 2) + 1);
    //     f_colour = vec4(texture(image, uvs).rgb + factor, 1.0);
    // }
    // else {
    //     f_colour = vec4(texture(image, uvs).rgb, 1.0);
    // }
}
```

1.10.17 grayscale.frag

```
#version 330 core

uniform sampler2D image;

in vec2 uvs;
out vec4 f_colour;

void main() {
    f_colour = vec4(texture(image, uvs).rgb, 1.0);
    float gray = dot(f_colour.rgb, vec3(0.299, 0.587, 0.114));
    f_colour.rgb = vec3(gray, gray, gray);
}
```

1.10.18 highlight_brightness.frag

See Section ??.

1.10.19 highlight_colour.frag

```
# version 330 core

```

```
uniform sampler2D image;
uniform sampler2D highlight;

uniform vec3 colour;
uniform float threshold;
uniform float intensity;

in vec2 uvs;
out vec4 f_colour;

vec3 normColour = colour / 255;

void main() {
    vec4 pixel = texture(image, uvs);
    float isClose = step(abs(pixel.r - normColour.r), threshold) * step(abs(pixel.
    g - normColour.g), threshold) * step(abs(pixel.b - normColour.b), threshold);

    if (isClose == 1.0) {
        f_colour = vec4(vec3(pixel.rgb * intensity), 1.0);
    } else {
        f_colour = vec4(texture(highlight, uvs).rgb, 1.0);
    }
}
```

1.10.20 lightmap.frag

See Section ??.

1.10.21 occlusion.frag

See Section ??.

1.10.22 rays.frag

```
#version 330 core

uniform sampler2D image;

in vec2 uvs;
out vec4 f_colour;

void main() {
    f_colour = vec4(texture(image, uvs).rgb, 1.0);
}
```

1.10.23 shadowmap.frag

See Section ??.

1.10.24 shake.frag

```
#version 330 core

uniform sampler2D image;
uniform vec2 displacement;

in vec2 uvs;
out vec4 f_colour;
```

```

void main() {
    f_colour = vec4(texture(image, uvs + displacement).rgb, 1.0);
}
```

1.11 data\shaders\vertex

1.11.1 base.vert

```
#version 330 core

in vec2 vert;
in vec2 texCoords;
out vec2 uvs;

void main() {
    uvs = texCoords;
    gl_Position = vec4(vert, 0.0, 1.0);
}
```

1.12 data\states

1.13 data\states\browser

1.13.1 browser.py

```
import pygame
import pyperclip
from data.helpers.database_helpers import delete_game, get_ordered_games
from data.states.browser.widget_dict import BROWSER_WIDGETS
from data.utils.event_types import BrowserEventType
from data.managers.logs import initialise_logger
from data.utils.constants import GAMES_PER_PAGE
from data.managers.window import window
from data.utils.enums import ShaderType
from data.utils.assets import MUSIC
from data.control import _State
from random import randint

logger = initialise_logger(__name__)

class Browser(_State):
    def __init__(self):
        super().__init__()

        self._selected_index = None
        self._filter_column = 'number_of_ply'
        self._filter_ascend = False
        self._games_list = []
        self._page_number = 1

    def cleanup(self):
        super().cleanup()

        if self._selected_index is not None:
            return self._games_list[self._selected_index]

        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:
```

```
                if self._selected_index is None:
                    return
                logger.info(f'COPYING TO CLIPBOARD: {self._games_list[self._selected_index]['final_fen_string']}')
                pyperclip.copy(self._games_list[self._selected_index]['final_fen_string'])

            case BrowserEventType.DELETE_CLICK:
                if self._selected_index is None:
                    return
                delete_game(self._games_list[self._selected_index]['id'])
                self.refresh_games_list()

            case BrowserEventType.REVIEW_CLICK:
                if self._selected_index is None:
                    return

                self.next = 'review'
                self.done = True

            case BrowserEventType.FILTER_COLUMN_CLICK:
                selected_word = BROWSER_WIDGETS['filter_column_dropdown'].get_selected_word()

                if selected_word is None:
                    return

                self.refresh_games_list()

            case BrowserEventType.FILTER_ASCEND_CLICK:
                selected_word = BROWSER_WIDGETS['filter_ascend_dropdown'].get_selected_word()

                if selected_word is None:
                    return

                self.refresh_games_list()

            case BrowserEventType.PAGE_CLICK:
                self._page_number = widget_event.data

                self.refresh_games_list()

            case BrowserEventType.HELP_CLICK:
                self._widget_group.add(BROWSER_WIDGETS['help'])
                self._widget_group.handle_resize(window.size)

    def draw(self):
        self._widget_group.draw()
```

1.13.2 widget_dict.py

```
from data.helpers.database_helpers import get_number_of_games
from data.components.custom_event import CustomEvent
from data.utils.event_types import BrowserEventType
from data.utils.constants import GAMES_PER_PAGE
from data.utils.assets import GRAPHICS
from data.widgets import *

BROWSER_HEIGHT = 0.6

browser_strip = BrowserStrip(
```

```
    relative_position=(0.0, 0.0),
    relative_height=BROWSER_HEIGHT,
    games_list=[]
)

number_of_pages = get_number_of_games() // GAMES_PER_PAGE + 1

carousel_widgets = {
    i: Text(
        relative_position=(0, 0),
        relative_size=(0.3, 0.1),
        text=f"PAGE {i} OF {number_of_pages}",
        fill_colour=(0, 0, 0, 0),
        fit_vertical=False,
        border_width=0,
    )
    for i in range(1, number_of_pages + 1)
}

sort_by_container = Rectangle(
    relative_size=(0.5, 0.1),
    relative_position=(0.01, 0.77),
    anchor_x='right',
    visible=True
)

buttons_container = Rectangle(
    relative_position=(0, 0.025),
    relative_size=(0.5, 0.1),
    scale_mode='height',
    anchor_x='center'
)

top_right_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.15, 0.075),
    fixed_position=(5, 5),
    anchor_x='right',
    scale_mode='height'
)

BROWSER_WIDGETS = {
    'help':
    Icon(
        relative_position=(0, 0),
        relative_size=(1.02, 1.02),
        icon=GRAPHICS['browser_help'],
        anchor_x='center',
        anchor_y='center',
        border_width=0,
        fill_colour=(0, 0, 0, 0)
    ),
    'default': [
        buttons_container,
        sort_by_container,
        top_right_container,
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            anchor_x='right',
            scale_mode='height',
```

```
            base_icon=GRAPHICS['home_base'],
            hover_icon=GRAPHICS['home_hover'],
            press_icon=GRAPHICS['home_press'],
            event=CustomEvent(BrowserEventType.MENU_CLICK)
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['help_base'],
            hover_icon=GRAPHICS['help_hover'],
            press_icon=GRAPHICS['help_press'],
            event=CustomEvent(BrowserEventType.HELP_CLICK)
        ),
        ReactiveIconButton(
            parent=buttons_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['copy_base'],
            hover_icon=GRAPHICS['copy_hover'],
            press_icon=GRAPHICS['copy_press'],
            event=CustomEvent(BrowserEventType.COPY_CLICK),
        ),
        ReactiveIconButton(
            parent=buttons_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            anchor_x='center',
            base_icon=GRAPHICS['delete_base'],
            hover_icon=GRAPHICS['delete_hover'],
            press_icon=GRAPHICS['delete_press'],
            event=CustomEvent(BrowserEventType.DELETE_CLICK),
        ),
        ReactiveIconButton(
            parent=buttons_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            anchor_x='right',
            base_icon=GRAPHICS['review_base'],
            hover_icon=GRAPHICS['review_hover'],
            press_icon=GRAPHICS['review_press'],
            event=CustomEvent(BrowserEventType.REVIEW_CLICK),
        ),
        Text(
            parent=sort_by_container,
            relative_position=(0, 0),
            relative_size=(0.3, 1),
            fit_vertical=False,
            text='SORT BY:',
            border_width=0,
            fill_colour=(0, 0, 0, 0)
        )
    ],
    'browser_strip':
        browser_strip,
    'scroll_area':
    ScrollArea(
        relative_position=(0.0, 0.15),
```

```
        relative_size=(1, BROWSER_HEIGHT),
        vertical=False,
        widget=browser_strip
    ),
    'filter_column_dropdown':
    Dropdown(
        parent=sort_by_container,
        relative_position=(0.3, 0),
        relative_height=0.75,
        anchor_x='right',
        word_list=['time', 'moves', 'winner'],
        fill_colour=(255, 100, 100),
        event=CustomEvent(BrowserEventType.FILTER_COLUMN_CLICK)
    ),
    'filter_ascend_dropdown':
    Dropdown(
        parent=sort_by_container,
        relative_position=(0, 0),
        relative_height=0.75,
        anchor_x='right',
        word_list=['desc', 'asc'],
        fill_colour=(255, 100, 100),
        event=CustomEvent(BrowserEventType.FILTER_ASCEND_CLICK)
    ),
    'page_carousel':
    Carousel(
        relative_position=(0.01, 0.77),
        margin=5,
        widgets_dict=carousel_widgets,
        event=CustomEvent(BrowserEventType.PAGE_CLICK),
    )
}
```

1.14 data\states\config

1.14.1 config.py

```
import pygame
from data.states.config.default_config import default_config
from data.states.config.widget_dict import CONFIG_WIDGETS
from data.utils.event_types import ConfigEventType
from data.managers.logs import initialise_logger
from data.managers.animation import animation
from data.utils.constants import ShaderType
from data.utils.assets import MUSIC, SFX
from data.managers.window import window
from data.managers.audio import audio
from data.managers.theme import theme
from data.utils.enums import Colour
from data.control import _State
from random import randint

logger = initialise_logger(__name__)

class Config(_State):
    def __init__(self):
        super().__init__()

        self._config = None
        self._valid_fen = True
        self._selected_preset = None
```

```
25
26      def cleanup(self):
27          super().cleanup()
28
29          window.clear_apply_arguments(ShaderType.BLOOM)
30
31          return self._config
32
33      def startup(self, persist=None):
34          super().startup(CONFIG_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
35          window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
    Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.9)
36
37          CONFIG_WIDGETS['invalid_fen_string'].kill()
38          CONFIG_WIDGETS['help'].kill()
39
40          self._config = default_config
41
42          if persist:
43              self._config['FEN_STRING'] = persist
44
45          self.set_fen_string(self._config['FEN_STRING'])
46          self.toggle_pvc(self._config['CPU_ENABLED'])
47          self.set_active_colour(self._config['COLOUR'])
48
49          CONFIG_WIDGETS['cpu_depth_carousel'].set_to_key(self._config['CPU_DEPTH'])
50          if self._config['CPU_ENABLED']:
51              self.create_depth_picker()
52          else:
53              self.remove_depth_picker()
54
55          self.draw()
56
57      def create_depth_picker(self):
58          # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.8))
59          # CONFIG_WIDGETS['start_button'].set_image()
60          CONFIG_WIDGETS['cpu_depth_carousel'].set_surface_size(window.size)
61          CONFIG_WIDGETS['cpu_depth_carousel'].set_image()
62          CONFIG_WIDGETS['cpu_depth_carousel'].set_geometry()
63          self._widget_group.add(CONFIG_WIDGETS['cpu_depth_carousel'])
64
65      def remove_depth_picker(self):
66          # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.7))
67          # CONFIG_WIDGETS['start_button'].set_image()
68
69          CONFIG_WIDGETS['cpu_depth_carousel'].kill()
70
71      def toggle_pvc(self, pvc_enabled):
72          if pvc_enabled:
73              CONFIG_WIDGETS['pvc_button'].set_locked(True)
74              CONFIG_WIDGETS['pvp_button'].set_locked(False)
75          else:
76              CONFIG_WIDGETS['pvp_button'].set_locked(True)
77              CONFIG_WIDGETS['pvc_button'].set_locked(False)
78
79          self._config['CPU_ENABLED'] = pvc_enabled
80
81          if self._config['CPU_ENABLED']:
82              self.create_depth_picker()
83          else:
84              self.remove_depth_picker()
85
```

```
86      def set_fen_string(self, new_fen_string):
87          CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
88          self._config['FEN_STRING'] = new_fen_string
89
90          self.set_preset_overlay(new_fen_string)
91
92          try:
93              CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
94              CONFIG_WIDGETS['invalid_fen_string'].kill()
95
96              if new_fen_string[-1].lower() == 'r':
97                  self.set_active_colour(Colour.RED)
98              else:
99                  self.set_active_colour(Colour.BLUE)
100
101             self._valid_fen = True
102         except:
103             CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
104             self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])
105
106             window.set_effect(ShaderType.SHAKE)
107             animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE)
    )
108
109             audio.play_sfx(SFX['error_1'])
110             audio.play_sfx(SFX['error_2'])
111
112             self._valid_fen = False
113
114     def get_event(self, event):
115         widget_event = self._widget_group.process_event(event)
116
117         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
118             CONFIG_WIDGETS['help'].kill()
119
120         if widget_event is None:
121             return
122
123         match widget_event.type:
124             case ConfigEventType.GAME_CLICK:
125                 if self._valid_fen:
126                     self.next = 'game'
127                     self.done = True
128
129             case ConfigEventType.MENU_CLICK:
130                 self.next = 'menu'
131                 self.done = True
132
133             case ConfigEventType.TIME_CLICK:
134                 self._config['TIME_ENABLED'] = not(widget_event.data)
135                 CONFIG_WIDGETS['timer_button'].set_next_icon()
136
137             case ConfigEventType.PVP_CLICK:
138                 self.toggle_pvc(False)
139
140             case ConfigEventType.PVC_CLICK:
141                 self.toggle_pvc(True)
142
143             case ConfigEventType.FEN_STRING_TYPE:
144                 self.set_fen_string(widget_event.text)
145
146             case ConfigEventType.TIME_TYPE:
```

```
147                 if widget_event.text == '':
148                     self._config['TIME'] = 5
149                 else:
150                     self._config['TIME'] = float(widget_event.text)
151 
152             case ConfigEventType.CPU_DEPTH_CLICK:
153                 self._config['CPU_DEPTH'] = int(widget_event.data)
154 
155             case ConfigEventType.PRESET_CLICK:
156                 self.set_fen_string(widget_event.fen_string)
157 
158             case ConfigEventType.SETUP_CLICK:
159                 if self._valid_fen:
160                     self.next = 'editor'
161                     self.done = True
162 
163             case ConfigEventType.COLOUR_CLICK:
164                 self.set_active_colour(widget_event.data.get_flipped_colour())
165 
166             case ConfigEventType.HELP_CLICK:
167                 self._widget_group.add(CONFIG_WIDGETS['help'])
168                 self._widget_group.handle_resize(window.size)
169 
170     def set_preset_overlay(self, fen_string):
171         fen_string_widget_map = {
172             'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
    PdNaFaNa3Sa b': 'preset_1',
173             'sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3/10/2
    RaNaFaNa3Sa b': 'preset_2',
174             'sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3RaNaPa3Pc/4Fa5
    /3PdNaPa3Sa b': 'preset_3'
175         }
176 
177         if fen_string in fen_string_widget_map:
178             self._selected_preset = CONFIG_WIDGETS[fen_string_widget_map[
    fen_string]]
179         else:
180             self._selected_preset = None
181 
182     def set_active_colour(self, colour):
183         if self._config['COLOUR'] != colour:
184             CONFIG_WIDGETS['to_move_button'].set_next_icon()
185 
186         self._config['COLOUR'] = colour
187 
188         if colour == Colour.BLUE:
189             CONFIG_WIDGETS['to_move_text'].set_text('BLUE TO MOVE')
190         elif colour == Colour.RED:
191             CONFIG_WIDGETS['to_move_text'].set_text('RED TO MOVE')
192 
193         if self._valid_fen:
194             self._config['FEN_STRING'] = self._config['FEN_STRING'][:-1] + colour.
    name[0].lower()
195             CONFIG_WIDGETS['fen_string_input'].set_text(self._config['FEN_STRING'
    ])
196 
197     def draw(self):
198         self._widget_group.draw()
199 
200         if self._selected_preset:
201             pygame.draw.rect(window.screen, theme['borderPrimary'], (*self.
    _selected_preset.position, *self._selected_preset.size), width=int(theme['
```

