Chapter 1

Source Code

1.1 data

1.1.1 assets.py

```python
from pathlib import Path
from data.utils.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
```

1.1.2 constants.py

```python
import pygame
from enum import IntEnum, StrEnum, auto

BG_COLOUR = (0, 0, 0)
PAUSE_COLOUR = (50, 50, 50, 128)
OVERLAY_COLOUR_LIGHT = (*pygame.Color('0xf14e52').rgb, 128)
OVERLAY_COLOUR_DARK = (*pygame.Color('0x9b222b').rgb, 192)
SCREEN_SIZE = (1200, 600)
# SCREEN_SIZE = (600, 600)
SCREEN_FLAGS = pygame.HWSURFACE | pygame.DOUBLEBUF | pygame.RESIZABLE | pygame.OPENGL
STARTING_SQUARE_SIZE = (SCREEN_SIZE[1] * 0.64) / 8 #Board height divded by 8
EMPTY_BB = 0
A_FILE_MASK = 0b11111111101111111110111111111011111111101111111110111111111011111111101111111110
J_FILE_MASK = 0b01111111110111111111011111111101111111110111111111011111111101111111110111111111
ONE_RANK_MASK = 0b11111111111111111111111111111111111111111111111111111111111111111111110000000000
```

```python
EIGHT_RANK_MASK = 0
    b00000000001111111111111111111111111111111111111111111111111111111111111111111111

TEST_MASK = 0
    b00000000100000000010000000000000000000000000000000000000000001000000000100000000

GAMES_PER_PAGE = 10

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()

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

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

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

class WidgetState(StrEnum):
    BASE = auto()
    HOVER = auto()
    PRESS = auto()

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']
}

class StatusText(StrEnum):
    PLAYER_MOVE = auto()
    CPU_MOVE = auto()
    WIN = auto()
    DRAW = 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):
```

```python
    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()

class Colour(IntEnum):
    BLUE = 0
    RED = 1

    def get_flipped_colour(self):
        if self == Colour.BLUE:
            return Colour.RED
        elif self == Colour.RED:
            return Colour.BLUE

class Piece(StrEnum):
    SPHINX = 's'
    PYRAMID = 'p'
    ANUBIS = 'n'
    SCARAB = 'r'
    PHAROAH = 'f'

class Score(IntEnum):
    PHAROAH = 0
    SPHINX = 0
    PYRAMID = 100
    ANUBIS = 110
    SCARAB = 200

    MOVE = 4
    POSITION = 11
    PHAROAH_SAFETY = 31
    CHECKMATE = 100000
    INFINITE = 6969696969

class Rank(IntEnum):
    ONE = 0
    TWO = 1
    THREE = 2
    FOUR = 3
    FIVE = 4
    SIX = 5
    SEVEN = 6
    EIGHT = 7

class File(IntEnum):
    A = 0
    B = 1
    C = 2
    D = 3
    E = 4
    F = 5
    G = 6
    H = 7
    I = 8
    J = 9

class Rotation(StrEnum):
    UP = 'a'
    RIGHT = 'b'
    DOWN = 'c'
    LEFT = 'd'

    def to_angle(self):
        if self == Rotation.UP:
            return 0
        elif self == Rotation.RIGHT:
            return 270
        elif self == Rotation.DOWN:
            return 180
        elif self == Rotation.LEFT:
            return 90

    def get_clockwise(self):
        if self == Rotation.UP:
            return Rotation.RIGHT
        elif self == Rotation.RIGHT:
            return Rotation.DOWN
        elif self == Rotation.DOWN:
            return Rotation.LEFT
        elif self == Rotation.LEFT:
            return Rotation.UP

    def get_anticlockwise(self):
        if self == Rotation.UP:
            return Rotation.LEFT
        elif self == Rotation.RIGHT:
            return Rotation.UP
        elif self == Rotation.DOWN:
            return Rotation.RIGHT
        elif self == Rotation.LEFT:
            return Rotation.DOWN

    def get_opposite(self):
        return self.get_clockwise().get_clockwise()

class RotationIndex(IntEnum):
    FIRSTBIT = 0
    SECONDBIT = 1

class RotationDirection(StrEnum):
    CLOCKWISE = 'cw'
    ANTICLOCKWISE = 'acw'

    def get_opposite(self):
        if self == RotationDirection.CLOCKWISE:
            return RotationDirection.ANTICLOCKWISE
        elif self == RotationDirection.ANTICLOCKWISE:
            return RotationDirection.CLOCKWISE

class MoveType(StrEnum):
    MOVE = 'm'
    ROTATE = 'r'

class LaserType(IntEnum):
    END = 0
    STRAIGHT = 1
    CORNER = 2

class LaserDirection(IntEnum):
    FROM_TOP = 1
    FROM_RIGHT = 2
    FROM_BOTTOM = 3
    FROM_LEFT = 4
```

1.1.3 control.py

```python
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.managers.window import window
from data.managers.audio import audio
from data.managers.theme import theme
from data.assets import DEFAULT_FONT

logger = initialise_logger(__file__)

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

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()

    def flip_state(self):
        self.state.done = False
        persist = self.state.cleanup()

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

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

    def update(self):
        if self.state.quit:
            self.done = True
        elif self.state.done:
            self.flip_state()

        self._clock.tick(FPS)
        animation.set_delta_time()

        self.state.update()

        if SHOW_FPS:
            self.draw_fps()

        window.update()

    def main_game_loop(self):
        while not self.done:
            self.event_loop()
            self.update()

    def update_window(self, resize=False):
        if resize:
            self.update_native_window_size()
            window.handle_resize()
            self.state.handle_resize()

        self.update()

    def draw_fps(self):
        fps = str(int(self._clock.get_fps()))
        DEFAULT_FONT.strength = 0.1
        DEFAULT_FONT.render_to(window.screen, (0, 0), fps, fgcolor=theme['textError'], size=15)

    def update_native_window_size(self):
        x, y = window.size

        max_window_x = 100000
        max_window_y = x / 1.4
        min_window_x = 400
        min_window_y = min_window_x/1.4

        if x / y < 1.4:
            min_window_x = x

        min_window_size = (min_window_x, min_window_y)
        max_window_size = (max_window_x, max_window_y)
        window.minimum_size = min_window_size
        window.maximum_size = max_window_size

    def event_loop(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.done = True

            if event.type == pygame.MOUSEBUTTONDOWN and event.button != 1: # ONLY PROCESS LEFT CLICKS
                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()
```

1.1.4 loading_screen.py

See Section ??.

1.1.5 main.py

See Section ??.

1.1.6 setup.py

```
import pygame

pygame.mixer.init()
pygame.init()

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)
```

1.1.7 windows_setup.py

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

def wndProc(oldWndProc, draw_callback, hWnd, message, wParam, lParam):
    if message == win32con.WM_SIZING or message == win32con.WM_TIMER: # Don't know
     what WM_TIMER does
        draw_callback(resize=True)
        win32gui.RedrawWindow(hWnd, None, None, win32con.RDW_INVALIDATE | win32con
    .RDW_ERASE)
    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

if os.name != 'nt' or sys.getwindowsversion()[0] < 6:
    raise NotImplementedError("Incompatible OS!")
```

1.1.8 __init__.py

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": ""
        }
    },
    "dimensions": {
        "borderRadius": 3,
        "borderWidth": 5,
        "margin": 10
    }
}
```

1.2.5 user_settings.json

```
{
    "primaryBoardColour": "0xB98766",
    "secondaryBoardColour": "0xF3D8B8",
    "laserColourBlue": "0x0000ff",
    "laserColourRed": "0xff0000",
    "displayMode": "windowed",
    "musicVolume": 0.085,
    "sfxVolume": 0.336,
    "particles": true,
    "opengl": true,
    "shader": "default"
}
```

1.3 data\components

1.3.1 circular_linked_list.py

See Section ??.

1.3.2 cursor.py

```python
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

```python
from data.constants import Colour
from data.states.game.components.move import Move

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

    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)

    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
        ])

    @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 != ''
        ]

# self.states = {
#     'CPU_ENABLED': game_config['CPU_ENABLED'],
#     'CPU_DEPTH': game_config['CPU_DEPTH'],
#     'AWAITING_CPU': False,
#     'WINNER': None,
#     'PAUSED': False,
#     'ACTIVE_COLOUR': Colour.BLUE,
#     'TIME_ENABLED': game_config['TIME_ENABLED'],
#     'TIME': game_config['TIME'],
#     'MOVES': []
# }


#     move_item = {
#     'time': {
#         Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
#         Colour.RED: GAME_WIDGETS['red_timer'].get_time()
#     },
#     'move': move_notation,
#     'laserResult': laser_result
# }
```

1.3.5 widget_group.py

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

class WidgetGroup(pygame.sprite.Group):
    def __init__(self, widget_dict):
        super().__init__()

        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

    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

```python
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 created_dt TIMESTAMP NOT NULL
    ''')

    connection.commit()
    connection.close()

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

```python
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\managers

1.6.1 animation.py

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

FPS = 60

class AnimationManager:
```

```python
    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()
```

1.6.2 audio.py

```python
import pygame
from data.utils.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 = []

    def set_sfx_volume(self, volume):
        self._sfx_volume = volume

        for channel in self._current_channels:
            channel.set_volume(self._sfx_volume)

    def set_music_volume(self, volume):
        self._music_volume = volume

        pygame.mixer.music.set_volume(self._music_volume)

    def pause_sfx(self):
        pygame.mixer.pause()

    def unpause_sfx(self):
        pygame.mixer.unpause()

    def stop_sfx(self, fadeout=0):
        pygame.mixer.fadeout(fadeout)

    def remove_unused_channels(self):
        unused_channels = []
        for channel in self._current_channels:
            if channel.get_busy() is False:
                unused_channels.append(channel)

        return unused_channels

    def play_sfx(self, sfx, loop=False):
        unused_channels = self.remove_unused_channels()

        if len(unused_channels) == 0:
            channel = pygame.mixer.find_channel()
        else:
            channel = unused_channels.pop(0)

        if channel is None:
            logger.warning('No available channel for SFX')
            return

        self._current_channels.append(channel)
        channel.set_volume(self._sfx_volume)

        if loop:
            channel.play(sfx, loops=-1)
        else:
            channel.play(sfx)

    def play_music(self, music_path):
        if 'menu' in str(music_path) and 'menu' in str(self._current_song):
            return

        if music_path == self._current_song:
            return

        pygame.mixer.music.stop()
        pygame.mixer.music.unload()
        pygame.mixer.music.load(music_path)
        pygame.mixer.music.set_volume(self._music_volume)
        pygame.mixer.music.play(loops=-1)

        self._current_song = music_path

audio = AudioManager()
```

1.6.3 cursor.py

```
import pygame
from data.assets import GRAPHICS
from data.constants import CursorMode

class CursorManager:
    def __init__(self):
```

```python
        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()
```

1.6.4 logs.py

```python
import logging.config
from data.utils.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)
```

1.6.5 shader.py

See Section ??.

1.6.6 theme.py

See Section ??.

1.6.7 window.py

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

```python

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)

            self.shader_arguments = {
                ShaderType.BASE: {},
                ShaderType.SHAKE: {},
                ShaderType.BLOOM: {},
                ShaderType.GRAYSCALE: {},
                ShaderType.CRT: {},
                ShaderType.RAYS: {}
            }

            if (selected_shader := get_user_settings()['shader']) is not None:
                for shader_type in SHADER_MAP[selected_shader]:
                    self.set_effect(shader_type)
        else:
            from data.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)
```

1.7 data\shaders

1.7.1 protocol.py

```
import pygame
import moderngl
from typing import Protocol, Optional
from data.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
```

1.8 data\shaders\classes

1.8.1 base.py

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

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)

        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
                )
```

```python
            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)

        self._shader_manager.get_fbo_texture(ShaderType.BASE).use(1)
        self._shader_manager.render_to_fbo(ShaderType.BASE, texture, background=1)
```

1.8.2 blend.py

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

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

        self._shader_manager.create_framebuffer(ShaderType._BLEND)

    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])
        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
```

1.8.3 bloom.py

See Section ??.

1.8.4 blur.py

See Section ??.

1.8.5 chromatic_abbreviation.py

```python
import pygame
from data.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)
```

1.8.6 crop.py

```python
from data.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)
```

1.8.7 crt.py

```python
from data.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)
```

1.8.8 grayscale.py

```python
from data.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)
```

1.8.9 highlight_brightness.py

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

```

```python
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)
```

1.8.10 highlight_colour.py

```python
from data.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)
```

1.8.11 lightmap.py

```python
from data.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):
        self._shader_manager = shader_manager

        shader_manager.load_shader(ShaderType._SHADOWMAP)

    def apply(self, texture, colour, softShadow, occlusion=None, falloff=0.0,
    clamp=(-180, 180)):
        self._shader_manager.create_framebuffer(ShaderType._LIGHTMAP, size=texture
    .size)
        self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)

        _Shadowmap(self._shader_manager).apply(texture, occlusion)
        shadow_map = self._shader_manager.get_fbo_texture(ShaderType._SHADOWMAP)

        self._shader_manager.render_to_fbo(ShaderType._LIGHTMAP, shadow_map,
    resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
    clamp, softShadow=softShadow)

        self._shader_manager._ctx.disable(self._shader_manager._ctx.BLEND)
```

1.8.12 occlusion.py

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