```
        borderWidth']))

    def update(self, **kwargs):
        self._widget_group.update()
        super().update(**kwargs)
```

1.14.2 default_config.py

```
from data.utils.enums import Colour

default_config = {
    'FEN_STRING': 'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa b',
    'COLOUR': Colour.BLUE,
    'TIME_ENABLED': True,
    'CPU_ENABLED': False,
    'CPU_DEPTH': 2,
    'TIME': 5,
}
```

1.14.3 widget_dict.py

```
from data.widgets import *
from data.states.config.default_config import default_config
from data.helpers.asset_helpers import get_highlighted_icon
from data.components.custom_event import CustomEvent
from data.utils.event_types import ConfigEventType
from data.utils.assets import GRAPHICS
from data.managers.theme import theme
from data.utils.enums import Colour

def float_validator(num_string):
    try:
        float(num_string)
        return True
    except:
        return False

if default_config['CPU_ENABLED']:
    pvp_icons = {False: GRAPHICS['swords'], True: GRAPHICS['swords']}
    pvc_icons = {True: GRAPHICS['robot'], False: GRAPHICS['robot']}
    pvc_locked = True
    pvp_locked = False
else:
    pvp_icons = {True: GRAPHICS['swords'], False: GRAPHICS['swords']}
    pvc_icons = {False: GRAPHICS['robot'], True: GRAPHICS['robot']}
    pvc_locked = False
    pvp_locked = True

if default_config['TIME_ENABLED']:
    time_enabled_icons = {True: GRAPHICS['timer'], False: get_highlighted_icon(GRAPHICS['timer'])}
else:
    time_enabled_icons = {False: get_highlighted_icon(GRAPHICS['timer']), True: GRAPHICS['timer']}

if default_config['COLOUR'] == Colour.BLUE:
    colour_icons = {Colour.BLUE: GRAPHICS['pharaoh_0_a'], Colour.RED: GRAPHICS['pharaoh_1_a']}
else:
```

```
    colour_icons = {Colour.RED: GRAPHICS['pharaoh_1_a'], Colour.BLUE: GRAPHICS['
    pharaoh_0_a']}

preview_container = Rectangle(
    relative_position=(-0.15, 0),
    relative_size=(0.65, 0.9),
    anchor_x='center',
    anchor_y='center',
)

config_container = Rectangle(
    relative_position=(0.325, 0),
    relative_size=(0.3, 0.9),
    anchor_x='center',
    anchor_y='center',
)

to_move_container = Rectangle(
    parent=config_container,
    relative_size=(0.9, 0.15),
    relative_position=(0, 0.1),
    anchor_x='center'
)

board_thumbnail = BoardThumbnail(
    parent=preview_container,
    relative_position=(0, 0),
    relative_width=0.7,
    scale_mode='width',
    anchor_x='right',
)

top_right_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.15, 0.075),
    fixed_position=(5, 5),
    anchor_x='right',
    scale_mode='height'
)

CONFIG_WIDGETS = {
    'help':
    Icon(
        relative_position=(0, 0),
        relative_size=(1.02, 1.02),
        icon=GRAPHICS['config_help'],
        anchor_x='center',
        anchor_y='center',
        border_width=0,
        fill_colour=(0, 0, 0, 0)
    ),
    'default': [
        preview_container,
        config_container,
        to_move_container,
        top_right_container,
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            anchor_x='right',
            scale_mode='height',
```

```
            base_icon=GRAPHICS['home_base'],
            hover_icon=GRAPHICS['home_hover'],
            press_icon=GRAPHICS['home_press'],
            event=CustomEvent(ConfigEventType.MENU_CLICK)
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['help_base'],
            hover_icon=GRAPHICS['help_hover'],
            press_icon=GRAPHICS['help_press'],
            event=CustomEvent(ConfigEventType.HELP_CLICK)
        ),
        TextInput(
            parent=config_container,
            relative_position=(0.3, 0.3),
            relative_size=(0.65, 0.15),
            fit_vertical=True,
            placeholder='TIME CONTROL (DEFAULT 5)',
            default=str(default_config['TIME']),
            border_width=5,
            margin=20,
            validator=float_validator,
            event=CustomEvent(ConfigEventType.TIME_TYPE)
        ),
        Text(
            parent=config_container,
            fit_vertical=False,
            relative_position=(0.75, 0.3),
            relative_size=(0.2, 0.15),
            text='MINS',
            border_width=0,
            fill_colour=(0, 0, 0, 0)
        ),
        TextButton(
            parent=preview_container,
            relative_position=(0.3, 0),
            relative_size=(0.15, 0.15),
            text='CUSTOM',
            anchor_y='bottom',
            fit_vertical=False,
            margin=10,
            event=CustomEvent(ConfigEventType.SETUP_CLICK)
        )
    ],
    'board_thumbnail':
        board_thumbnail,
    'fen_string_input':
    TextInput(
        parent=preview_container,
        relative_position=(0, 0),
        relative_size=(0.55, 0.15),
        fit_vertical=False,
        placeholder='ENTER FEN STRING',
        default='sc3ncfcncpb2/2pc7/3Pd7/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
    Pa2/2PdNaFaNa3Sa b',
        border_width=5,
        anchor_y='bottom',
        anchor_x='right',
        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'],
```

```
    ),
    'preset_1':
    BoardThumbnailButton(
        parent=preview_container,
        relative_width=0.25,
        relative_position=(0, 0),
        scale_mode='width',
        fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3
    /7Pa2/2PdNaFaNa3Sa b",
        event=CustomEvent(ConfigEventType.PRESET_CLICK)
    ),
    'preset_2':
    BoardThumbnailButton(
        parent=preview_container,
        relative_width=0.25,
        relative_position=(0, 0.35),
        scale_mode='width',
        fen_string="sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3
    /10/2RaNaFaNa3Sa b",
        event=CustomEvent(ConfigEventType.PRESET_CLICK)
    ),
    'preset_3':
    BoardThumbnailButton(
        parent=preview_container,
        relative_width=0.25,
        relative_position=(0, 0.7),
        scale_mode='width',
        fen_string="sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3
    RaNaPa3Pc/4Fa5/3PdNaPa3Sa b",
        event=CustomEvent(ConfigEventType.PRESET_CLICK)
    ),
    'to_move_button':
    MultipleIconButton(
        parent=to_move_container,
        scale_mode='height',
        relative_position=(0, 0),
        relative_size=(1, 1),
        icons_dict=colour_icons,
        anchor_x='left',
        event=CustomEvent(ConfigEventType.COLOUR_CLICK)
    ),
    'to_move_text':
    Text(
        parent=to_move_container,
        relative_position=(0, 0),
        relative_size=(0.75, 1),
        fit_vertical=False,
        text='TO MOVE',
        anchor_x='right'
    ),
    'cpu_depth_carousel':
    Carousel(
        parent=config_container,
        relative_position=(0, 0.65),
        event=CustomEvent(ConfigEventType.CPU_DEPTH_CLICK),
        anchor_x='center',
        border_width=0,
        fill_colour=(0, 0, 0, 0),
        widgets_dict={
            2: Text(
                parent=config_container,
                relative_position=(0, 0),
```

```
                relative_size=(0.8, 0.075),
                text="EASY",
                margin=0,
                border_width=0,
                fill_colour=(0, 0, 0, 0)
            ),
            3: Text(
                parent=config_container,
                relative_position=(0, 0),
                relative_size=(0.8, 0.075),
                text="MEDIUM",
                margin=0,
                border_width=0,
                fill_colour=(0, 0, 0, 0)
            ),
            4: Text(
                parent=config_container,
                relative_position=(0, 0),
                relative_size=(0.8, 0.075),
                text="HARD",
                margin=0,
                border_width=0,
                fill_colour=(0, 0, 0, 0)
            ),
        }
    )
}
```

1.15 data\states\editor

1.15.1 editor.py

```
import pygame
import pyperclip
from data.states.game.components.bitboard_collection import BitboardCollection
from data.utils.enums import Colour, RotationDirection, Piece, Rotation
from data.states.game.components.fen_parser import encode_fen_string
from data.states.game.components.overlay_draw import OverlayDraw
from data.states.game.components.piece_group import PieceGroup
from data.helpers.bitboard_helpers import coords_to_bitboard
from data.helpers.board_helpers import screen_pos_to_coords
from data.states.game.components.father import DragAndDrop
from data.states.editor.widget_dict import EDITOR_WIDGETS
from data.utils.event_types import EditorEventType
from data.managers.logs import initialise_logger
from data.managers.window import window
from data.control import _State

logger = initialise_logger(__name__)

class Editor(_State):
    def __init__(self):
        super().__init__()

        self._bitboards = None
        self._piece_group = None
        self._selected_coords = None
        self._selected_tool = None
        self._selected_tool_colour = None
        self._initial_fen_string = None
        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['
    chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
77
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,
    dragged_rotation)
199         self._overlay_draw.set_hover_limit(False)
200 
201     def remove_dragged_piece(self, coords):
```

```
202         piece, colour, rotation = self._drag_and_drop.get_dragged_info()
203 
204         if coords and coords != self._selected_coords and piece != Piece.SPHINX:
205             self.remove_piece()
206             self.selected_coords = coords
207             self.set_piece(piece, colour, rotation)
208             self.selected_coords = None
209 
210         self._drag_and_drop.remove_dragged_piece()
211         self._overlay_draw.set_hover_limit(True)
212 
213     def set_piece(self, piece, colour, rotation):
214         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
    selected_coords == (9, 0):
215             return
216 
217         self.remove_piece()
218 
219         selected_bitboard = coords_to_bitboard(self.selected_coords)
220         self._bitboards.set_square(selected_bitboard, piece, colour)
221         self._bitboards.set_rotation(selected_bitboard, rotation)
222 
223         self.refresh_pieces()
224 
225     def remove_piece(self):
226         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
    selected_coords == (9, 0):
227             return
228 
229         selected_bitboard = coords_to_bitboard(self.selected_coords)
230         self._bitboards.clear_square(selected_bitboard, Colour.BLUE)
231         self._bitboards.clear_square(selected_bitboard, Colour.RED)
232         self._bitboards.clear_rotation(selected_bitboard)
233 
234         self.refresh_pieces()
235 
236     def rotate_piece(self, rotation_direction):
237         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
    selected_coords == (9, 0):
238             return
239 
240         selected_bitboard = coords_to_bitboard(self.selected_coords)
241 
242         if self._bitboards.get_piece_on(selected_bitboard, Colour.BLUE) is None
    and self._bitboards.get_piece_on(selected_bitboard, Colour.RED) is None:
243             return
244 
245         current_rotation = self._bitboards.get_rotation_on(selected_bitboard)
246 
247         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),
```

```
    relative_size=(0.13, 0.65),
    scale_mode='height',
    anchor_y='center',
    anchor_x='center'
)

red_pieces_container = Rectangle(
    relative_position=(-0.25, 0),
    relative_size=(0.13, 0.65),
    scale_mode='height',
    anchor_y='center',
    anchor_x='center'
)

bottom_actions_container = Rectangle(
    relative_position=(0, 0.05),
    relative_size=(0.4, 0.1),
    anchor_x='center',
    anchor_y='bottom'
)

top_actions_container = Rectangle(
    relative_position=(0, 0.05),
    relative_size=(0.3, 0.1),
    anchor_x='center',
    scale_mode='height'
)

top_right_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.15, 0.075),
    fixed_position=(5, 5),
    anchor_x='right',
    scale_mode='height'
)

EDITOR_WIDGETS = {
    'help':
    Icon(
        relative_position=(0, 0),
        relative_size=(1.02, 1.02),
        icon=GRAPHICS['editor_help'],
        anchor_x='center',
        anchor_y='center',
        border_width=0,
        fill_colour=(0, 0, 0, 0)
    ),
    'default': [
        red_pieces_container,
        blue_pieces_container,
        bottom_actions_container,
        top_actions_container,
        top_right_container,
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            anchor_x='right',
            scale_mode='height',
            base_icon=GRAPHICS['home_base'],
            hover_icon=GRAPHICS['home_hover'],
            press_icon=GRAPHICS['home_press'],
```

```
            event=CustomEvent(EditorEventType.MENU_CLICK)
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['help_base'],
            hover_icon=GRAPHICS['help_hover'],
            press_icon=GRAPHICS['help_press'],
            event=CustomEvent(EditorEventType.HELP_CLICK)
        ),
        ReactiveIconButton(
            parent=bottom_actions_container,
            relative_position=(0.06, 0),
            relative_size=(1, 1),
            anchor_x='center',
            scale_mode='height',
            base_icon=GRAPHICS['clockwise_arrow_base'],
            hover_icon=GRAPHICS['clockwise_arrow_hover'],
            press_icon=GRAPHICS['clockwise_arrow_press'],
            event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
    rotation_direction=RotationDirection.CLOCKWISE)
        ),
        ReactiveIconButton(
            parent=bottom_actions_container,
            relative_position=(-0.06, 0),
            relative_size=(1, 1),
            anchor_x='center',
            scale_mode='height',
            base_icon=GRAPHICS['anticlockwise_arrow_base'],
            hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
            press_icon=GRAPHICS['anticlockwise_arrow_press'],
            event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
    rotation_direction=RotationDirection.ANTICLOCKWISE)
        ),
        ReactiveIconButton(
            parent=top_actions_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            anchor_x='right',
            base_icon=GRAPHICS['copy_base'],
            hover_icon=GRAPHICS['copy_hover'],
            press_icon=GRAPHICS['copy_press'],
            event=CustomEvent(EditorEventType.COPY_CLICK),
        ),
        ReactiveIconButton(
            parent=top_actions_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['delete_base'],
            hover_icon=GRAPHICS['delete_hover'],
            press_icon=GRAPHICS['delete_press'],
            event=CustomEvent(EditorEventType.EMPTY_CLICK),
        ),
        ReactiveIconButton(
            parent=top_actions_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
```

```
133             anchor_x='center',
134             base_icon=GRAPHICS['discard_arrow_base'],
135             hover_icon=GRAPHICS['discard_arrow_hover'],
136             press_icon=GRAPHICS['discard_arrow_press'],
137             event=CustomEvent(EditorEventType.RESET_CLICK),
138         ),
139         ReactiveIconButton(
140             relative_position=(0, 0),
141             fixed_position=(10, 0),
142             relative_size=(0.1, 0.1),
143             anchor_x='right',
144             anchor_y='center',
145             scale_mode='height',
146             base_icon=GRAPHICS['play_arrow_base'],
147             hover_icon=GRAPHICS['play_arrow_hover'],
148             press_icon=GRAPHICS['play_arrow_press'],
149             event=CustomEvent(EditorEventType.START_CLICK),
150         ),
151         ReactiveIconButton(
152             relative_position=(0, 0),
153             fixed_position=(10, 0),
154             relative_size=(0.1, 0.1),
155             anchor_y='center',
156             scale_mode='height',
157             base_icon=GRAPHICS['return_arrow_base'],
158             hover_icon=GRAPHICS['return_arrow_hover'],
159             press_icon=GRAPHICS['return_arrow_press'],
160             event=CustomEvent(EditorEventType.CONFIG_CLICK),
161         )
162     ],
163     'blue_piece_buttons': {},
164     'red_piece_buttons': {},
165     'erase_button':
166     MultipleIconButton(
167         parent=red_pieces_container,
168         relative_position=(0, 0),
169         relative_size=(0.2, 0.2),
170         scale_mode='height',
171         margin=10,
172         icons_dict={True: GRAPHICS['eraser'], False: get_highlighted_icon(GRAPHICS
    ['eraser'])},
173         event=CustomEvent(EditorEventType.ERASE_CLICK),
174     ),
175     'move_button':
176     MultipleIconButton(
177         parent=blue_pieces_container,
178         relative_position=(0, 0),
179         relative_size=(0.2, 0.2),
180         scale_mode='height',
181         box_colours=BLUE_BUTTON_COLOURS,
182         icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
    ['finger'])},
183         event=CustomEvent(EditorEventType.MOVE_CLICK),
184     ),
185     'chessboard':
186     Chessboard(
187         relative_position=(0, 0),
188         relative_width=0.4,
189         scale_mode='width',
190         anchor_x='center',
191         anchor_y='center'
192     ),
```

```
    'blue_start_button':
    MultipleIconButton(
        parent=bottom_actions_container,
        relative_position=(0, 0),
        relative_size=(1, 1),
        scale_mode='height',
        anchor_x='right',
        box_colours=BLUE_BUTTON_COLOURS,
        icons_dict={False: get_highlighted_icon(GRAPHICS['pharaoh_0_a']), True:
    GRAPHICS['pharaoh_0_a']},
        event=CustomEvent(EditorEventType.BLUE_START_CLICK)
    ),
    'red_start_button':
    MultipleIconButton(
        parent=bottom_actions_container,
        relative_position=(0, 0),
        relative_size=(1, 1),
        scale_mode='height',
        icons_dict={True: GRAPHICS['pharaoh_1_a'], False: get_highlighted_icon(
    GRAPHICS['pharaoh_1_a'])},
        event=CustomEvent(EditorEventType.RED_START_CLICK)
    )
}

for index, piece in enumerate([piece for piece in Piece if piece != Piece.SPHINX])
    :
    blue_icon = GRAPHICS[f'{piece.name.lower()}_0_a']
    dimmed_blue_icon = get_highlighted_icon(blue_icon)

    EDITOR_WIDGETS['blue_piece_buttons'][piece] = MultipleIconButton(
        parent=blue_pieces_container,
        relative_position=(0, (index + 1) / 5),
        relative_size=(0.2, 0.2),
        scale_mode='height',
        box_colours=BLUE_BUTTON_COLOURS,
        icons_dict={True: blue_icon, False: dimmed_blue_icon},
        event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
    active_colour=Colour.BLUE)
    )

    red_icon = GRAPHICS[f'{piece.name.lower()}_1_a']

    dimmed_red_icon = get_highlighted_icon(red_icon)

    EDITOR_WIDGETS['red_piece_buttons'][piece] = MultipleIconButton(
        parent=red_pieces_container,
        relative_position=(0, (index + 1) / 5),
        relative_size=(0.2, 0.2),
        scale_mode='height',
        icons_dict={True: red_icon, False: dimmed_red_icon},
        event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
    active_colour=Colour.RED)
    )
```

1.16 data\states\game

1.16.1 game.py

```
import pygame
from functools import partial
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
50         self.model = GameModel(persist)
51         self.view = GameView(self.model)
52         self.pause_view = PauseView(self.model)
53         self.win_view = WinView(self.model)
54         self.controller = GameController(self.model, self.view, self.win_view,
       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),
```

```
    fit_vertical=False,
    anchor_x='right',
    anchor_y='bottom',
    text="   Draw",
    margin=5,
    event=CustomEvent(GameEventType.DRAW_CLICK)
)

top_right_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.225, 0.075),
    fixed_position=(5, 5),
    anchor_x='right',
    scale_mode='height'
)

GAME_WIDGETS = {
    'help':
    Icon(
        relative_position=(0, 0),
        relative_size=(1.02, 1.02),
        icon=GRAPHICS['game_help'],
        anchor_x='center',
        anchor_y='center',
        border_width=0,
        fill_colour=(0, 0, 0, 0)
    ),
    'tutorial':
    Icon(
        relative_position=(0, 0),
        relative_size=(0.9, 0.9),
        icon=GRAPHICS['game_tutorial'],
        anchor_x='center',
        anchor_y='center',
    ),
    'default': [
        right_container,
        rotate_container,
        top_right_container,
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            anchor_x='right',
            scale_mode='height',
            base_icon=GRAPHICS['home_base'],
            hover_icon=GRAPHICS['home_hover'],
            press_icon=GRAPHICS['home_press'],
            event=CustomEvent(GameEventType.MENU_CLICK)
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['tutorial_base'],
            hover_icon=GRAPHICS['tutorial_hover'],
            press_icon=GRAPHICS['tutorial_press'],
            event=CustomEvent(GameEventType.TUTORIAL_CLICK)
        ),
        ReactiveIconButton(
            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         ),
```

```
    'by_timeout':
    Icon(
        parent=win_container,
        relative_position=(0, 0.375),
        relative_size=(0.8, 0.1),
        anchor_x='center',
        border_width=0,
        margin=0,
        icon=GRAPHICS['by_timeout'],
        fill_colour=(0, 0, 0, 0),
    )
}
```

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

```
from data.states.game.components.particles_draw import ParticlesDraw
from data.helpers.board_helpers import coords_to_screen_pos
from data.managers.animation import animation
from data.utils.constants import ShaderType
from data.managers.window import window
from data.utils.enums import Colour

class CaptureDraw:
    def __init__(self, board_position, board_size):
        self._board_position = board_position
        self._square_size = board_size[0] / 10
        self._particles_draw = ParticlesDraw()

    def add_capture(self, piece, colour, rotation, piece_coords, sphinx_coords,
    active_colour, particles=True, shake=True):
        if particles:
            self._particles_draw.add_captured_piece(
                piece,
                colour,
                rotation,
                coords_to_screen_pos(piece_coords, self._board_position, self.
    _square_size),
                self._square_size
            )
            self._particles_draw.add_sparks(
                3,
                (255, 0, 0) if active_colour == Colour.RED else (0, 0, 255),
                coords_to_screen_pos(sphinx_coords, self._board_position, self.
    _square_size)
            )

        if shake:
            window.set_effect(ShaderType.SHAKE)
```

```
            animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE)
    )

    def draw(self, screen):
        self._particles_draw.draw(screen)

    def update(self):
        self._particles_draw.update()

    def handle_resize(self, board_position, board_size):
        self._board_position = board_position
        self._square_size = board_size[0] / 10
```

1.17.4 father.py

```
import pygame
from data.states.game.components.piece_sprite import PieceSprite
from data.utils.enums import CursorMode
from data.managers.cursor import cursor

DRAG_THRESHOLD = 500

class DragAndDrop:
    def __init__(self, board_position, board_size, change_cursor=True):
        self._board_position = board_position
        self._board_size = board_size
        self._change_cursor = change_cursor
        self._ticks_since_drag = 0

        self.dragged_sprite = None

    def set_dragged_piece(self, piece, colour, rotation):
        sprite = PieceSprite(piece=piece, colour=colour, rotation=rotation)
        sprite.set_geometry((0, 0), self._board_size[0] / 10)
        sprite.set_image()

        self.dragged_sprite = sprite
        self._ticks_since_drag = pygame.time.get_ticks()

        if self._change_cursor:
            cursor.set_mode(CursorMode.CLOSEDHAND)

    def remove_dragged_piece(self):
        self.dragged_sprite = None
        time_dragged = pygame.time.get_ticks() - self._ticks_since_drag
        self._ticks_since_drag = 0

        if self._change_cursor:
            cursor.set_mode(CursorMode.OPENHAND)

        return time_dragged > DRAG_THRESHOLD

    def get_dragged_info(self):
        return self.dragged_sprite.type, self.dragged_sprite.colour, self.
    dragged_sprite.rotation

    def draw(self, screen):
        if self.dragged_sprite is None:
            return

        self.dragged_sprite.rect.center = pygame.mouse.get_pos()
        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 PHARAOH 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 Pharaoh 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)
```

```
                empty_count = 0

            row_string += square

        if index % 10 == 9:
            if empty_count > 0:
                fen_string = '/' + row_string + str(empty_count) + fen_string
            else:
                fen_string = '/' + row_string + fen_string

            row_string = ''
            empty_count = 0

    fen_string = fen_string[1:]

    if bitboard_collection.active_colour == Colour.BLUE:
        colour = 'b'
    else:
        colour = 'r'

    return fen_string + ' ' + colour
```

1.17.6 laser.py

```
from data.utils.constants import A_FILE_MASK, J_FILE_MASK, ONE_RANK_MASK,
    EIGHT_RANK_MASK, EMPTY_BB
from data.helpers import bitboard_helpers as bb_helpers
from data.utils.enums import Piece, Colour, Rotation

class Laser:
    def __init__(self, bitboards):
        self._bitboards = bitboards
        self.hit_square_bitboard, self.piece_hit, self.laser_path, self.
    path_bitboard, self.pieces_on_trajectory, self.end_cap = self.
    calculate_trajectory()

        if (self.hit_square_bitboard != EMPTY_BB):
            self.piece_rotation = self._bitboards.get_rotation_on(self.
    hit_square_bitboard)
            self.piece_colour = self._bitboards.get_colour_on(self.
    hit_square_bitboard)

    def calculate_trajectory(self):
        current_square = self._bitboards.get_piece_bitboard(Piece.SPHINX, self.
    _bitboards.active_colour)
        previous_direction = self._bitboards.get_rotation_on(current_square)
        trajectory_bitboard = 0b0
        trajectory_list = []
        square_animation_states = []
        pieces_on_trajectory = []

        while current_square:
            current_piece = self._bitboards.get_piece_on(current_square, Colour.
    BLUE) or self._bitboards.get_piece_on(current_square, Colour.RED)
            current_rotation = self._bitboards.get_rotation_on(current_square)

            next_square, direction, piece_hit = self.calculate_next_square(
    current_square, current_piece, current_rotation, previous_direction)

            trajectory_bitboard |= current_square
            trajectory_list.append(bb_helpers.bitboard_to_coords(current_square))
            square_animation_states.append(direction)
```

```
31
32              if previous_direction != direction or (current_piece == Piece.ANUBIS
    and not piece_hit):
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                  if piece_hit or current_piece == Piece.ANUBIS:
42                      end_cap = True
43                  else:
44                      end_cap = False
45
46                  return hit_square_bitboard, piece_hit, list(zip(trajectory_list,
    square_animation_states)), trajectory_bitboard, pieces_on_trajectory, end_cap
47
48              current_square = next_square
49              previous_direction = direction
50
51      def calculate_next_square(self, square, piece, rotation, previous_direction):
52          match piece:
53              case Piece.SPHINX:
54                  if previous_direction != rotation:
55                      return EMPTY_BB, previous_direction, None
56
57                  next_square = self.next_square_bitboard(square, rotation)
58                  return next_square, previous_direction, Piece.SPHINX
59
60              case Piece.PYRAMID:
61                  if previous_direction in [rotation, rotation.get_clockwise()]:
62                      return EMPTY_BB, previous_direction, Piece.PYRAMID
63
64                  if previous_direction == rotation.get_anticlockwise():
65                      new_direction = previous_direction.get_clockwise()
66                  else:
67                      new_direction = previous_direction.get_anticlockwise()
68
69                  next_square = self.next_square_bitboard(square, new_direction)
70
71                  return next_square, new_direction, None
72
73              case Piece.ANUBIS:
74                  if previous_direction == rotation.get_clockwise().get_clockwise():
75                      return EMPTY_BB, previous_direction, None
76
77                  return EMPTY_BB, previous_direction, Piece.ANUBIS
78
79              case Piece.SCARAB:
80                  if previous_direction in [rotation.get_clockwise(), rotation.
    get_anticlockwise()]:
81                      new_direction = previous_direction.get_anticlockwise()
82                  else:
83                      new_direction = previous_direction.get_clockwise()
84
85                  next_square = self.next_square_bitboard(square, new_direction)
86
87                  return next_square, new_direction, None
88
89              case Piece.PHARAOH:
```

```
                return EMPTY_BB, previous_direction, Piece.PHARAOH

            case None:
                next_square = self.next_square_bitboard(square, previous_direction
    )

                return next_square, previous_direction, None

    def next_square_bitboard(self, src_bitboard, previous_direction):
        match previous_direction:
            case Rotation.UP:
                masked_src_bitboard = src_bitboard & EIGHT_RANK_MASK
                return masked_src_bitboard << 10
            case Rotation.RIGHT:
                masked_src_bitboard = src_bitboard & J_FILE_MASK
                return masked_src_bitboard << 1
            case Rotation.DOWN:
                masked_src_bitboard = src_bitboard & ONE_RANK_MASK
                return masked_src_bitboard >> 10
            case Rotation.LEFT:
                masked_src_bitboard = src_bitboard & A_FILE_MASK
                return masked_src_bitboard >> 1
```

1.17.7 laser_draw.py

See Section ??.

1.17.8 move.py

```
import re
from data.helpers.bitboard_helpers import notation_to_bitboard, coords_to_bitboard
    , bitboard_to_coords, bitboard_to_notation
from data.utils.enums import MoveType, Colour, RotationDirection
from data.managers.logs import initialise_logger

logger = initialise_logger(__name__)

class Move():
    def __init__(self, move_type, src, dest=None, rotation_direction=None):
        self.move_type = move_type
        self.src = src
        self.dest = dest
        self.rotation_direction = rotation_direction

    def to_notation(self, colour, piece, hit_square_bitboard):
        hit_square = ''
        if colour == Colour.BLUE:
            piece = piece.upper()

        if hit_square_bitboard:
            hit_square = 'x' + bitboard_to_notation(hit_square_bitboard)

        if self.move_type == MoveType.MOVE:
            return 'M' + piece + bitboard_to_notation(self.src) +
    bitboard_to_notation(self.dest) + hit_square
        else:
            return 'R' + piece + bitboard_to_notation(self.src) + self.
    rotation_direction + hit_square

    def __str__(self):
        rotate_text = ''
```

```
30          coords_1 = '(' + chr(bitboard_to_coords(self.src)[0] + 65) + ',' + str(
    bitboard_to_coords(self.src)[1] + 1) + ')'
31
32          if self.move_type == MoveType.ROTATE:
33              rotate_text = ' ' + self.rotation_direction.name
34              return f'{self.move_type.name}{rotate_text}: ON {coords_1}'
35
36          elif self.move_type == MoveType.MOVE:
37              coords_2 = '(' + chr(bitboard_to_coords(self.dest)[0] + 65) + ', ' +
    str(bitboard_to_coords(self.dest)[1] + 1) + ')'
38              return f'{self.move_type.name}{rotate_text}: FROM {coords_1} TO {
    coords_2}'
39
40          # (Rotation: {self.rotation_direction})
41
42      @classmethod
43      def instance_from_notation(move_cls, notation):
44          try:
45              notation = notation.split('x')[0]
46              move_type = notation[0].lower()
47
48              moves = notation[2:]
49              letters = re.findall(r'[A-Za-z]+', moves)
50              numbers = re.findall(r'\d+', moves)
51
52              if move_type == MoveType.MOVE:
53                  src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
54                  dest_bitboard = notation_to_bitboard(letters[1] + numbers[1])
55
56                  return move_cls(move_type, src_bitboard, dest_bitboard)
57
58              elif move_type == MoveType.ROTATE:
59                  src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
60                  rotation_direction = RotationDirection(letters[1])
61
62                  return move_cls(move_type, src_bitboard, src_bitboard,
    rotation_direction)
63              else:
64                  raise ValueError('(Move.instance_from_notation) Invalid move type:
    ', move_type)
65
66          except Exception as error:
67              logger.info('(Move.instance_from_notation) Error occured while parsing
    :', error)
68              raise error
69
70      @classmethod
71      def instance_from_input(move_cls, move_type, src, dest=None, rotation=None):
72          try:
73              if move_type == MoveType.MOVE:
74                  src_bitboard = notation_to_bitboard(src)
75                  dest_bitboard = notation_to_bitboard(dest)
76
77              elif move_type == MoveType.ROTATE:
78                  src_bitboard = notation_to_bitboard(src)
79                  dest_bitboard = src_bitboard
80
81              return move_cls(move_type, src_bitboard, dest_bitboard, rotation)
82          except Exception as error:
83              logger.info('Error (Move.instance_from):', error)
84              raise error
85
```