```python

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

    def apply(self, texture, occlusion_colour=(255, 0, 0)):
        self._shader_manager.create_framebuffer(ShaderType._OCCLUSION, size=texture.size)
        self._shader_manager.render_to_fbo(ShaderType._OCCLUSION, texture, checkColour=tuple(num / 255 for num in occlusion_colour))
```

1.8.13 rays.py

See Section ??.

1.8.14 shadowmap.py

```python
import moderngl
from data.constants import ShaderType
from data.shaders.protocol import SMProtocol
from data.shaders.classes.occlusion import _Occlusion

LIGHT_RESOLUTION = 256

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

        shader_manager.load_shader(ShaderType._OCCLUSION)

    def apply(self, texture, occlusion_texture=None):
        self._shader_manager.create_framebuffer(ShaderType._SHADOWMAP, size=(texture.size[0], 1), filter=moderngl.LINEAR)

        if occlusion_texture is None:
            _Occlusion(self._shader_manager).apply(texture)
            occlusion_texture = self._shader_manager.get_fbo_texture(ShaderType._OCCLUSION)

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

1.8.15 shake.py

```python
from data.constants import ShaderType
from data.shaders.protocol import SMProtocol
from random import randint

SHAKE_INTENSITY = 3

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

        self._shader_manager.create_framebuffer(ShaderType.SHAKE)

    def apply(self, texture, intensity=SHAKE_INTENSITY):
        displacement = (randint(-intensity, intensity) / 1000, randint(-intensity, intensity) / 1000)
        self._shader_manager.render_to_fbo(ShaderType.SHAKE, texture, displacement=displacement)
```

1.8.16 __init__.py

```python
from data.shaders.classes.chromatic_abbreviation import ChromaticAbbreviation
from data.shaders.classes.highlight_brightness import _HighlightBrightness
from data.shaders.classes.highlight_colour import _HighlightColour
from data.shaders.classes.shadowmap import _Shadowmap
from data.shaders.classes.occlusion import _Occlusion
from data.shaders.classes.grayscale import Grayscale
from data.shaders.classes.lightmap import _Lightmap
from data.shaders.classes.blend import _Blend
from data.shaders.classes.shake import Shake
from data.shaders.classes.bloom import Bloom
from data.shaders.classes.blur import _Blur
from data.shaders.classes.crop import _Crop
from data.shaders.classes.rays import Rays
from data.shaders.classes.base import Base
from data.shaders.classes.crt import CRT
from data.constants import ShaderType

shader_pass_lookup = {
    ShaderType.CHROMATIC_ABBREVIATION: ChromaticAbbreviation,
    ShaderType.GRAYSCALE: Grayscale,
    ShaderType.SHAKE: Shake,
    ShaderType.BLOOM: Bloom,
    ShaderType.BASE: Base,
    ShaderType.RAYS: Rays,
    ShaderType.CRT: CRT,

    ShaderType._HIGHLIGHT_BRIGHTNESS: _HighlightBrightness,
    ShaderType._HIGHLIGHT_COLOUR: _HighlightColour,
    ShaderType._CALIBRATE: lambda *args: None,
    ShaderType._OCCLUSION: _Occlusion,
    ShaderType._SHADOWMAP: _Shadowmap,
    ShaderType._LIGHTMAP: _Lightmap,
    ShaderType._BLEND: _Blend,
    ShaderType._BLUR: _Blur,
    ShaderType._CROP: _Crop,
}
```

1.9 data\shaders\fragments

1.9.1 background_balatro.frag

```glsl
# version 330 core

// Original by localthunk (https://www.playbalatro.com)

// 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);
    }

    float contrast_mod = (0.25*CONTRAST + 0.5*SPIN_AMOUNT + 1.2);
    float paint_res = min(2., max(0.,length(uv)*(0.035)*contrast_mod));
    float c1p = max(0.,1. - contrast_mod*abs(1.-paint_res));
    float c2p = max(0.,1. - contrast_mod*abs(paint_res));
    float c3p = 1. - min(1., c1p + c2p);
    float light = (LIGTHING - 0.2)*max(c1p*5. - 4., 0.) + LIGTHING*max(c2p*5. -
    4., 0.);
    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;
}

void main() {
    f_colour = effect(screenSize.xy, uvs* screenSize.xy);
}
```

1.9.2 background_gradient.frag

```
// Modified from https://www.shadertoy.com/view/wdyczG

#version 330 core

uniform float time;
uniform vec2 screenSize;

in vec2 uvs;
out vec4 f_colour;
```

```

#define S(a,b,t) smoothstep(a,b,t)

mat2 Rot(float a)
{
    float s = sin(a);
    float c = cos(a);
    return mat2(c, -s, s, c);
}

// Created by inigo quilez - iq/2014
// License Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
vec2 hash( vec2 p )
{
    p = vec2( dot(p,vec2(2127.1,81.17)), dot(p,vec2(1269.5,283.37)) );
  return fract(sin(p)*43758.5453);
}

float noise( in vec2 p )
{
    vec2 i = floor( p );
    vec2 f = fract( p );

  vec2 u = f*f*(3.0-2.0*f);

    float n = mix( mix( dot( -1.0+2.0*hash( i + vec2(0.0,0.0) ), f - vec2(0.0,0.0) ),
                        dot( -1.0+2.0*hash( i + vec2(1.0,0.0) ), f - vec2(1.0,0.0) ), u.x),
                   mix( dot( -1.0+2.0*hash( i + vec2(0.0,1.0) ), f - vec2(0.0,1.0) ),
                        dot( -1.0+2.0*hash( i + vec2(1.0,1.0) ), f - vec2(1.0,1.0) ), u.x), u.y);
  return 0.5 + 0.5*n;
}

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);
}
```

1.9.3 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));
}
```

1.9.4 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);
}
```

1.9.5 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));
}
```

1.9.6 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);
  }
}
```

1.9.7 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() {
//     f_colour = vec4(texture(image, uvs).rgba);
// }

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);

    // if (image2Pos.x < uvs.x &&
    //     uvs.x < (image2Pos.x + relativeSize.x) &&
    //     image2Pos.y < uvs.y &&
    //     uvs.y < (image2Pos.y + relativeSize.y)) {

    //     vec2 image2Coords = vec2((uvs.x - image2Pos.x) / relativeSize.x, (uvs.y
     - image2Pos.y) / relativeSize.y);
    //     colour += texture(image2, image2Coords).rgb;
    // }

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

1.9.8 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);
}
```

1.9.9 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);
}
```

1.9.10 blur.frag

See Section ??.