```
86      @classmethod
87      def instance_from_coords(move_cls, move_type, src_coords, dest_coords=None,
        rotation_direction=None):
88          try:
89              src_bitboard = coords_to_bitboard(src_coords)
90              dest_bitboard = coords_to_bitboard(dest_coords)
91
92              return move_cls(move_type, src_bitboard, dest_bitboard,
        rotation_direction)
93          except Exception as error:
94              logger.info('Error (Move.instance_from_coords):', error)
95              raise error
96
97      @classmethod
98      def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
        None, rotation_direction=None):
99          try:
100             return move_cls(move_type, src_bitboard, dest_bitboard,
        rotation_direction)
101         except Exception as error:
102             logger.info('Error (Move.instance_from_bitboards):', error)
103             raise error
```

1.17.9 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,
        create_square_overlay, create_circle_overlay
4
5   class OverlayDraw:
6       def __init__(self, board_position, board_size, limit_hover=True):
7           self._board_position = board_position
8           self._board_size = board_size
9
10          self._hovered_coords = None
11          self._selected_coords = None
12          self._available_coords = None
13
14          self._limit_hover = limit_hover
15
16          self._selected_overlay = None
17          self._hovered_overlay = None
18          self._available_overlay = None
19
20          self.initialise_overlay_surfaces()
21
22      @property
23      def square_size(self):
24          return self._board_size[0] / 10
25
26      def initialise_overlay_surfaces(self):
27          self._selected_overlay = create_square_overlay(self.square_size,
        OVERLAY_COLOUR_DARK)
28          self._hovered_overlay = create_square_overlay(self.square_size,
        OVERLAY_COLOUR_LIGHT)
29          self._available_overlay = create_circle_overlay(self.square_size,
        OVERLAY_COLOUR_LIGHT)
30
31      def set_hovered_coords(self, mouse_pos):
32          self._hovered_coords = screen_pos_to_coords(mouse_pos, self.
        _board_position, self._board_size)
```

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

1.17.10 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
```

```
            if piece_and_rotation:
                if piece_and_rotation[0].isupper():
                    colour = Colour.BLUE
                else:
                    colour = Colour.RED

                piece = PieceSprite(piece=Piece(piece_and_rotation[0].lower()), colour=colour, rotation=piece_and_rotation[1])
                piece.set_coords((x, y))
                piece.set_geometry(board_position, board_size[0] / 10)
                piece.set_image()
                self.add(piece)

    def set_geometry(self, board_position, board_size):
        for sprite in self.sprites():
            sprite.set_geometry(board_position, board_size[0] / 10)

    def handle_resize(self, board_position, board_size):
        self.set_geometry(board_position, board_size)

        for sprite in self.sprites():
            sprite.set_image()

    def remove_piece(self, coords):
        for sprite in self.sprites():
            if sprite.coords == coords:
                sprite.kill()
```

1.17.12 piece_sprite.py

```
import pygame
from data.helpers.board_helpers import coords_to_screen_pos
from data.helpers.asset_helpers import scale_and_cache
from data.utils.assets import GRAPHICS
from data.utils.enums import Piece

class PieceSprite(pygame.sprite.Sprite):
    def __init__(self, piece, colour, rotation):
        super().__init__()
        self.colour = colour
        self.rotation = rotation

        self.type = piece
        self.coords = None
        self.size = None

    @property
    def image_name(self):
        return Piece(self.type).name.lower() + '_' + str(self.colour) + '_' + self.rotation

    def set_image(self):
        self.image = scale_and_cache(GRAPHICS[self.image_name], (self.size, self.size))

    def set_geometry(self, new_position, square_size):
        self.size = square_size
        self.rect = pygame.FRect((0, 0, square_size, square_size))

        if self.coords:
            self.rect.topleft = coords_to_screen_pos(self.coords, new_position, square_size)
```

```
        else:
            self.rect.topleft = new_position

    def set_coords(self, new_coords):
        self.coords = new_coords
```

1.17.13 psqt.py

```
from data.utils.enums import Piece

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

PSQT = {
    Piece.PYRAMID: [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    ],
    Piece.ANUBIS: [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    ],
    Piece.SCARAB: [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
        0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
        0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
        0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    ],
    Piece.PHARAOH: [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 2, 2, 2, 2, 0, 0, 0,
        0, 0, 0, 2, 4, 4, 2, 0, 0, 0,
    ],
```

```
}
```

1.18 data\states\game\cpu

1.18.1 base.py

```
import time
from pprint import PrettyPrinter
from data.utils.enums import Colour, Score, Miscellaneous
from data.states.game.cpu.evaluator import Evaluator
from data.managers.logs import initialise_logger

logger = initialise_logger(__name__)
printer = PrettyPrinter(indent=2, sort_dicts=False)

class BaseCPU:
    def __init__(self, callback, verbose=True):
        self._evaluator = Evaluator(verbose=False)
        self._verbose = verbose
        self._callback = callback
        self._stats = {}

    def initialise_stats(self):
        self._stats = {
            'nodes': 0,
            'leaf_nodes' : 0,
            'draws': 0,
            'mates': 0,
            'ms_per_node': 0,
            'time_taken': time.time()
        }

    def print_stats(self, score, move):
        """
        Prints statistics after traversing tree.

        Args:
            score (int): Final score obtained after traversal.
            move (Move): Best move obtained after traversal.
        """
        if self._verbose is False:
            return

        self._stats['time_taken'] = round(1000 * (time.time() - self._stats['
    time_taken']), 3)
        self._stats['ms_per_node'] = round(self._stats['time_taken'] / self._stats
    ['nodes'], 3)

        # Prints stats across multiple lines
        if self._verbose is True:
            logger.info(f'\n\n'
                        f'{self.__str__()} Search Results:\n'
                        f'{printer.pformat(self._stats)}\n'
                        f'Best score:  {score}   Best move: {move}\n'
                        )

        # Prints stats in a compacted format
        elif self._verbose.lower() == 'compact':
            logger.info(self._stats)
            logger.info(f'Best score: {score}   Best move: {move}')

```

```
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         self._cache[hashed] = score
14
15         return score
```

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

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
       NoneType I think (just deal with None being returned)
12         try:
13             best_move = self.search(board, self._max_depth, -Score.INFINITE, Score
       .INFINITE, stop_event)
14
15             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 occured:')
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
```

```
from data.states.game.cpu.engines.iterative_deepening import
    IterativeDeepeningMixin
from data.states.game.cpu.base import BaseCPU
from data.utils.enums import Score

class NegamaxCPU(BaseCPU):
    def __init__(self, max_depth, callback, verbose=False):
        super().__init__(callback, verbose)
        self._max_depth = max_depth

    def find_move(self, board, stop_event):
        self.initialise_stats()
        best_score, best_move = self.search(board, self._max_depth, stop_event)

        if self._verbose:
            self.print_stats(best_score, best_move)

        self._callback(best_move)

    def search(self, board, depth, stop_event, moves=None):
        if (base_case := super().search(board, depth, stop_event, absolute=True)):
            return base_case

        best_move = None
        best_score = -Score.INFINITE

        for move in board.generate_all_moves(board.get_active_colour()):
            laser_result = board.apply_move(move)

            new_score = self.search(board, depth - 1, stop_event)[0]
            new_score = -new_score

            if new_score > best_score:
                best_score = new_score
                best_move = move
            elif new_score == best_score:
                best_move = choice([best_move, move])

            board.undo_move(move, laser_result)

        return best_score, best_move

class ABNegamaxCPU(BaseCPU):
    def __init__(self, max_depth, callback, verbose=True):
        super().__init__(callback, verbose)
        self._max_depth = max_depth

    def initialise_stats(self):
        """Initialises the statistics for the search."""
        super().initialise_stats()
        self._stats['beta_prunes'] = 0

    def find_move(self, board, stop_event):
        """Finds the best move for the current board state.

        Args:
            board (Board): The current board state.
            stop_event (threading.Event): The event to signal stopping the search.
        """
        self.initialise_stats()
        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
```

```
            if (active_colour == Colour.BLUE and score > best_score) or (
    active_colour == Colour.RED and score < best_score):
                best_move = move
                best_score = score

            board.undo_move(move, laser_result)

        return best_score, best_move
```

1.19.6 transposition_table.py

See Section ??.

1.19.7 __init__.py

```
from data.states.game.cpu.engines.simple import SimpleCPU
from data.states.game.cpu.engines.negamax import NegamaxCPU
from data.states.game.cpu.engines.minimax import MinimaxCPU
from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU
from data.states.game.cpu.engines.iterative_deepening import IDMinimaxCPU
from data.states.game.cpu.engines.transposition_table import TTMinimaxCPU
```

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

```
import pygame
from data.states.game.widget_dict import PAUSE_WIDGETS
from data.components.widget_group import WidgetGroup
from data.utils.event_types import GameEventType
from data.utils.constants import PAUSE_COLOUR
from data.managers.window import window
from data.managers.audio import audio

class PauseView:
    def __init__(self, model):
        self._model = model

        self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
        self._screen_overlay.fill(PAUSE_COLOUR)

        self._widget_group = WidgetGroup(PAUSE_WIDGETS)
        self._widget_group.handle_resize(window.size)

```

```
        self._model.register_listener(self.process_model_event, 'pause')

        self._event_to_func_map = {
            GameEventType.PAUSE_CLICK: self.handle_pause_click
        }

        self.states = {
            'PAUSED': False
        }

    def handle_pause_click(self, event):
        self.states['PAUSED'] = not self.states['PAUSED']

        if self.states['PAUSED']:
            audio.pause_sfx()
        else:
            audio.unpause_sfx()

    def handle_resize(self):
        self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
        self._screen_overlay.fill(PAUSE_COLOUR)
        self._widget_group.handle_resize(window.size)

    def draw(self):
        if self.states['PAUSED']:
            window.screen.blit(self._screen_overlay, (0, 0))
            self._widget_group.draw()

    def process_model_event(self, event):
        try:
            self._event_to_func_map.get(event.type)(event)
        except:
            raise KeyError('Event type not recognized in Paused View (PauseView.
    process_model_event)', event)

    def convert_mouse_pos(self, event):
        return self._widget_group.process_event(event)
```

1.20.5 win_view.py

```
from data.utils.enums import Colour, Miscellaneous, CursorMode
from data.components.widget_group import WidgetGroup
from data.states.game.widget_dict import WIN_WIDGETS
from data.managers.window import window
from data.managers.cursor import cursor

class WinView:
    def __init__(self, model):
        self._model = model

        self._widget_group = WidgetGroup(WIN_WIDGETS)
        self._widget_group.handle_resize(window.size)

    def handle_resize(self):
        self._widget_group.handle_resize(window.size)

    def draw(self):
        if self._model.states['WINNER'] is not None:
            if cursor.get_mode() != CursorMode.ARROW:
                cursor.set_mode(CursorMode.ARROW)

            if self._model.states['WINNER'] == Colour.BLUE:
```

```
                WIN_WIDGETS['red_won'].kill()
                WIN_WIDGETS['draw_won'].kill()
            elif self._model.states['WINNER'] == Colour.RED:
                WIN_WIDGETS['blue_won'].kill()
                WIN_WIDGETS['draw_won'].kill()
            elif self._model.states['WINNER'] == Miscellaneous.DRAW:
                WIN_WIDGETS['red_won'].kill()
                WIN_WIDGETS['blue_won'].kill()

            self._widget_group.draw()

    def set_win_type(self, win_type):
        WIN_WIDGETS['by_draw'].kill()
        WIN_WIDGETS['by_timeout'].kill()
        WIN_WIDGETS['by_resignation'].kill()
        WIN_WIDGETS['by_checkmate'].kill()

        match win_type:
            case 'CAPTURE':
                self._widget_group.add(WIN_WIDGETS['by_checkmate'])
            case 'DRAW':
                self._widget_group.add(WIN_WIDGETS['by_draw'])
            case 'RESIGN':
                self._widget_group.add(WIN_WIDGETS['by_resignation'])
            case 'TIME':
                self._widget_group.add(WIN_WIDGETS['by_timeout'])

    def convert_mouse_pos(self, event):
        return self._widget_group.process_event(event)
```

1.21 data\states\menu

1.21.1 menu.py

```
import pygame
import sys
from random import randint
from data.helpers.asset_helpers import get_rotational_angle
from data.helpers.asset_helpers import scale_and_cache
from data.states.menu.widget_dict import MENU_WIDGETS
from data.utils.assets import GRAPHICS, MUSIC, SFX
from data.managers.logs import initialise_logger
from data.utils.event_types import MenuEventType
from data.managers.animation import animation
from data.utils.constants import ShaderType
from data.managers.window import window
from data.managers.audio import audio
from data.control import _State

logger = initialise_logger(__file__)

class Menu(_State):
    def __init__(self):
        super().__init__()
        self._fire_laser = False
        self._bloom_mask = None
        self._laser_mask = None

    def cleanup(self):
        super().cleanup()

```

```
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.MOUSEBUTTONUP, 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 of sfx 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.FRect(0, 0, *self.sphinx_size)
107         sphinx_rect.center = self.sphinx_center
108
109         window.screen.blit(sphinx_surface, sphinx_rect)
110
111     def update_masks(self):
112         self.draw()
113
114         widget_mask = window.screen.copy()
115         laser_mask = pygame.mask.from_surface(widget_mask)
116         laser_mask = laser_mask.to_surface(setcolor=(255, 0, 0, 255), unsetcolor
    =(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,
```

```
                (190, 190, 255),
                0.99,
                (self.sphinx_rotation - 2 + random_offset(), self.sphinx_rotation
    + 2 + random_offset())
            ]])

            window.set_effect(ShaderType.SHAKE)
            window.set_apply_arguments(ShaderType.SHAKE, intensity=1)
            pygame.mouse.set_pos(pygame.mouse.get_pos()[0] + random_offset(),
    pygame.mouse.get_pos()[1] + random_offset())

        super().update(**kwargs)

    def handle_resize(self):
        super().handle_resize()
        self.update_masks()
```