1.9.11 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;
}
```

1.9.12 calibrate.frag

```
#version 330 core

uniform sampler2D image;

in vec2 uvs;
out vec4 f_colour;

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

1.9.13 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;
}
```

1.9.14 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);
}
```

1.9.15 crt.frag

```
#version 330 core

precision mediump float;
uniform sampler2D image;

in vec2 uvs;
out vec4 f_colour;
uniform int mode = 1;

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;

    off_center *= 1.0 + 0.8 * pow(abs(off_center.yx), vec2(flatness));
    // 1.0 -> 1.5 make distance to screen
    // vec 2 -> screen 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);
      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;
    }
  }
}
```

1.9.16 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);
    // }
}
```

1.9.17 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);
}
```

1.9.18 highlight_brightness.frag

See Section ??.

1.9.19 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);
    }
}
```

1.9.20 lightmap.frag

See Section ??.

1.9.21 occlusion.frag

See Section ??.

1.9.22 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);
}
```

1.9.23 shadowmap.frag

See Section ??.

1.9.24 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);
}
```

1.10 data\shaders\vertex

1.10.1 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);
}
```

1.11 data\states

1.11.1 __init__.py

1.12 data\states\browser

1.12.1 browser.py

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

    def startup(self, persist=None):
        self.refresh_games_list() # BEFORE RESIZE TO FILL WIDGET BEFORE RESIZING
        super().startup(BROWSER_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])

    self._filter_column = 'number_of_ply'
    self._filter_ascend = False

    window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)

    BROWSER_WIDGETS['help'].kill()
    BROWSER_WIDGETS['browser_strip'].kill()

    self.draw()

def refresh_games_list(self):
    column_map = {
        'moves': 'number_of_ply',
        'winner': 'winner',
        'time': 'created_dt'
    }

    ascend_map = {
        'asc': True,
        'desc': False
    }

    filter_column = BROWSER_WIDGETS['filter_column_dropdown'].
get_selected_word()
    filter_ascend = BROWSER_WIDGETS['filter_ascend_dropdown'].
get_selected_word()

    self._selected_index = None

    start_row = (self._page_number - 1) * GAMES_PER_PAGE + 1
    end_row = (self._page_number) * GAMES_PER_PAGE
    self._games_list = get_ordered_games(column_map[filter_column], ascend_map
[filter_ascend], start_row=start_row, end_row=end_row)

    BROWSER_WIDGETS['browser_strip'].initialise_games_list(self._games_list)
    BROWSER_WIDGETS['browser_strip'].set_surface_size(window.size)
    BROWSER_WIDGETS['scroll_area'].set_image()

def get_event(self, event):
    widget_event = self._widget_group.process_event(event)

    if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
        BROWSER_WIDGETS['help'].kill()

    if widget_event is None:
        return

    match widget_event.type:
        case BrowserEventType.MENU_CLICK:
            self.next = 'menu'
            self.done = True

        case BrowserEventType.BROWSER_STRIP_CLICK:
            self._selected_index = widget_event.selected_index

        case BrowserEventType.COPY_CLICK:
            if self._selected_index is None:
                return
            logger.info('COPYING TO CLIPBOARD:', self._games_list[self.
_selected_index]['fen_string'])
```

```
                pyperclip.copy(self._games_list[self._selected_index]['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()
```

1.12.2 widget_dict.py

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

BROWSER_HEIGHT = 0.6

browser_strip = BrowserStrip(
    relative_position=(0.0, 0.0),
    relative_height=BROWSER_HEIGHT,
    games_list=[]
)

```

```python
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),
    )
}
```

1.13 data\states\config

1.13.1 config.py

```
import pygame
from data.constants import ConfigEventType, Colour, ShaderType
from data.states.config.default_config import default_config
from data.states.config.widget_dict import CONFIG_WIDGETS
from data.managers.logs import initialise_logger
from data.managers.animation import animation
from data.managers.window import window
from data.managers.audio import audio
from data.managers.theme import theme
from data.assets import MUSIC, SFX
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

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

        window.clear_apply_arguments(ShaderType.BLOOM)

        return self._config
```

```python

    def startup(self, persist=None):
        super().startup(CONFIG_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
        window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
    Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.9)

        CONFIG_WIDGETS['invalid_fen_string'].kill()
        CONFIG_WIDGETS['help'].kill()

        self._config = default_config

        if persist:
            self._config['FEN_STRING'] = persist

        self.set_fen_string(self._config['FEN_STRING'])
        self.toggle_pvc(self._config['CPU_ENABLED'])
        self.set_active_colour(self._config['COLOUR'])

        CONFIG_WIDGETS['cpu_depth_carousel'].set_to_key(self._config['CPU_DEPTH'])
        if self._config['CPU_ENABLED']:
            self.create_depth_picker()
        else:
            self.remove_depth_picker()

        self.draw()

    def create_depth_picker(self):
        # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.8))
        # CONFIG_WIDGETS['start_button'].set_image()
        CONFIG_WIDGETS['cpu_depth_carousel'].set_surface_size(window.size)
        CONFIG_WIDGETS['cpu_depth_carousel'].set_image()
        CONFIG_WIDGETS['cpu_depth_carousel'].set_geometry()
        self._widget_group.add(CONFIG_WIDGETS['cpu_depth_carousel'])

    def remove_depth_picker(self):
        # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.7))
        # CONFIG_WIDGETS['start_button'].set_image()

        CONFIG_WIDGETS['cpu_depth_carousel'].kill()

    def toggle_pvc(self, pvc_enabled):
        if pvc_enabled:
            CONFIG_WIDGETS['pvc_button'].set_locked(True)
            CONFIG_WIDGETS['pvp_button'].set_locked(False)
        else:
            CONFIG_WIDGETS['pvp_button'].set_locked(True)
            CONFIG_WIDGETS['pvc_button'].set_locked(False)

        self._config['CPU_ENABLED'] = pvc_enabled

        if self._config['CPU_ENABLED']:
            self.create_depth_picker()
        else:
            self.remove_depth_picker()

    def set_fen_string(self, new_fen_string):
        CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
        self._config['FEN_STRING'] = new_fen_string

        self.set_preset_overlay(new_fen_string)

        try:
            CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
            CONFIG_WIDGETS['invalid_fen_string'].kill()

            if new_fen_string[-1].lower() == 'r':
                self.set_active_colour(Colour.RED)
            else:
                self.set_active_colour(Colour.BLUE)

            self._valid_fen = True
        except:
            CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
            self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])

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

            audio.play_sfx(SFX['error_1'])
            audio.play_sfx(SFX['error_2'])

            self._valid_fen = False

    def get_event(self, event):
        widget_event = self._widget_group.process_event(event)

        if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
            CONFIG_WIDGETS['help'].kill()

        if widget_event is None:
            return

        match widget_event.type:
            case ConfigEventType.GAME_CLICK:
                if self._valid_fen:
                    self.next = 'game'
                    self.done = True

            case ConfigEventType.MENU_CLICK:
                self.next = 'menu'
                self.done = True

            case ConfigEventType.TIME_CLICK:
                self._config['TIME_ENABLED'] = not(widget_event.data)
                CONFIG_WIDGETS['timer_button'].set_next_icon()

            case ConfigEventType.PVP_CLICK:
                self.toggle_pvc(False)

            case ConfigEventType.PVC_CLICK:
                self.toggle_pvc(True)

            case ConfigEventType.FEN_STRING_TYPE:
                self.set_fen_string(widget_event.text)

            case ConfigEventType.TIME_TYPE:
                if widget_event.text == '':
                    self._config['TIME'] = 5
                else:
                    self._config['TIME'] = float(widget_event.text)

            case ConfigEventType.CPU_DEPTH_CLICK:
                self._config['CPU_DEPTH'] = int(widget_event.data)

            case ConfigEventType.PRESET_CLICK:
                self.set_fen_string(widget_event.fen_string)

            case ConfigEventType.SETUP_CLICK:
                self.next = 'editor'
                self.done = True

            case ConfigEventType.COLOUR_CLICK:
                self.set_active_colour(widget_event.data.get_flipped_colour())

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

    def set_preset_overlay(self, fen_string):
        fen_string_widget_map = {
            'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa b': 'preset_1',
            'sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3/10/2RaNaFaNa3Sa b': 'preset_2',
            'sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3RaNaPa3Pc/4Fa5/3PdNaPa3Sa b': 'preset_3'
        }

        if fen_string in fen_string_widget_map:
            self._selected_preset = CONFIG_WIDGETS[fen_string_widget_map[fen_string]]
        else:
            self._selected_preset = None

    def set_active_colour(self, colour):
        if self._config['COLOUR'] != colour:
            CONFIG_WIDGETS['to_move_button'].set_next_icon()

        self._config['COLOUR'] = colour

        if colour == Colour.BLUE:
            CONFIG_WIDGETS['to_move_text'].set_text('BLUE TO MOVE')
        elif colour == Colour.RED:
            CONFIG_WIDGETS['to_move_text'].set_text('RED TO MOVE')

        if self._valid_fen:
            self._config['FEN_STRING'] = self._config['FEN_STRING'][:-1] + colour.name[0].lower()
            CONFIG_WIDGETS['fen_string_input'].set_text(self._config['FEN_STRING'])

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

        if self._selected_preset:
            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)
```

1.13.2 default_config.py

```python
from data.constants import Colour

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

1.13.3 widget_dict.py

```python
import pygame
from data.widgets import *
from data.states.config.default_config import default_config
from data.components.custom_event import CustomEvent
from data.constants import ConfigEventType, Colour
from data.assets import GRAPHICS
from data.utils.asset_helpers import get_highlighted_icon
from data.managers.theme import theme

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['pharoah_0_a'], Colour.RED: GRAPHICS['pharoah_1_a']}
else:
    colour_icons = {Colour.RED: GRAPHICS['pharoah_1_a'], Colour.BLUE: GRAPHICS['pharoah_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/6pb3/7Pa2/2PdNaFaNa3Sa b',
    border_width=5,
    anchor_y='bottom',
    anchor_x='right',
    margin=20,
    event=CustomEvent(ConfigEventType.FEN_STRING_TYPE)
),
'start_button':
TextButton(
    parent=config_container,
    relative_position=(0, 0),
    relative_size=(0.9, 0.3),
    anchor_y='bottom',
    anchor_x='center',
    text='START NEW GAME',
        strength=0.1,
        text_colour=theme['textSecondary'],
        margin=20,
        fit_vertical=False,
        event=CustomEvent(ConfigEventType.GAME_CLICK)
    ),
    'timer_button':
    MultipleIconButton(
        parent=config_container,
        scale_mode='height',
        relative_position=(0.05, 0.3),
        relative_size=(0.15, 0.15),
        margin=10,
        border_width=5,
        border_radius=5,
        icons_dict=time_enabled_icons,
        event=CustomEvent(ConfigEventType.TIME_CLICK)
    ),
    'pvp_button':
    MultipleIconButton(
        parent=config_container,
        relative_position=(-0.225, 0.5),
        relative_size=(0.45, 0.15),
        margin=15,
        anchor_x='center',
        icons_dict=pvp_icons,
        stretch=False,
        event=CustomEvent(ConfigEventType.PVP_CLICK)
    ),
    'pvc_button':
    MultipleIconButton(
        parent=config_container,
        relative_position=(0.225, 0.5),
        relative_size=(0.45, 0.15),
        anchor_x='center',
        margin=15,
        icons_dict=pvc_icons,
        stretch=False,
        event=CustomEvent(ConfigEventType.PVC_CLICK)
    ),
    'invalid_fen_string':
    Text(
        parent=board_thumbnail,
        relative_position=(0, 0),
        relative_size=(0.9, 0.1),
        fit_vertical=False,
        anchor_x='center',
        anchor_y='center',
        text='INVALID FEN STRING!',
        margin=10,
        fill_colour=theme['fillError'],
        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)
                ),
            }
        )
}
```

1.14 data\states\editor

1.14.1 editor.py

```
import pygame
import pyperclip
from data.constants import EditorEventType, Colour, RotationDirection, Piece,
    Rotation
from data.states.game.components.bitboard_collection import BitboardCollection
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.states.game.components.father import DragAndDrop
from data.utils.bitboard_helpers import coords_to_bitboard
from data.states.editor.widget_dict import EDITOR_WIDGETS
from data.utils.board_helpers import screen_pos_to_coords
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

        self._drag_and_drop = None
        self._overlay_draw = None

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

        self.deselect_tool()

        return encode_fen_string(self._bitboards)
```

```python

    def startup(self, persist):
        super().startup(EDITOR_WIDGETS)
        EDITOR_WIDGETS['help'].kill()

        self._drag_and_drop = DragAndDrop(EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
        self._overlay_draw = OverlayDraw(EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
        self._bitboards = BitboardCollection(persist['FEN_STRING'])
        self._piece_group = PieceGroup()

        self._selected_coords = None
        self._selected_tool = None
        self._selected_tool_colour = None
        self._initial_fen_string = persist['FEN_STRING']
        self._starting_colour = Colour.BLUE

        self.refresh_pieces()
        self.set_starting_colour(Colour.BLUE if persist['FEN_STRING'][-1].lower() == 'b' else Colour.RED)
        self.draw()