1.21.2 widget_dict.py

```
from data.components.custom_event import CustomEvent
from data.utils.event_types import MenuEventType
from data.utils.assets import GRAPHICS
from data.managers.theme import theme
from data.widgets import *

top_right_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.15, 0.075),
    fixed_position=(5, 5),
    anchor_x='right',
    scale_mode='height'
)

MENU_WIDGETS = {
    'credits':
    Icon(
        relative_position=(0, 0),
        relative_size=(0.7, 0.7),
        icon=GRAPHICS['credits'],
        anchor_x='center',
        anchor_y='center',
        margin=50
    ),
    'default': [
        top_right_container,
        Rectangle(
            relative_position=(0.65, 0.15),
            relative_size=(0.15, 0.15),
            scale_mode='height',
            border_width=0,
            border_radius=50,
            fill_colour=theme['fillSecondary'],
            visible=True
        ),
        Rectangle(
            relative_position=(0.8, 0.1),
            relative_size=(0.10, 0.10),
            scale_mode='height',
            border_width=0,
            border_radius=100,
            fill_colour=theme['fillSecondary'],
            visible=True
```

```
        ),
        Rectangle(
            relative_position=(0.5, 0.1),
            relative_size=(0.20, 0.20),
            scale_mode='height',
            border_width=0,
            border_radius=10,
            fill_colour=theme['fillSecondary'],
            visible=True
        ),
        Rectangle(
            relative_position=(0.9, 0.2),
            relative_size=(0.15, 0.15),
            scale_mode='height',
            border_width=0,
            border_radius=20,
            fill_colour=theme['fillSecondary'],
            visible=True
        ),
        Rectangle(
            relative_position=(0.85, 0.4),
            relative_size=(0.20, 0.20),
            scale_mode='height',
            border_width=0,
            border_radius=30,
            fill_colour=theme['fillSecondary'],
            visible=True
        ),
        Rectangle(
            relative_position=(0.7, 0.4),
            relative_size=(0.10, 0.10),
            scale_mode='height',
            border_width=0,
            border_radius=50,
            fill_colour=theme['fillSecondary'],
            visible=True
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            anchor_x='right',
            scale_mode='height',
            base_icon=GRAPHICS['quit_base'],
            hover_icon=GRAPHICS['quit_hover'],
            press_icon=GRAPHICS['quit_press'],
            event=CustomEvent(MenuEventType.QUIT_CLICK)
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['credits_base'],
            hover_icon=GRAPHICS['credits_hover'],
            press_icon=GRAPHICS['credits_press'],
            event=CustomEvent(MenuEventType.CREDITS_CLICK)
        ),
        ReactiveIconButton(
            relative_position=(0.05, -0.2),
            relative_size=(0, 0.15),
            anchor_y='center',
```

```
            base_icon=GRAPHICS['play_text_base'],
            hover_icon=GRAPHICS['play_text_hover'],
            press_icon=GRAPHICS['play_text_press'],
            event=CustomEvent(MenuEventType.CONFIG_CLICK)
        ),
        ReactiveIconButton(
            relative_position=(0.05, 0),
            relative_size=(0, 0.15),
            anchor_y='center',
            base_icon=GRAPHICS['review_text_base'],
            hover_icon=GRAPHICS['review_text_hover'],
            press_icon=GRAPHICS['review_text_press'],
            event=CustomEvent(MenuEventType.BROWSER_CLICK)
        ),
        ReactiveIconButton(
            relative_position=(0.05, 0.2),
            relative_size=(0, 0.15),
            anchor_y='center',
            base_icon=GRAPHICS['settings_text_base'],
            hover_icon=GRAPHICS['settings_text_hover'],
            press_icon=GRAPHICS['settings_text_press'],
            event=CustomEvent(MenuEventType.SETTINGS_CLICK)
        ),
        # Icon(
        #     relative_position=(0.0, 0.1),
        #     relative_size=(0.3, 0.2),
        #     anchor_x='center',
        #     fill_colour=theme['fillSecondary'],
        #     icon=GRAPHICS['title_screen_art'],
        #     stretch=False
        # ),
    ]
}

# Widgets used for testing light rays effect
TEST_WIDGETS = {
    'default': [
        Rectangle(
            relative_position=(0.4, 0.2),
            relative_size=(0.1, 0.1),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.5, 0.7),
            relative_size=(0.1, 0.1),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.6, 0.6),
            relative_size=(0.2, 0.2),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
```

```
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.4, 0.6),
            relative_size=(0.1, 0.1),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.6, 0.4),
            relative_size=(0.1, 0.1),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.3, 0.4),
            relative_size=(0.1, 0.1),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.475, 0.15),
            relative_size=(0.2, 0.2),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        ),
        Rectangle(
            relative_position=(0.6, 0.2),
            relative_size=(0.1, 0.1),
            scale_mode='height',
            visible=True,
            border_width=0,
            fill_colour=(255, 0, 0),
            border_radius=1000
        )
    ]
}
```

1.22 data\states\review

1.22.1 review.py

See Section ??.

1.22.2 widget_dict.py

```
from data.widgets import *
from data.components.custom_event import CustomEvent
```

```
from data.utils.event_types import ReviewEventType
from data.utils.assets import GRAPHICS
from data.utils.enums import Colour

MOVE_LIST_WIDTH = 0.2

right_container = Rectangle(
    relative_position=(0.05, 0),
    relative_size=(0.2, 0.7),
    anchor_y='center',
    anchor_x='right'
)

info_container = Rectangle(
    parent=right_container,
    relative_position=(0, 0.5),
    relative_size=(1, 0.5),
    visible=True
)

arrow_container = Rectangle(
    relative_position=(0, 0.05),
    relative_size=(0.4, 0.1),
    anchor_x='center',
    anchor_y='bottom'
)

move_list = MoveList(
    parent=right_container,
    relative_position=(0, 0),
    relative_width=1,
    minimum_height=300,
    move_list=[]
)

top_right_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.15, 0.075),
    fixed_position=(5, 5),
    anchor_x='right',
    scale_mode='height'
)

REVIEW_WIDGETS = {
    'help':
    Icon(
        relative_position=(0, 0),
        relative_size=(1.02, 1.02),
        icon=GRAPHICS['review_help'],
        anchor_x='center',
        anchor_y='center',
        border_width=0,
        fill_colour=(0, 0, 0, 0)
    ),
    'default': [
        arrow_container,
        right_container,
        info_container,
        top_right_container,
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
```

```
            relative_size=(1, 1),
            anchor_x='right',
            scale_mode='height',
            base_icon=GRAPHICS['home_base'],
            hover_icon=GRAPHICS['home_hover'],
            press_icon=GRAPHICS['home_press'],
            event=CustomEvent(ReviewEventType.MENU_CLICK)
        ),
        ReactiveIconButton(
            parent=top_right_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['help_base'],
            hover_icon=GRAPHICS['help_hover'],
            press_icon=GRAPHICS['help_press'],
            event=CustomEvent(ReviewEventType.HELP_CLICK)
        ),
        ReactiveIconButton(
            parent=arrow_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['left_arrow_filled_base'],
            hover_icon=GRAPHICS['left_arrow_filled_hover'],
            press_icon=GRAPHICS['left_arrow_filled_press'],
            event=CustomEvent(ReviewEventType.PREVIOUS_CLICK)
        ),
        ReactiveIconButton(
            parent=arrow_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            anchor_x='right',
            base_icon=GRAPHICS['right_arrow_filled_base'],
            hover_icon=GRAPHICS['right_arrow_filled_hover'],
            press_icon=GRAPHICS['right_arrow_filled_press'],
            event=CustomEvent(ReviewEventType.NEXT_CLICK)
        ),
    ],
    'move_list':
        move_list,
    'scroll_area':
    ScrollArea(
        parent=right_container,
        relative_position=(0, 0),
        relative_size=(1, 0.5),
        vertical=True,
        widget=move_list
    ),
    'chessboard':
    Chessboard(
        relative_position=(0, 0),
        relative_width=0.4,
        scale_mode='width',
        anchor_x='center',
        anchor_y='center'
    ),
    'move_number_text':
    Text(
        parent=info_container,
        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(
```

```
        relative_position=(0.05, 0.05),
        relative_size=(0.2, 0.1),
        active_colour=Colour.RED
    ),
}
```

1.23 data\states\settings

1.23.1 settings.py

```
import pygame
from random import randint
from data.helpers.data_helpers import get_default_settings, get_user_settings,
    update_user_settings
from data.utils.constants import WidgetState, ShaderType, SHADER_MAP
from data.states.settings.widget_dict import SETTINGS_WIDGETS
from data.utils.event_types import SettingsEventType
from data.managers.logs import initialise_logger
from data.managers.window import window
from data.managers.audio import audio
from data.widgets import ColourPicker
from data.utils.assets import MUSIC
from data.control import _State

logger = initialise_logger(__name__)

class Settings(_State):
    def __init__(self):
        super().__init__()

        self._colour_picker = None
        self._settings = None

    def cleanup(self):
        super().cleanup()

        update_user_settings(self._settings)

        return None

    def startup(self, persist=None):
        super().startup(SETTINGS_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])

        window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
    BACKGROUND_BALATRO)
        self._settings = get_user_settings()
        self.reload_settings()

        self.draw()

    def create_colour_picker(self, mouse_pos, button_type):
        if button_type == SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK:
            selected_colour = self._settings['primaryBoardColour']
            event_type = SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK
        else:
            selected_colour = self._settings['secondaryBoardColour']
            event_type = SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK

        self._colour_picker = ColourPicker(
            relative_position=(mouse_pos[0] / window.size[0], mouse_pos[1] /
    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.
    mouse.get_pos()[1] / window.size[1])
60
61          if self._settings['displayMode'] == 'fullscreen':
62              window.set_fullscreen(desktop=True)
63              window.handle_resize()
64
65          elif self._settings['displayMode'] == 'windowed':
66              window.set_windowed()
67              window.handle_resize()
68              window.restore()
69
70          self._widget_group.handle_resize(window.size)
71
72          new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
    relative_mouse_pos[1] * window.size[1])
73          pygame.mouse.set_pos(new_mouse_pos)
74
75      def reload_shaders(self):
76          window.clear_all_effects()
77
78          for shader_type in SHADER_MAP[self._settings['shader']]:
79              window.set_effect(shader_type)
80
81      def reload_settings(self):
82          SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self.
    _settings['primaryBoardColour'])
83          SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self.
    _settings['secondaryBoardColour'])
84          SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.
    BASE)
85          SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.
    BASE)
86          SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['
    musicVolume'])
87          SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume
    '])
88          SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings
    ['displayMode'])
89          SETTINGS_WIDGETS['shader_carousel'].set_to_key(self._settings['shader'])
90          SETTINGS_WIDGETS['particles_switch'].set_toggle_state(self._settings['
    particles'])
91          SETTINGS_WIDGETS['opengl_switch'].set_toggle_state(self._settings['opengl'
    ])
92
93          self.reload_shaders()
94          self.reload_display_mode()
95
96      def get_event(self, event):
97          widget_event = self._widget_group.process_event(event)
98
99          if widget_event is None:
```

```
100             if event.type == pygame.MOUSEBUTTONDOWN and self._colour_picker:
101                 self.remove_colour_picker()
102             return
103 
104         match widget_event.type:
105             case SettingsEventType.VOLUME_SLIDER_SLIDE:
106                 return
107 
108             case SettingsEventType.VOLUME_SLIDER_CLICK:
109                 if widget_event.volume_type == 'music':
110                     audio.set_music_volume(widget_event.volume)
111                     self._settings['musicVolume'] = widget_event.volume
112                 elif widget_event.volume_type == 'sfx':
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),
```

```
            anchor_x='right',
            scale_mode='height',
            base_icon=GRAPHICS['home_base'],
            hover_icon=GRAPHICS['home_hover'],
            press_icon=GRAPHICS['home_press'],
            fixed_position=(5, 5),
            event=CustomEvent(SettingsEventType.MENU_CLICK)
        ),
        Text(
            relative_position=(0.01, 0.1),
            text='Display mode',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.2),
            text='Music',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.3),
            text='SFX',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.4),
            text='Primary board colour',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.5),
            text='Secondary board colour',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.6),
            text='Particles',
            relative_size=(0.4, 0.04),
```

```
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.7),
            text='Shaders (OPENGL GPU REQUIRED)',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        Text(
            relative_position=(0.01, 0.8),
            text='Super Secret Settings',
            relative_size=(0.4, 0.04),
            center=False,
            border_width=0,
            margin=0,
            font_size=21,
            fill_colour=(0, 0, 0, 0)
        ),
        TextButton(
            parent=reset_container,
            relative_position=(0, 0),
            relative_size=(1, 0.5),
            fit_vertical=False,
            margin=10,
            text='DISCARD CHANGES',
            text_colour=theme['textSecondary'],
            event=CustomEvent(SettingsEventType.RESET_USER)
        ),
        TextButton(
            parent=reset_container,
            relative_position=(0, 0.5),
            relative_size=(1, 0.5),
            fit_vertical=False,
            margin=10,
            text='RESET TO DEFAULT',
            text_colour=theme['textSecondary'],
            event=CustomEvent(SettingsEventType.RESET_DEFAULT)
        )
    ],
    'display_mode_dropdown':
    Dropdown(
        relative_position=(0.4, 0.1),
        relative_width=0.2,
        word_list=['fullscreen', 'windowed'],
        fill_colour=(255, 100, 100),
        event=CustomEvent(SettingsEventType.DROPDOWN_CLICK)
    ),
    'primary_colour_button':
    ColourButton(
        relative_position=(0.4, 0.4),
        relative_size=(0.08, 0.05),
        fill_colour=user_settings['primaryBoardColour'],
        border_width=5,
        event=CustomEvent(SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK)
```

```
    ),
    'secondary_colour_button':
    ColourButton(
        relative_position=(0.4, 0.5),
        relative_size=(0.08, 0.05),
        fill_colour=user_settings['secondaryBoardColour'],
        border_width=5,
        event=CustomEvent(SettingsEventType.SECONDARY_COLOUR_BUTTON_CLICK)
    ),
    'music_volume_slider':
    VolumeSlider(
        relative_position=(0.4, 0.2),
        relative_length=(0.5),
        default_volume=user_settings['musicVolume'],
        border_width=5,
        volume_type='music'
    ),
    'sfx_volume_slider':
    VolumeSlider(
        relative_position=(0.4, 0.3),
        relative_length=(0.5),
        default_volume=user_settings['sfxVolume'],
        border_width=5,
        volume_type='sfx'
    ),
    'shader_carousel':
    Carousel(
        relative_position = (0.4, 0.8),
        margin=5,
        border_width=0,
        fill_colour=(0, 0, 0, 0),
        widgets_dict=carousel_widgets,
        event=CustomEvent(SettingsEventType.SHADER_PICKER_CLICK),
    ),
    'particles_switch':
    Switch(
        relative_position=(0.4, 0.6),
        relative_height=0.04,
        event=CustomEvent(SettingsEventType.PARTICLES_CLICK)
    ),
    'opengl_switch':
    Switch(
        relative_position=(0.4, 0.7),
        relative_height=0.04,
        event=CustomEvent(SettingsEventType.OPENGL_CLICK)
    ),
}
```

1.24 data\utils

1.24.1 assets.py

```
from pathlib import Path
from data.helpers.load_helpers import *

module_path = Path(__file__).parent
GRAPHICS = load_all_gfx((module_path / '../../resources/graphics').resolve())
FONTS = load_all_fonts((module_path / '../../resources/fonts').resolve())
SFX = load_all_sfx((module_path / '../../resources/sfx').resolve())
MUSIC = load_all_music((module_path / '../../resources/music').resolve())

```

```
DEFAULT_FONT = FONTS['vhs-gothic']
DEFAULT_FONT.strong = True
DEFAULT_FONT.strength = 0.05
```