    @property
    def selected_coords(self):
        return self._selected_coords

    @selected_coords.setter
    def selected_coords(self, new_coords):
        self._overlay_draw.set_selected_coords(new_coords)
        self._selected_coords = new_coords

    def get_event(self, event):
        widget_event = self._widget_group.process_event(event)

        if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
            EDITOR_WIDGETS['help'].kill()

        if event.type == pygame.MOUSEBUTTONDOWN:
            clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)

            if clicked_coords:
                self.selected_coords = clicked_coords

                if self._selected_tool is None:
                    return

                if self._selected_tool == 'MOVE':
                    self.set_dragged_piece(clicked_coords)

                elif self._selected_tool == 'ERASE':
                    self.remove_piece()
                else:
                    self.set_piece(self._selected_tool, self._selected_tool_colour, Rotation.UP)

                return

        if event.type == pygame.MOUSEBUTTONUP:
            clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)

        if self._drag_and_drop.dragged_sprite:
            self.remove_dragged_piece(clicked_coords)
            return

    if widget_event is None:
        if event.type == pygame.MOUSEBUTTONDOWN and self._widget_group.on_widget(event.pos) is False:
            self.selected_coords = None

        return

    match widget_event.type:
        case None:
            return

        case EditorEventType.MENU_CLICK:
            self.next = 'menu'
            self.done = True

        case EditorEventType.PICK_PIECE_CLICK:
            if widget_event.piece == self._selected_tool and widget_event.active_colour == self._selected_tool_colour:
                self.deselect_tool()
            else:
                self.select_tool(widget_event.piece, widget_event.active_colour)

        case EditorEventType.ROTATE_PIECE_CLICK:
            self.rotate_piece(widget_event.rotation_direction)

        case EditorEventType.EMPTY_CLICK:
            self._bitboards = BitboardCollection(fen_string='sc9/10/10/10/10/10/10/9Sa b')
            self.refresh_pieces()

        case EditorEventType.RESET_CLICK:
            self.reset_board()

        case EditorEventType.COPY_CLICK:
            logger.info(f'COPYING TO CLIPBOARD: {encode_fen_string(self._bitboards)}')
            pyperclip.copy(encode_fen_string(self._bitboards))

        case EditorEventType.BLUE_START_CLICK:
            self.set_starting_colour(Colour.BLUE)

        case EditorEventType.RED_START_CLICK:
            self.set_starting_colour(Colour.RED)

        case EditorEventType.START_CLICK:
            self.next = 'config'
            self.done = True

        case EditorEventType.CONFIG_CLICK:
            self.reset_board()
            self.next = 'config'
            self.done = True

        case EditorEventType.ERASE_CLICK:
            if self._selected_tool == 'ERASE':
                self.deselect_tool()
                else:
                    self.select_tool('ERASE', None)

            case EditorEventType.MOVE_CLICK:
                if self._selected_tool == 'MOVE':
                    self.deselect_tool()
                else:
                    self.select_tool('MOVE', None)

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

    def reset_board(self):
        self._bitboards = BitboardCollection(self._initial_fen_string)
        self.refresh_pieces()

    def refresh_pieces(self):
        self._piece_group.initialise_pieces(self._bitboards.convert_to_piece_list(), EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)

    def set_starting_colour(self, new_colour):
        if new_colour == Colour.BLUE:
            EDITOR_WIDGETS['blue_start_button'].set_locked(True)
            EDITOR_WIDGETS['red_start_button'].set_locked(False)
        elif new_colour == Colour.RED:
            EDITOR_WIDGETS['blue_start_button'].set_locked(False)
            EDITOR_WIDGETS['red_start_button'].set_locked(True)

        if new_colour != self._starting_colour:
            EDITOR_WIDGETS['blue_start_button'].set_next_icon()
            EDITOR_WIDGETS['red_start_button'].set_next_icon()

        self._starting_colour = new_colour
        self._bitboards.active_colour = new_colour

    def set_dragged_piece(self, coords):
        bitboard_under_mouse = coords_to_bitboard(coords)
        dragged_piece = self._bitboards.get_piece_on(bitboard_under_mouse, Colour.BLUE) or self._bitboards.get_piece_on(bitboard_under_mouse, Colour.RED)

        if dragged_piece is None:
            return

        dragged_colour = self._bitboards.get_colour_on(bitboard_under_mouse)
        dragged_rotation = self._bitboards.get_rotation_on(bitboard_under_mouse)

        self._drag_and_drop.set_dragged_piece(dragged_piece, dragged_colour, dragged_rotation)
        self._overlay_draw.set_hover_limit(False)

    def remove_dragged_piece(self, coords):
        piece, colour, rotation = self._drag_and_drop.get_dragged_info()

        if coords and coords != self._selected_coords and piece != Piece.SPHINX:
            self.remove_piece()
            self.selected_coords = coords
            self.set_piece(piece, colour, rotation)
            self.selected_coords = None

        self._drag_and_drop.remove_dragged_piece()
        self._overlay_draw.set_hover_limit(True)

    def set_piece(self, piece, colour, rotation):
        if self.selected_coords is None or self.selected_coords == (0, 7) or self.selected_coords == (9, 0):
            return

        self.remove_piece()

        selected_bitboard = coords_to_bitboard(self.selected_coords)
        self._bitboards.set_square(selected_bitboard, piece, colour)
        self._bitboards.set_rotation(selected_bitboard, rotation)

        self.refresh_pieces()

    def remove_piece(self):
        if self.selected_coords is None or self.selected_coords == (0, 7) or self.selected_coords == (9, 0):
            return

        selected_bitboard = coords_to_bitboard(self.selected_coords)
        self._bitboards.clear_square(selected_bitboard, Colour.BLUE)
        self._bitboards.clear_square(selected_bitboard, Colour.RED)
        self._bitboards.clear_rotation(selected_bitboard)

        self.refresh_pieces()

    def rotate_piece(self, rotation_direction):
        if self.selected_coords is None or self.selected_coords == (0, 7) or self.selected_coords == (9, 0):
            return

        selected_bitboard = coords_to_bitboard(self.selected_coords)

        if self._bitboards.get_piece_on(selected_bitboard, Colour.BLUE) is None and self._bitboards.get_piece_on(selected_bitboard, Colour.RED) is None:
            return

        current_rotation = self._bitboards.get_rotation_on(selected_bitboard)

        if rotation_direction == RotationDirection.CLOCKWISE:
            self._bitboards.update_rotation(selected_bitboard, selected_bitboard, current_rotation.get_clockwise())
        elif rotation_direction == RotationDirection.ANTICLOCKWISE:
            self._bitboards.update_rotation(selected_bitboard, selected_bitboard, current_rotation.get_anticlockwise())

        self.refresh_pieces()

    def select_tool(self, piece, colour):
        dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: 'red_piece_buttons' }

        self.deselect_tool()

        if piece == 'ERASE':
            EDITOR_WIDGETS['erase_button'].set_locked(True)
            EDITOR_WIDGETS['erase_button'].set_next_icon()
        elif piece == 'MOVE':
            EDITOR_WIDGETS['move_button'].set_locked(True)
            EDITOR_WIDGETS['move_button'].set_next_icon()
        else:
            EDITOR_WIDGETS[dict_name_map[colour]][piece].set_locked(True)
```