1.24.2 constants.py

```
import pygame
from data.utils.enums import ShaderType, WidgetState

SCREEN_SIZE = (1600, 800)

PAUSE_COLOUR = (50, 50, 50, 128)
OVERLAY_COLOUR_LIGHT = (*pygame.Color('0xf14e52').rgb, 128)
OVERLAY_COLOUR_DARK = (*pygame.Color('0x9b222b').rgb, 192)

EMPTY_BB = 0
A_FILE_MASK = 0
    b1111111110111111111011111111101111111110111111111011111111101111111110111111111011111111101111111110

J_FILE_MASK = 0
    b0111111111011111111101111111110111111111011111111101111111110111111111011111111101111111110111111111

ONE_RANK_MASK = 0
    b1111111111111111111111111111111111111111111111111111111111111111111111111111111111110000000000

EIGHT_RANK_MASK = 0
    b0000000000111111111111111111111111111111111111111111111111111111111111111111111111111111111111

TEST_MASK = 0
    b0000000010000000001000000000000000000000000000000000000000000000010000000001000000000


GAMES_PER_PAGE = 10

SHADER_MAP = {
    'default': [
        ShaderType.BLOOM
    ],
    'retro': [
        ShaderType.CRT
    ],
    'really_retro': [
        ShaderType.CRT,
        ShaderType.GRAYSCALE
    ],
}

BLUE_BUTTON_COLOURS = {
    WidgetState.BASE: ['0x1c2638', '0x23495d', '0x39707a', '0x95e0cc'],
    WidgetState.HOVER: ['0xdaf2e9', '0x23495d', '0x39707a', '0x95e0cc'],
    WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
}

INPUT_COLOURS = {
    WidgetState.BASE: ['0x1c2638', '0x39707a', '0x23495d', '0x95e0cc'],
    WidgetState.HOVER: ['0xdaf2e9', '0x39707a', '0x23495d', '0x95e0cc'],
    WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0x1c2638', '0x39707a']
}

RED_BUTTON_COLOURS = {
    WidgetState.BASE: ['0x000000', '0x1c2638', '0x9b222b', '0xf14e52'],
    WidgetState.HOVER: ['0xdaf2e9', '0x1c2638', '0x9b222b', '0xf14e52'],
```

```
    WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0xf14e52', '0x95e0cc']
}

LOCKED_RED_BUTTON_COLOURS = {
    WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
    WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
    WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0xf14e52']
}

LOCKED_BLUE_BUTTON_COLOURS = {
    WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
    WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
    WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
}
```

1.24.3 enums.py

```
from enum import IntEnum, StrEnum, auto

class CursorMode(IntEnum):
    ARROW = auto()
    IBEAM = auto()
    OPENHAND = auto()
    CLOSEDHAND = auto()
    NO = auto()

class ShaderType(StrEnum):
    BASE = auto()
    SHAKE = auto()
    BLOOM = auto()
    GRAYSCALE = auto()
    CRT = auto()
    RAYS = auto()
    CHROMATIC_ABBREVIATION = auto()
    BACKGROUND_WAVES = auto()
    BACKGROUND_BALATRO = auto()
    BACKGROUND_LASERS = auto()
    BACKGROUND_GRADIENT = auto()
    BACKGROUND_NONE = auto()

    _BLUR = auto()
    _HIGHLIGHT_BRIGHTNESS = auto()
    _HIGHLIGHT_COLOUR = auto()
    _CALIBRATE = auto()
    _LIGHTMAP = auto()
    _SHADOWMAP = auto()
    _OCCLUSION = auto()
    _BLEND = auto()
    _CROP = auto()

class TranspositionFlag(StrEnum):
    LOWER = auto()
    EXACT = auto()
    UPPER = auto()

class Miscellaneous(StrEnum):
    PLACEHOLDER = auto()
    DRAW = auto()

class WidgetState(StrEnum):
    BASE = auto()
    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      PHARAOH = 'f'
70
71  class Score(IntEnum):
72      PHARAOH = 0
73      SPHINX = 0
74      PYRAMID = 100
75      ANUBIS = 110
76      SCARAB = 200
77
78      MOVE = 4
79      POSITION = 11
80      PHARAOH_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
```

```
    FROM_RIGHT = 2
    FROM_BOTTOM = 3
    FROM_LEFT = 4
```

1.24.4 event_types.py

```
from enum import StrEnum, auto

class EditorEventType(StrEnum):
    MENU_CLICK = auto()
    PICK_PIECE_CLICK = auto()
    ROTATE_PIECE_CLICK = auto()
    COPY_CLICK = auto()
    EMPTY_CLICK = auto()
    RESET_CLICK = auto()
    BLUE_START_CLICK = auto()
    RED_START_CLICK = auto()
    START_CLICK = auto()
    CONFIG_CLICK = auto()
    ERASE_CLICK = auto()
    MOVE_CLICK = auto()
    HELP_CLICK = auto()

class ReviewEventType(StrEnum):
    MENU_CLICK = auto()
    PREVIOUS_CLICK = auto()
    NEXT_CLICK = auto()
    HELP_CLICK = auto()

class BrowserEventType(StrEnum):
    MENU_CLICK = auto()
    BROWSER_STRIP_CLICK = auto()
    COPY_CLICK = auto()
    DELETE_CLICK = auto()
    REVIEW_CLICK = auto()
    FILTER_COLUMN_CLICK = auto()
    FILTER_ASCEND_CLICK = auto()
    PAGE_CLICK = auto()
    HELP_CLICK = auto()

class GameEventType(StrEnum):
    BOARD_CLICK = auto()
    PIECE_CLICK = auto()
    PAUSE_CLICK = auto()
    MENU_CLICK = auto()
    GAME_CLICK = auto()
    HELP_CLICK = auto()
    TUTORIAL_CLICK = auto()
    RESIGN_CLICK = auto()
    DRAW_CLICK = auto()
    REVIEW_CLICK = auto()
    PIECE_DROP = auto()
    UPDATE_PIECES = auto()
    ROTATE_PIECE = auto()
    SET_LASER = auto()
    TIMER_END = auto()

class MenuEventType(StrEnum):
    CONFIG_CLICK = auto()
    SETTINGS_CLICK = auto()
    BROWSER_CLICK = auto()
    QUIT_CLICK = auto()
```

```
    CREDITS_CLICK = auto()

class SettingsEventType(StrEnum):
    RESET_DEFAULT = auto()
    RESET_USER = auto()
    MENU_CLICK = auto()
    COLOUR_SLIDER_SLIDE = auto()
    COLOUR_SLIDER_CLICK = auto()
    COLOUR_PICKER_HOVER = auto()
    PRIMARY_COLOUR_PICKER_CLICK = auto()
    SECONDARY_COLOUR_PICKER_CLICK = auto()
    PRIMARY_COLOUR_BUTTON_CLICK = auto()
    SECONDARY_COLOUR_BUTTON_CLICK = auto()
    VOLUME_SLIDER_SLIDE = auto()
    VOLUME_SLIDER_CLICK = auto()
    SHADER_PICKER_CLICK = auto()
    OPENGL_CLICK = auto()
    DROPDOWN_CLICK = auto()
    PARTICLES_CLICK = auto()

class ConfigEventType(StrEnum):
    GAME_CLICK = auto()
    MENU_CLICK = auto()
    FEN_STRING_TYPE = auto()
    TIME_TYPE = auto()
    TIME_CLICK = auto()
    PVP_CLICK = auto()
    PVC_CLICK = auto()
    CPU_DEPTH_CLICK = auto()
    PRESET_CLICK = auto()
    SETUP_CLICK = auto()
    COLOUR_CLICK = auto()
    HELP_CLICK = auto()
```

1.25 data\widgets

1.25.1 board_thumbnail.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.chessboard import Chessboard
from data.states.game.components.piece_group import PieceGroup
from data.states.game.components.bitboard_collection import BitboardCollection

class BoardThumbnail(_Widget):
    def __init__(self, relative_width, fen_string='', **kwargs):
        super().__init__(relative_size=(relative_width, relative_width * 0.8), **
    kwargs)

        self._board = Chessboard(
            parent=self._parent,
            relative_position=(0, 0),
            scale_mode=kwargs.get('scale_mode'),
            relative_width=relative_width
        )

        self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)

        self.initialise_board(fen_string)
        self.set_image()
        self.set_geometry()
```

```
23
24      def initialise_board(self, fen_string):
25          if len(fen_string) == 0:
26              piece_list = []
27          else:
28              piece_list = BitboardCollection(fen_string).convert_to_piece_list()
29
30          self._piece_group = PieceGroup()
31          self._piece_group.initialise_pieces(piece_list, (0, 0), self.size)
32
33          self._board.refresh_board()
34          self.set_image()
35
36      def set_image(self):
37          self.image = pygame.transform.scale(self._empty_surface, self.size)
38
39          self._board.set_image()
40          self.image.blit(self._board.image, (0, 0))
41
42          self._piece_group.draw(self.image)
43
44      def set_geometry(self):
45          super().set_geometry()
46          self._board.set_geometry()
47
48      def set_surface_size(self, new_surface_size):
49          super().set_surface_size(new_surface_size)
50          self._board.set_surface_size(new_surface_size)
51          self._piece_group.handle_resize((0, 0), self.size)
52
53      def process_event(self, event):
54          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
```

```
from data.widgets.bases.widget import _Widget

FONT_DIVISION = 7

class BrowserItem(_Widget):
    def __init__(self, relative_width, game, **kwargs):
        super().__init__(relative_size=(relative_width, relative_width * 2), scale_mode='height', **kwargs)

        self._relative_font_size = text_width_to_font_size('YYYY-MM-DD HH:MM:SS', self._font, self.size[0]) / self.surface_size[1]

        self._game = game
        self._board_thumbnail = BoardThumbnail(
            relative_position=(0, 0),
            scale_mode='height',
            relative_width=relative_width,
            fen_string=self._game['final_fen_string']
        )

        self.set_image()
        self.set_geometry()

    def get_text_to_render(self):
        depth_to_text = {
            2: 'EASY',
            3: 'MEDIUM',
            4: 'HARD'
        }

        format_moves = lambda no_of_moves:  int(no_of_moves / 2) if (no_of_moves / 2 % 1 == 0) else round(no_of_moves / 2, 1)

        if self._game['cpu_enabled'] == 1:
            depth_text = depth_to_text[self._game['cpu_depth']]
            cpu_text = f'PVC ({depth_text})'
        else:
            cpu_text = 'PVP'

        return [
            cpu_text,
            self._game['created_dt'].strftime('%Y-%m-%d %H:%M:%S'),
            f'WINNER: {get_winner_string(self._game['winner'])}',
            f'NO. MOVES: {format_moves(self._game['number_of_ply'])}'
        ]

    def set_image(self):
        self.image = pygame.Surface(self.size, pygame.SRCALPHA)
        resized_board = scale_and_cache(self._board_thumbnail.image, (self.size[0], self.size[0] * 0.8))
        self.image.blit(resized_board, (0, 0))

        get_line_y = lambda line: (self.size[0] * 0.8) + ((self.size[0] * 0.8) / FONT_DIVISION) * (line + 0.5)

        text_to_render = self.get_text_to_render()

        for index, text in enumerate(text_to_render):
            self._font.render_to(self.image, (0, get_line_y(index)), text, fgcolor=self._text_colour, size=self.font_size)

    def process_event(self, event):
```

```
        pass
```

1.25.4 browser_strip.py

```
import pygame
from data.components.custom_event import CustomEvent
from data.utils.event_types import BrowserEventType
from data.widgets.browser_item import BrowserItem
from data.widgets.bases.widget import _Widget

WIDTH_FACTOR = 0.3

class BrowserStrip(_Widget):
    def __init__(self, relative_height, games_list, **kwargs):
        super().__init__(relative_size=None, **kwargs)
        self._relative_item_width = relative_height / 2
        self._get_rect = None

        self._games_list = []
        self._items_list = []
        self._selected_index = None

        self.initialise_games_list(games_list)

    @property
    def item_width(self):
        return self._relative_item_width * self.surface_size[1]

    @property
    def size(self):
        if self._get_rect:
            height = self._get_rect().height
        else:
            height = 0
        width = max(0, len(self._games_list) * (self.item_width + self.margin) +
    self.margin)

        return (width, height)

    def register_get_rect(self, get_rect_func):
        self._get_rect = get_rect_func

    def initialise_games_list(self, games_list):
        self._items_list = []
        self._games_list = games_list
        self._selected_index = None

        for game in games_list:
            browser_item = BrowserItem(relative_position=(0, 0), game=game,
    relative_width=self._relative_item_width)
            self._items_list.append(browser_item)

        self.set_image()
        self.set_geometry()

    def set_image(self):
        self.image = pygame.Surface(self.size, pygame.SRCALPHA)
        browser_list = []

        for index, item in enumerate(self._items_list):
            item.set_image()
```

```
56              browser_list.append((item.image, (index * (self.item_width + self.
    margin) + self.margin, self.margin)))
57
58          self.image.blits(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
    [1], max_widget_size[1] / self.surface_size[1])
20          self._relative_size = ((max_widget_size[0] + 2 * (self.margin + self.
    arrow_size[0])) / self.surface_size[1], (max_widget_size[1]) / self.
    surface_size[1])
21
22          self._left_arrow = ReactiveIconButton(
23              relative_position=(0, 0),
24              relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
25              scale_mode='height',
26              base_icon=GRAPHICS['left_arrow_base'],
27              hover_icon=GRAPHICS['left_arrow_hover'],
28              press_icon=GRAPHICS['left_arrow_press'],
29              event=CustomEvent(Miscellaneous.PLACEHOLDER),
30              sfx=SFX['carousel_click']
31          )
32          self._right_arrow = ReactiveIconButton(
33              relative_position=(0, 0),
34              relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
35              scale_mode='height',
36              base_icon=GRAPHICS['right_arrow_base'],
37              hover_icon=GRAPHICS['right_arrow_hover'],
38              press_icon=GRAPHICS['right_arrow_press'],
39              event=CustomEvent(Miscellaneous.PLACEHOLDER),
40              sfx=SFX['carousel_click']
41          )
42
43          self._event = event
44          self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
45
46          self.set_image()
47          self.set_geometry()
48
49      @property
50      def max_widget_size(self):
51          return (self._relative_max_widget_size[0] * self.surface_size[1], self.
    _relative_max_widget_size[1] * self.surface_size[1])
52
53      @property
54      def arrow_size(self):
55          height = self.max_widget_size[1] * 0.75
56          width = (GRAPHICS['left_arrow_base'].width / GRAPHICS['left_arrow_base'].
    height) * height
57          return (width, height)
58
59      @property
60      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

```
import pygame
from data.helpers.data_helpers import get_user_settings
from data.helpers.board_helpers import create_board
from data.widgets.bases.widget import _Widget
from data.utils.enums import CursorMode
from data.managers.cursor import cursor

class Chessboard(_Widget):
    def __init__(self, relative_width, change_cursor=True, **kwargs):
        super().__init__(relative_size=(relative_width, relative_width * 0.8), **kwargs)

        self._board_surface = None
        self._change_cursor = change_cursor
        self._cursor_is_hand = False

        self.refresh_board()
        self.set_image()
        self.set_geometry()

    def refresh_board(self):
        user_settings = get_user_settings()
        self._board_surface = create_board(self.size, user_settings['primaryBoardColour'], user_settings['secondaryBoardColour'])

        self.set_image()

    def set_image(self):
        self.image = pygame.transform.smoothscale(self._board_surface, self.size)

    def process_event(self, event):
        if self._change_cursor and event.type in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN]:
            current_cursor = cursor.get_mode()

            if self.rect.collidepoint(event.pos):
                if current_cursor == CursorMode.ARROW:
                    cursor.set_mode(CursorMode.OPENHAND)
                elif current_cursor == CursorMode.OPENHAND and (pygame.mouse.get_pressed()[0] is True or event.type == pygame.MOUSEBUTTONDOWN):
                    cursor.set_mode(CursorMode.CLOSEDHAND)
                elif current_cursor == CursorMode.CLOSEDHAND and (pygame.mouse.get_pressed()[0] is False or event.type == pygame.MOUSEBUTTONUP):
                    cursor.set_mode(CursorMode.OPENHAND)
            else:
                if current_cursor == CursorMode.OPENHAND or (current_cursor == CursorMode.CLOSEDHAND and event.type == pygame.MOUSEBUTTONUP):
                    cursor.set_mode(CursorMode.ARROW)
```

1.25.7 colour_button.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.bases.pressable import _Pressable
from data.utils.constants import WidgetState

class ColourButton(_Pressable, _Widget):
    def __init__(self, event, **kwargs):
        _Pressable.__init__(
```

```
            self,
            event=event,
            hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
            down_func=lambda: self.set_state_colour(WidgetState.PRESS),
            up_func=lambda: self.set_state_colour(WidgetState.BASE),
            sfx=None
        )
        _Widget.__init__(self, **kwargs)

        self._empty_surface = pygame.Surface(self.size)

        self.initialise_new_colours(self._fill_colour)
        self.set_state_colour(WidgetState.BASE)

        self.set_image()
        self.set_geometry()

    def set_image(self):
        self.image = pygame.transform.scale(self._empty_surface, self.size)
        self.image.fill(self._fill_colour)
        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

```
import pygame
from data.widgets.bases.widget import _Widget

class _ColourDisplay(_Widget):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        self._colour = None

        self._empty_surface = pygame.Surface(self.size)

    def set_colour(self, new_colour):
        self._colour = new_colour

    def set_image(self):
        self.image = pygame.transform.scale(self._empty_surface, self.size)
        self.image.fill(self._colour)

    def process_event(self, event):
        pass
```

1.25.9 colour_picker.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.colour_square import _ColourSquare
from data.widgets.colour_slider import _ColourSlider
from data.widgets.colour_display import _ColourDisplay
from data.components.custom_event import CustomEvent

class ColourPicker(_Widget):
    def __init__(self, relative_width, event_type, **kwargs):
        super().__init__(relative_size=(relative_width, relative_width),
    scale_mode='width', **kwargs)

        self.image = pygame.Surface(self.size)
```

```
13          self.rect = self.image.get_rect()
14
15          self._square = _ColourSquare(
16              parent=self,
17              relative_position=(0.1, 0.1),
18              relative_width=0.5,
19              event_type=event_type
20          )
21          self._square.set_colour(kwargs.get('selected_colour'))
22
23          self._slider = _ColourSlider(
24              parent=self,
25              relative_position=(0.0, 0.7),
26              relative_width=1.0,
27              border_width=self.border_width,
28              border_colour=self._border_colour
29          )
30          self._slider.set_colour(kwargs.get('selected_colour'))
31
32          self._display = _ColourDisplay(
33              parent=self,
34              relative_position=(0.7, 0.1),
35              relative_size=(0.2, 0.5)
36          )
37          self._display.set_colour(kwargs.get('selected_colour'))
38
39          self._event_type = event_type
40          self._hover_event_type = event_type
41
42          self.set_image()
43          self.set_geometry()
44
45      def global_to_relative_pos(self, global_pos):
46          return (global_pos[0] - self.position[0], global_pos[1] - self.position
    [1])
47
48      def set_image(self):
49          self.image = pygame.Surface(self.size)
50          self.image.fill(self._fill_colour)
51
52          self._square.set_image()
53          self._square.set_geometry()
54          self.image.blit(self._square.image, self.global_to_relative_pos(self.
    _square.position))
55
56          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.MOUSEBUTTONUP, 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),
       scale_mode='width', **kwargs)
8
9          self._colour = None
10
11     def set_colour(self, new_colour):
12         self._colour = pygame.Color(new_colour)
13
14     def get_colour(self):
15         return self._colour
16
17     def set_image(self):
18         self.image = create_square_gradient(side_length=self.size[0], colour=self.
       _colour)
19
20     def process_event(self, event):
21         if event.type == pygame.MOUSEBUTTONDOWN:
22             relative_mouse_pos = (event.pos[0] - self.position[0], event.pos[1] -
       self.position[1])
23
24             if (
25                 0 > relative_mouse_pos[0] or
26                 self.size[0] < relative_mouse_pos[0] or
27                 0 > relative_mouse_pos[1] or
28                 self.size[1] < relative_mouse_pos[1]
29             ): return None
30
```

```
            self.set_colour(self.image.get_at(relative_mouse_pos))

            return self._colour

        return None
```

1.25.12 dropdown.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.bases.pressable import _Pressable
from data.utils.constants import WidgetState
from data.helpers.data_helpers import get_user_settings
from data.helpers.font_helpers import text_width_to_font_size, text_height_to_font_size
from data.utils.assets import GRAPHICS

user_settings = get_user_settings()

class Dropdown(_Pressable, _Widget):
    def __init__(self, word_list, event=None, **kwargs):
        _Pressable.__init__(
            self,
            event=event,
            hover_func=self.hover_func,
            down_func=lambda: self.set_state_colour(WidgetState.PRESS),
            up_func=self.up_func,
            sfx=None
        )
        _Widget.__init__(self, relative_size=None, **kwargs)

        if kwargs.get('relative_width'):
            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]
        elif kwargs.get('relative_height'):
            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]

        self._word_list = [word_list[0].capitalize()]
        self._word_list_copy = [word.capitalize() for word in word_list]

        self._expanded = False
        self._hovered_index = None

        self._empty_surface = pygame.Surface((0, 0))
        self._background_colour = self._fill_colour

        self.initialise_new_colours(self._fill_colour)
        self.set_state_colour(WidgetState.BASE)

        self.set_image()
        self.set_geometry()

    @property
    def size(self):
        max_word = sorted(self._word_list_copy, key=len)[-1]
        max_word_rect = self._font.get_rect(max_word, size=self.font_size)
        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)))
        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.FRect(0, 0, self.size[0], self.size[1])
101         pygame.draw.rect(self.image, self._background_colour, fill_rect)
102         pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int(
    self.border_width))
103
104         word_box_height = (self.size[1] - (2 * self.margin) - ((len(self.
    _word_list) - 1) * self.margin)) / len(self._word_list)
105
106         arrow_size = (GRAPHICS['dropdown_arrow_open'].width / GRAPHICS['
```

```
    dropdown_arrow_open'].height * word_box_height, word_box_height)
107         open_arrow_surface = pygame.transform.scale(GRAPHICS['dropdown_arrow_open'
    ], arrow_size)
108         closed_arrow_surface = pygame.transform.scale(GRAPHICS['
    dropdown_arrow_close'], arrow_size)
109         arrow_position = (self.size[0] - arrow_size[0] - self.margin, (
    word_box_height) / 3)
110 
111         if self._expanded:
112             self.image.blit(closed_arrow_surface, arrow_position)
113         else:
114             self.image.blit(open_arrow_surface, arrow_position)
115 
116         for index, word in enumerate(self._word_list):
117             word_position = (self.margin, self.margin + (word_box_height + self.
    margin) * index)
118             self._font.render_to(self.image, word_position, word, fgcolor=self.
    _text_colour, size=self.font_size)
119 
120         if self._hovered_index is not None:
121             overlay_surface = pygame.Surface((self.size[0], word_box_height + 2 *
    self.margin), pygame.SRCALPHA)
122             overlay_surface.fill((*self._fill_colour.rgb, 128))
123             overlay_position = (0, (word_box_height + self.margin) * self.
    _hovered_index)
124             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=
       False, box_colours=None, **kwargs):
7          super().__init__(**kwargs)
8  
9          if fit_icon:
10             aspect_ratio = icon.width / icon.height
11             self._relative_size = (self._relative_size[1] * aspect_ratio, self.
       _relative_size[1])
12 
13         self._icon = icon
14         self._is_mask = is_mask
15         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.
       _box_colours)
```

```
        else:
            self.image = pygame.transform.scale(self._empty_surface, self.size)

            if self._fill_colour:
                pygame.draw.rect(self.image, self._fill_colour, self.image.get_rect(), border_radius=int(self.border_radius))

        if self._stretch:
            if self._smooth:
                scaled_icon = pygame.transform.smoothscale(self._icon, (self.size[0] - (2 * self.margin), self.size[1] - (2 * self.margin)))
            else:
                scaled_icon = pygame.transform.scale(self._icon, (self.size[0] - (2 * self.margin), self.size[1] - (2 * self.margin)))

            icon_position = (self.margin, self.margin)
        else:
            max_height = self.size[1] - (2 * self.margin)
            max_width = self.size[0] - (2 * self.margin)
            scale_factor = min(max_width / self._icon.width, max_height / self._icon.height)

            if self._smooth:
                scaled_icon = pygame.transform.smoothscale_by(self._icon, (scale_factor, scale_factor))
            else:
                scaled_icon = pygame.transform.scale_by(self._icon, (scale_factor, scale_factor))
            icon_position = ((self.size[0] - scaled_icon.width) / 2, (self.size[1] - scaled_icon.height) / 2)

        if self._is_mask:
            self.image.blit(scaled_icon, icon_position, None, pygame.BLEND_RGBA_MULT)
        else:
            self.image.blit(scaled_icon, icon_position)

        if self._box_colours is None and self.border_width:
            pygame.draw.rect(self.image, self._border_colour, self.image.get_rect(), width=int(self.border_width), border_radius=int(self.border_radius))

    def process_event(self, event):
        pass
```

1.25.14 icon_button.py

```
from data.widgets.bases.pressable import _Pressable
from data.widgets.bases.box import _Box
from data.widgets.icon import Icon
from data.utils.constants import WidgetState, RED_BUTTON_COLOURS

class IconButton(_Box, _Pressable, Icon):
    def __init__(self, event, box_colours=RED_BUTTON_COLOURS, **kwargs):
        _Box.__init__(self, box_colours=box_colours)
        _Pressable.__init__(
            self,
            event=event,
            hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
            down_func=lambda: self.set_state_colour(WidgetState.PRESS),
            up_func=lambda: self.set_state_colour(WidgetState.BASE),
        )
        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
```

```
        for index, move in enumerate(self._move_list):
            if index % 2 == 0:
                text_position = (self.size[0] / 7, row_gap * (1 + 2 * (index // 2)
    ))
            else:
                text_position = (self.size[0] * 4 / 7, row_gap * (1 + 2 * (index
    // 2)))

            self._font.render_to(self.image, text_position, text=move, size=self.
    font_size, fgcolor=self._text_colour)

            move_number = (index // 2) + 1
            move_number_position = (self.size[0] / 14, row_gap * (1 + 2 * (index
    // 2)))
            self._font.render_to(self.image, move_number_position, text=str(
    move_number), size=self.font_size, fgcolor=self._text_colour)

    def process_event(self, event, scrolled_pos=None):
        pass
```

1.25.16 multiple_icon_button.py

```
import pygame
from data.utils.constants import WidgetState, LOCKED_BLUE_BUTTON_COLOURS,
    LOCKED_RED_BUTTON_COLOURS, RED_BUTTON_COLOURS, BLUE_BUTTON_COLOURS
from data.components.custom_event import CustomEvent
from data.widgets.bases.circular import _Circular
from data.widgets.icon_button import IconButton
from data.widgets.bases.box import _Box

class MultipleIconButton(_Circular, IconButton):
  def __init__(self, icons_dict, **kwargs):
    _Circular.__init__(self, items_dict=icons_dict)
    IconButton.__init__(self, icon=self.current_item, **kwargs)

    self._fill_colour_copy = self._fill_colour

    self._locked = None

  def set_locked(self, is_locked):
    self._locked = is_locked
    if self._locked:
      r, g, b, a  = pygame.Color(self._fill_colour_copy).rgba
      if self._box_colours_dict == BLUE_BUTTON_COLOURS:
        _Box.__init__(self, box_colours=LOCKED_BLUE_BUTTON_COLOURS)
      elif self._box_colours_dict == RED_BUTTON_COLOURS:
        _Box.__init__(self, box_colours=LOCKED_RED_BUTTON_COLOURS)
      else:
        self.initialise_new_colours((max(r + 50, 0), max(g + 50, 0), max(b + 50,
    0), a))
    else:
      if self._box_colours_dict == LOCKED_BLUE_BUTTON_COLOURS:
        _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
      elif self._box_colours_dict == LOCKED_RED_BUTTON_COLOURS:
        _Box.__init__(self, box_colours=RED_BUTTON_COLOURS)
      else:
        self.initialise_new_colours(self._fill_colour_copy)

    if self.rect.collidepoint(pygame.mouse.get_pos()):
      self.set_state_colour(WidgetState.HOVER)
    else:
      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
       .size[1] - 2 * self.margin), pygame.SRCALPHA)
38
39         if (len(self._piece_list) == 0):
40             self.set_image()
41             return
42
43         piece_width = min(self.size[1] - 2 * self.margin, (self.size[0] - 2 * self
       .margin) / len(self._piece_list))
44         piece_list = []
```

```
45
46          for index, piece in enumerate(self._piece_list):
47              piece_instance = PieceSprite(piece, self._active_colour.
    get_flipped_colour(), Rotation.UP)
48              piece_instance.set_geometry((0, 0), piece_width)
49              piece_instance.set_image()
50              piece_list.append((piece_instance.image, (piece_width * index, (self.
    _piece_surface.height - piece_width) / 2)))
51
52          self._piece_surface.fblits(piece_list)
53
54          self.set_image()
55
56      def set_image(self):
57          self.image = create_text_box(self.size, self.border_width, self.
    _box_colours)
58
59          resized_piece_surface = scale_and_cache(self._piece_surface, (self.size[0]
     - 2 * self.margin, self.size[1] - 2 * self.margin))
60          self.image.blit(resized_piece_surface, (self.margin, self.margin))
61
62      def process_event(self, event):
63          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(),
     border_radius=int(self.border_radius))
18
19             if self.border_width:
20                 pygame.draw.rect(self.image, self._border_colour, self.image.
    get_rect(), width=int(self.border_width), border_radius=int(self.border_radius
    ))
21
22     def process_event(self, event):
23         pass
```

1.25.21 scrollbar.py

```
import pygame
from data.widgets.bases.pressable import _Pressable
from data.widgets.bases.widget import _Widget
from data.utils.constants import WidgetState
from data.utils.enums import Miscellaneous

class _Scrollbar(_Pressable, _Widget):
    def __init__(self, vertical, **kwargs):
        _Pressable.__init__(
            self,
            event=Miscellaneous.PLACEHOLDER,
            hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
            down_func=self.down_func,
            up_func=self.up_func,
            prolonged=True,
            sfx=None
        )
        _Widget.__init__(self, **kwargs)

        self._vertical = vertical
        self._last_mouse_px = None

        self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)

        self.initialise_new_colours(self._fill_colour)
        self.set_state_colour(WidgetState.BASE)

        self.set_image()
        self.set_geometry()

    def down_func(self):
        if self._vertical:
            self._last_mouse_px = pygame.mouse.get_pos()[1]
        else:
            self._last_mouse_px = pygame.mouse.get_pos()[0]

        self.set_state_colour(WidgetState.PRESS)

    def up_func(self):
        self._last_mouse_px = None
        self.set_state_colour(WidgetState.BASE)

    def set_relative_position(self, relative_position):
        self._relative_position = relative_position
        self.set_geometry()

    def set_relative_size(self, new_relative_size):
        self._relative_size = new_relative_size

    def set_image(self):
        self.image = pygame.transform.scale(self._empty_surface, self.size)

        if self._vertical:
            rounded_radius = self.size[0] / 2
        else:
            rounded_radius = self.size[1] / 2

        pygame.draw.rect(self.image, self._fill_colour, (0, 0, self.size[0], self.
    size[1]), border_radius=int(rounded_radius))