```python
            EDITOR_WIDGETS[dict_name_map[colour]][piece].set_next_icon()

        self._selected_tool = piece
        self._selected_tool_colour = colour

    def deselect_tool(self):
        dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: 'red_piece_buttons' }

        if self._selected_tool:
            if self._selected_tool == 'ERASE':
                EDITOR_WIDGETS['erase_button'].set_locked(False)
                EDITOR_WIDGETS['erase_button'].set_next_icon()
            elif self._selected_tool == 'MOVE':
                EDITOR_WIDGETS['move_button'].set_locked(False)
                EDITOR_WIDGETS['move_button'].set_next_icon()
            else:
                EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self._selected_tool].set_locked(False)
                EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self._selected_tool].set_next_icon()

        self._selected_tool = None
        self._selected_tool_colour = None

    def handle_resize(self):
        super().handle_resize()
        self._piece_group.handle_resize(EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
        self._drag_and_drop.handle_resize(EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
        self._overlay_draw.handle_resize(EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)

    def draw(self):
        self._widget_group.draw()
        self._overlay_draw.draw(window.screen)
        self._piece_group.draw(window.screen)
        self._drag_and_drop.draw(window.screen)
```

1.14.2 widget_dict.py

```python
from data.constants import Piece, Colour, RotationDirection, EditorEventType, BLUE_BUTTON_COLOURS
from data.utils.asset_helpers import get_highlighted_icon
from data.components.custom_event import CustomEvent
from data.assets import GRAPHICS
from data.widgets import *

blue_pieces_container = Rectangle(
    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',
            anchor_x='center',
            base_icon=GRAPHICS['discard_arrow_base'],
            hover_icon=GRAPHICS['discard_arrow_hover'],
            press_icon=GRAPHICS['discard_arrow_press'],
            event=CustomEvent(EditorEventType.RESET_CLICK),
        ),
        ReactiveIconButton(
            relative_position=(0, 0),
            fixed_position=(10, 0),
            relative_size=(0.1, 0.1),
            anchor_x='right',
            anchor_y='center',
            scale_mode='height',
            base_icon=GRAPHICS['play_arrow_base'],
            hover_icon=GRAPHICS['play_arrow_hover'],
            press_icon=GRAPHICS['play_arrow_press'],
            event=CustomEvent(EditorEventType.START_CLICK),
        ),
        ReactiveIconButton(
            relative_position=(0, 0),
            fixed_position=(10, 0),
            relative_size=(0.1, 0.1),
            anchor_y='center',
            scale_mode='height',
            base_icon=GRAPHICS['return_arrow_base'],
            hover_icon=GRAPHICS['return_arrow_hover'],
            press_icon=GRAPHICS['return_arrow_press'],
            event=CustomEvent(EditorEventType.CONFIG_CLICK),
        )
    ],
    'blue_piece_buttons': {},
    'red_piece_buttons': {},
    'erase_button':
    MultipleIconButton(
        parent=red_pieces_container,
        relative_position=(0, 0),
        relative_size=(0.2, 0.2),
        scale_mode='height',
        margin=10,
        icons_dict={True: GRAPHICS['eraser'], False: get_highlighted_icon(GRAPHICS
    ['eraser'])},
        event=CustomEvent(EditorEventType.ERASE_CLICK),
    ),
    'move_button':
    MultipleIconButton(
        parent=blue_pieces_container,
        relative_position=(0, 0),
        relative_size=(0.2, 0.2),
        scale_mode='height',
        box_colours=BLUE_BUTTON_COLOURS,
        icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
    ['finger'])},
        event=CustomEvent(EditorEventType.MOVE_CLICK),
    ),
    'chessboard':
    Chessboard(
        relative_position=(0, 0),
        relative_width=0.4,
        scale_mode='width',
        anchor_x='center',
        anchor_y='center'
    ),
    '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['pharoah_0_a']), True:
    GRAPHICS['pharoah_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['pharoah_1_a'], False: get_highlighted_icon(
    GRAPHICS['pharoah_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)
    )
```

1.15 data\states\game

1.15.1 game.py

```
import pygame
from functools import partial
from data.states.game.mvc.game_controller import GameController
from data.utils.database_helpers import insert_into_games
from data.states.game.mvc.game_model import GameModel
from data.states.game.mvc.pause_view import PauseView
from data.states.game.mvc.game_view import GameView
from data.states.game.mvc.win_view import WinView
from data.components.game_entry import GameEntry
from data.managers.logs import initialise_logger
from data.managers.window import window
from data.managers.audio import audio
from data.constants import ShaderType
from data.assets import MUSIC, SFX
from data.control import _State

logger = initialise_logger(__name__)

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

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

        window.clear_apply_arguments(ShaderType.BLOOM)
        window.clear_effect(ShaderType.RAYS)

        game_entry = GameEntry(self.model.states, final_fen_string=self.model.
    get_fen_string())
        inserted_game = insert_into_games(game_entry.convert_to_row())

        return inserted_game

    def switch_to_menu(self):
        self.next = 'menu'
        self.done = True

    def switch_to_review(self):
        self.next = 'review'
        self.done = True

    def startup(self, persist):
        music = MUSIC[['cpu_easy', 'cpu_medium', 'cpu_hard'][persist['CPU_DEPTH']
    - 2]] if persist['CPU_ENABLED'] else MUSIC['pvp']
        super().startup(music=music)

        window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
    BACKGROUND_LASERS)
        window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
    Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
        binded_startup = partial(self.startup, persist)

        self.model = GameModel(persist)
        self.view = GameView(self.model)
        self.pause_view = PauseView(self.model)
        self.win_view = WinView(self.model)
        self.controller = GameController(self.model, self.view, self.win_view,
    self.pause_view, self.switch_to_menu, self.switch_to_review, binded_startup)

        self.view.draw()

        audio.play_sfx(SFX['game_start_1'])
        audio.play_sfx(SFX['game_start_2'])

    def get_event(self, event):
        self.controller.handle_event(event)

    def handle_resize(self):
        self.view.handle_resize()
        self.win_view.handle_resize()
        self.pause_view.handle_resize()

    def draw(self):
        self.view.draw()
        self.win_view.draw()
        self.pause_view.draw()

    def update(self):
        self.controller.check_cpu()
        super().update()
```

1.15.2 widget_dict.py