```

```
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] /
    self._widget.rect.height) * self.size[1])
55          else:
56              return (min(1, self.size[0] / (self._widget.rect.width + 0.001)) *
    self.size[0], self.size[1] * SCROLLBAR_WIDTH_FACTOR)
57
58      @property
59      def size(self):
60          if self._vertical is False:
61              return (self._relative_size[0] * self.surface_size[0], self.
    _relative_size[1] * self.surface_size[1]) # scale with horizontal width to
    always fill entire length of screen
62          else:
63              return super().size
64
65      def calculate_scroll_percentage(self, offset, scrollbar=False):
66          if self._vertical:
67              widget_height = self._widget.rect.height
68
69              if widget_height < self.size[1]:
70                  return 0
71
72              if scrollbar:
73                  self._scroll_percentage += offset / (self.size[1] - self.
    scrollbar_size[1] + 0.001)
74              else:
75                  max_scroll_height = widget_height - self.size[1]
76                  current_scroll_height = self._scroll_percentage *
    max_scroll_height
77                  self._scroll_percentage = (current_scroll_height + offset) / (
    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_scoll_width = widget_width - self.size[0]
88                  current_scroll_width = self._scroll_percentage * max_scoll_width
89                  self._scroll_percentage = (current_scroll_width + offset) /
    max_scoll_width
90
91          return min(1, max(0, self._scroll_percentage))
92
```

```
93      def calculate_widget_rect(self):
94          widget_position = self.calculate_widget_position()
95          return pygame.FRect(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,
       scrollbar=True)
153 
154         if self.rect.collidepoint(pygame.mouse.get_pos()):
155             if event.type == pygame.MOUSEBUTTONDOWN:
156                 if event.button == 4:
157                     self._scroll_percentage = self.calculate_scroll_percentage(-
      self.scroll_factor)
158                     self.set_image()
159                     return
160                 elif event.button == 5:
161                     if self._scroll_percentage == 100:
162                         return
163 
164                     self._scroll_percentage = self.calculate_scroll_percentage(
      self.scroll_factor)
165                     self.set_image()
166                     return
167 
168         widget_event = self._widget.process_event(event, scrolled_pos=self.
      calculate_widget_position())
169         if widget_event is not None:
170             self.set_image()
171         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=
      theme['fillPrimary']):
8         super().__init__(
9             event=None,
10             down_func=self.down_func,
11             up_func=self.up_func,
12             hover_func=self.hover_func,
13             prolonged=True,
14             sfx=None
15         )
16         self._border_colour = border_colour
17         self._radius = radius
18         self._percent = None
19 
20         self.state = WidgetState.BASE
21         self.initialise_new_colours(fill_colour)
22 
23     def get_position(self):
24         return (self.rect.x, self.rect.y)
25 
26     def set_position(self, position):
27         self.rect = self._thumb_surface.get_rect()
28         self.rect.topleft = position
```

```
29
30      def get_surface(self):
31          return self._thumb_surface
32
33      def set_surface(self, radius, border_width):
34          self._thumb_surface = create_slider_thumb(radius, self._colours[self.state
        ], self._border_colour, border_width)
35
36      def get_pressed(self):
37          return self._pressed
38
39      def down_func(self):
40          self.state = WidgetState.PRESS
41
42      def up_func(self):
43          self.state = WidgetState.BASE
44
45      def hover_func(self):
46          self.state = WidgetState.HOVER
```

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'],
       on_colour=theme['fillSecondary'], off_colour=theme['fillPrimary'], **kwargs):
11         _Pressable.__init__(
12             self,
13             event=event,
14             hover_func=self.hover_func,
15             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
16             up_func=self.up_func,
17         )
18         _Widget.__init__(self, relative_size=(relative_height * 2, relative_height
       ), scale_mode='height',fill_colour=fill_colour, **kwargs)
19
20         self._on_colour = on_colour
21         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:
```

```
            self._background_colour = self._on_colour
        else:
            self._background_colour = self._off_colour

        self.set_image()

    def up_func(self):
        if self.get_widget_state() == WidgetState.PRESS:
            toggle_state = not(self._is_toggled)
            self.set_toggle_state(toggle_state)

        self.set_state_colour(WidgetState.BASE)

    def draw_thumb(self):
        margin = self.size[1] * 0.1
        thumb_radius = (self.size[1] / 2) - margin

        if self._is_toggled:
            thumb_center = (self.size[0] - margin - thumb_radius, self.size[1] / 2)
        else:
            thumb_center = (margin + thumb_radius, self.size[1] / 2)

        pygame.draw.circle(self.image, self._fill_colour, thumb_center, thumb_radius)

    def set_image(self):
        self.image = create_switch(self.size, self._background_colour)
        self.draw_thumb()

    def process_event(self, event):
        data = super().process_event(event)

        if data:
            return CustomEvent(**vars(data), toggled=self._is_toggled)
```

1.25.25 text.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.helpers.font_helpers import text_width_to_font_size, text_height_to_font_size, height_to_font_size
from data.helpers.widget_helpers import create_text_box

class Text(_Widget): # Pure text
    def __init__(self, text, center=True, fit_vertical=True, box_colours=None, strength=0.05, font_size=None, **kwargs):
        super().__init__(**kwargs)
        self._text = text
        self._fit_vertical = fit_vertical
        self._strength = strength
        self._box_colours = box_colours

        if fit_vertical:
            self._relative_font_size = text_height_to_font_size(self._text, self._font, (self.size[1] - 2 * (self.margin + self.border_width))) / self.surface_size[1]
        else:
            self._relative_font_size = text_width_to_font_size(self._text, self._font, (self.size[0] - 2 * (self.margin + self.border_width))) / self.surface_size[1]

```

```
19          if font_size:
20              self._relative_font_size = font_size / self.surface_size[1]
21
22          self._center = center
23          self.rect = self._font.get_rect(self._text, size=self.font_size)
24          self.rect.topleft = self.position
25
26          self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
27
28          self.set_image()
29          self.set_geometry()
30
31      def resize_text(self):
32          if self._fit_vertical:
33              self._relative_font_size = text_height_to_font_size(self._text, self.
    _font, (self.size[1] - 2 * (self.margin + self.border_width))) / self.
    surface_size[1]
34          else:
35              ideal_font_size = height_to_font_size(self._font, target_height=(self.
    size[1] - (self.margin + self.border_width))) / self.surface_size[1]
36              new_font_size = text_width_to_font_size(self._text, self._font, (self.
    size[0] - (self.margin + self.border_width))) / self.surface_size[1]
37
38              if new_font_size < ideal_font_size:
39                  self._relative_font_size = new_font_size
40              else:
41                  self._relative_font_size = ideal_font_size
42
43      def set_text(self, new_text):
44          self._text = new_text
45
46          self.resize_text()
47          self.set_image()
48
49      def set_image(self):
50          if self._box_colours:
51              self.image = create_text_box(self.size, self.border_width, self.
    _box_colours)
52          else:
53              text_surface = pygame.transform.scale(self._empty_surface, self.size)
54              self.image = text_surface
55
56              if self._fill_colour:
57                  fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
58                  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:
```

```
            self._timer_running = True
            animation.set_timer(1000, self.decrement_second)

        self._active = is_active

    def set_time(self, milliseconds):
        self._current_ms = milliseconds
        self._text = self.format_to_text()
        self.set_image()
        self.set_geometry()

    def get_time(self):
        return self._current_ms / (1000 * 60)

    def decrement_second(self):
        if self._active:
            self.set_time(self._current_ms - 1000)

            if self._current_ms <= 0:
                self._active = False
                self._timer_running = False
                self.set_time(0)
                pygame.event.post(pygame.event.Event(pygame.MOUSEMOTION, pos=
    pygame.mouse.get_pos())) # RANDOM EVENT TO TRIGGER process_event
            else:
                animation.set_timer(1000, self.decrement_second)
        else:
            self._timer_running = False

    def format_to_text(self):
        raw_seconds = self._current_ms / 1000
        minutes, seconds = divmod(raw_seconds, 60)
        return f'{str(int(minutes)).zfill(2)}:{str(int(seconds)).zfill(2)}'

    def process_event(self, event):
        if self._current_ms <= 0:
            return CustomEvent(**vars(self._event), active_colour=self.
    _active_colour)
```

1.25.29 volume_slider.py

```
import pygame
from data.helpers.asset_helpers import scale_and_cache
from data.helpers.widget_helpers import create_slider
from data.utils.event_types import SettingsEventType
from data.components.custom_event import CustomEvent
from data.widgets.slider_thumb import _SliderThumb
from data.widgets.bases.widget import _Widget
from data.utils.constants import WidgetState
from data.managers.theme import theme

class VolumeSlider(_Widget):
    def __init__(self, relative_length, default_volume, volume_type, thumb_colour=
    theme['fillSecondary'], **kwargs):
        super().__init__(relative_size=(relative_length, relative_length * 0.2),
    **kwargs)

        self._volume_type = volume_type
        self._selected_percent = default_volume
        self._last_mouse_x = None

```

```
19          self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
    _border_colour, fill_colour=thumb_colour)
20          self._gradient_surface = create_slider(self.calculate_slider_size(), self.
    _fill_colour, self.border_width, self._border_colour)
21
22          self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
23
24      @property
25      def position(self):
26          '''Minus so easier to position slider by starting from the left edge of
    the slider instead of the thumb'''
27          return (self._relative_position[0] * self.surface_size[0] - (self.size[1]
    / 2), self._relative_position[1] * self.surface_size[1])
28
29      def calculate_slider_position(self):
30          return (self.size[1] / 2, self.size[1] / 4)
31
32      def calculate_slider_size(self):
33          return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
34
35      def calculate_selected_percent(self, mouse_pos):
36          if self._last_mouse_x is None:
37              return
38
39          x_change = (mouse_pos[0] - self._last_mouse_x) / (self.
    calculate_slider_size()[0] - 2 * self.border_width)
40          return max(0, min(self._selected_percent + x_change, 1))
41
42      def calculate_thumb_position(self):
43          gradient_size = self.calculate_slider_size()
44          x = gradient_size[0] * self._selected_percent
45          y = 0
46
47          return (x, y)
48
49      def relative_to_global_position(self, position):
50          relative_x, relative_y = position
51          return (relative_x + self.position[0], relative_y + self.position[1])
52
53      def set_image(self):
54          gradient_scaled = scale_and_cache(self._gradient_surface, self.
    calculate_slider_size())
55          gradient_position = self.calculate_slider_position()
56
57          self.image = pygame.transform.scale(self._empty_surface, (self.size))
58          self.image.blit(gradient_scaled, gradient_position)
59
60          thumb_position = self.calculate_thumb_position()
61          self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
    border_width)
62          self._thumb.set_position(self.relative_to_global_position((thumb_position
    [0], thumb_position[1])))
63
64          thumb_surface = self._thumb.get_surface()
65          self.image.blit(thumb_surface, thumb_position)
66
67      def set_volume(self, volume):
68          self._selected_percent = volume
69          self.set_image()
70
71      def process_event(self, event):
72          if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
```

```
    MOUSEBUTTONUP]:
73              return
74
75          before_state = self._thumb.state
76          self._thumb.process_event(event)
77          after_state = self._thumb.state
78
79          if before_state != after_state:
80              self.set_image()
81
82              if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:
83                  self._last_mouse_x = None
84                  return CustomEvent(SettingsEventType.VOLUME_SLIDER_CLICK, volume=
    round(self._selected_percent, 3), volume_type=self._volume_type)
85
86          if self._thumb.state == WidgetState.PRESS:
87              selected_percent = self.calculate_selected_percent(event.pos)
88              self._last_mouse_x = event.pos[0]
89
90              if selected_percent:
91                  self._selected_percent = selected_percent
92                  self.set_image()
93                  return CustomEvent(SettingsEventType.VOLUME_SLIDER_SLIDE)
```

1.25.30 __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
```

```
__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

```
from data.utils.constants import WidgetState

class _Box:
    def __init__(self, box_colours):
        self._box_colours_dict = box_colours
        self._box_colours = self._box_colours_dict[WidgetState.BASE]

    def set_state_colour(self, state):
        self._box_colours = self._box_colours_dict[state]
        super().set_state_colour(state)
```

1.26.2 circular.py

See Section ??.

1.26.3 pressable.py

```
import pygame
from data.utils.constants import WidgetState
from data.managers.audio import audio
from data.utils.assets import SFX

class _Pressable:
    def __init__(self, event, down_func=None, up_func=None, hover_func=None,
    prolonged=False, sfx=SFX['button_click'], **kwargs):
        self._down_func = down_func
        self._up_func = up_func
        self._hover_func = hover_func
        self._pressed = False
        self._prolonged = prolonged
        self._sfx = sfx

        self._event = event

        self._widget_state = WidgetState.BASE

        self._colours = {}

    def set_state_colour(self, state):
        self._fill_colour = self._colours[state]

        self.set_image()

    def initialise_new_colours(self, colour):
        r, g, b, a = pygame.Color(colour).rgba

        self._colours = {
            WidgetState.BASE: pygame.Color(r, g, b, a),
            WidgetState.HOVER: pygame.Color(min(r + 25, 255), min(g + 25, 255),
    min(b + 25, 255), a),
```

```
            WidgetState.PRESS: pygame.Color(min(r + 50, 255), min(g + 50, 255),
    min(b + 50, 255), a)
        }

    def get_widget_state(self):
        return self._widget_state

    def process_event(self, event):
        match event.type:
            case pygame.MOUSEBUTTONDOWN:
                if self.rect.collidepoint(event.pos):
                    self._down_func()
                    self._widget_state = WidgetState.PRESS

            case pygame.MOUSEBUTTONUP:
                if self.rect.collidepoint(event.pos):
                    if self._widget_state == WidgetState.PRESS:
                        if self._sfx:
                            audio.play_sfx(self._sfx)

                        self._up_func()
                        self._widget_state = WidgetState.HOVER
                        return self._event

                    elif self._widget_state == WidgetState.BASE:
                        self._hover_func()

                elif self._prolonged and self._widget_state == WidgetState.PRESS:
                    if self._sfx:
                        audio.play_sfx(self._sfx)
                    self._up_func()
                    self._widget_state = WidgetState.BASE
                    return self._event

            case pygame.MOUSEMOTION:
                if self.rect.collidepoint(event.pos):
                    if self._widget_state == WidgetState.PRESS:
                        return
                    elif self._widget_state == WidgetState.BASE:
                        self._hover_func()
                        self._widget_state = WidgetState.HOVER
                    elif self._widget_state == WidgetState.HOVER:
                        self._hover_func()
                else:
                    if self._prolonged is False:
                        if self._widget_state in [WidgetState.PRESS, WidgetState.
    HOVER]:
                            self._widget_state = WidgetState.BASE
                            self._up_func()
                        elif self._widget_state == WidgetState.BASE:
                            return
                    elif self._prolonged is True:
                        if self._widget_state in [WidgetState.PRESS, WidgetState.
    BASE]:
                            return
                        else:
                            self._widget_state = WidgetState.BASE
                            self._up_func()
```

1.26.4 widget.py

See Section ??.