```
from data.widgets import *
from data.components.custom_event import CustomEvent
from data.constants import GameEventType, RotationDirection, Colour
from data.assets import GRAPHICS

right_container = Rectangle(
    relative_position=(0.05, 0),
    relative_size=(0.2, 0.5),
    anchor_y='center',
    anchor_x='right',
)

rotate_container = Rectangle(
    relative_position=(0, 0.05),
    relative_size=(0.2, 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=[]
)

resign_button = TextButton(
    parent=right_container,
    relative_position=(0, 0),
    relative_size=(0.5, 0.2),
    fit_vertical=False,
    anchor_y='bottom',
    text="   Resign",
    margin=5,
    event=CustomEvent(GameEventType.RESIGN_CLICK)
)

draw_button = TextButton(
    parent=right_container,
    relative_position=(0, 0),
    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,
            relative_position=(0.33, 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(GameEventType.HELP_CLICK)
        ),
        ReactiveIconButton(
            parent=rotate_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            anchor_x='right',
            base_icon=GRAPHICS['clockwise_arrow_base'],
            hover_icon=GRAPHICS['clockwise_arrow_hover'],
            press_icon=GRAPHICS['clockwise_arrow_press'],
            event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
    RotationDirection.CLOCKWISE)
        ),
        ReactiveIconButton(
            parent=rotate_container,
            relative_position=(0, 0),
            relative_size=(1, 1),
            scale_mode='height',
            base_icon=GRAPHICS['anticlockwise_arrow_base'],
            hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
            press_icon=GRAPHICS['anticlockwise_arrow_press'],
            event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
    RotationDirection.ANTICLOCKWISE)
        ),
        resign_button,
        draw_button,
        Icon(
            parent=resign_button,
            relative_position=(0, 0),
            relative_size=(0.75, 0.75),
            fill_colour=(0, 0, 0, 0),
            scale_mode='height',
            anchor_y='center',
            border_radius=0,
            border_width=0,
            margin=5,
            icon=GRAPHICS['resign']
        ),
        Icon(
            parent=draw_button,
            relative_position=(0, 0),
            relative_size=(0.75, 0.75),
            fill_colour=(0, 0, 0, 0),
            scale_mode='height',
            anchor_y='center',
            border_radius=0,
            border_width=0,
            margin=5,
            icon=GRAPHICS['draw']
        ),
    ],
    'scroll_area': # REMEMBER SCROLL AREA AFTER CONTAINER FOR RESIZING
    ScrollArea(
        parent=right_container,
        relative_position=(0, 0),
        relative_size=(1, 0.8),
        vertical=True,
        widget=move_list
    ),
    'move_list':
        move_list,
    'blue_timer':
    Timer(
        relative_position=(0.05, 0.05),
        anchor_y='center',
        relative_size=(0.1, 0.1),
        active_colour=Colour.BLUE,
        event=CustomEvent(GameEventType.TIMER_END),
    ),
    'red_timer':
    Timer(
        relative_position=(0.05, -0.05),
        anchor_y='center',
        relative_size=(0.1, 0.1),
        active_colour=Colour.RED,
        event=CustomEvent(GameEventType.TIMER_END),
    ),
    'status_text':
    Text(
        relative_position=(0, 0.05),
        relative_size=(0.4, 0.1),
        anchor_x='center',
        fit_vertical=False,
        margin=10,
        text="g",
        minimum_width=400
    ),
    'chessboard':
    Chessboard(
        relative_position=(0, 0),
        anchor_x='center',
        anchor_y='center',
        scale_mode='width',
        relative_width=0.4
    ),
    'blue_piece_display':
    PieceDisplay(
        relative_position=(0.05, 0.05),
        relative_size=(0.2, 0.1),
        anchor_y='bottom',
        active_colour=Colour.BLUE
    ),
    'red_piece_display':
    PieceDisplay(
        relative_position=(0.05, 0.05),
        relative_size=(0.2, 0.1),
        active_colour=Colour.RED
    )
}

PAUSE_WIDGETS = {
    'default': [
        TextButton(
            relative_position=(0, -0.125),
            relative_size=(0.3, 0.2),
            anchor_x='center',
            anchor_y='center',
            text='GO TO MENU',
            fit_vertical=False,
            event=CustomEvent(GameEventType.MENU_CLICK)
        ),
        TextButton(
            relative_position=(0, 0.125),
            relative_size=(0.3, 0.2),
            anchor_x='center',
            anchor_y='center',
            text='RESUME GAME',
            fit_vertical=False,
            event=CustomEvent(GameEventType.PAUSE_CLICK)
        )
    ]
}

win_container = Rectangle(
    relative_position=(0, 0),
    relative_size=(0.4, 0.8),
    scale_mode='height',
    anchor_x='center',
    anchor_y='center',
    fill_colour=(128, 128, 128, 200),
    visible=True
)

WIN_WIDGETS = {
    'default': [
        win_container,
        TextButton(
            parent=win_container,
            relative_position=(0, 0.5),
            relative_size=(0.8, 0.15),
            text='GO TO MENU',
            anchor_x='center',
            fit_vertical=False,
            event=CustomEvent(GameEventType.MENU_CLICK)
        ),
        TextButton(
            parent=win_container,
            relative_position=(0, 0.65),
            relative_size=(0.8, 0.15),
            text='REVIEW GAME',
            anchor_x='center',
            fit_vertical=False,
            event=CustomEvent(GameEventType.REVIEW_CLICK)
        ),
        TextButton(
            parent=win_container,
            relative_position=(0, 0.8),
            relative_size=(0.8, 0.15),
            text='NEW GAME',
            anchor_x='center',
            fit_vertical=False,
            event=CustomEvent(GameEventType.GAME_CLICK)
        ),
    ],
    'blue_won':
    Icon(
        parent=win_container,
        relative_position=(0, 0.05),
        relative_size=(0.8, 0.3),
        anchor_x='center',
        border_width=0,
        margin=0,
        icon=GRAPHICS['blue_won'],
        fill_colour=(0, 0, 0, 0),
    ),
    'red_won':
    Icon(
        parent=win_container,
        relative_position=(0, 0.05),
        relative_size=(0.8, 0.3),
        anchor_x='center',
        border_width=0,
        margin=0,
        icon=GRAPHICS['red_won'],
        fill_colour=(0, 0, 0, 0),
        fit_icon=True,
    ),
    'draw_won':
    Icon(
        parent=win_container,
        relative_position=(0, 0.05),
        relative_size=(0.8, 0.3),
        anchor_x='center',
        border_width=0,
        margin=0,
        icon=GRAPHICS['draw_won'],
        fill_colour=(0, 0, 0, 0),
    ),
    'by_checkmate':
    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_checkmate'],
        fill_colour=(0, 0, 0, 0),
    ),
    'by_resignation':
    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_resignation'],
        fill_colour=(0, 0, 0, 0),
    ),
    'by_draw':
    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_draw'],
        fill_colour=(0, 0, 0, 0),
    ),
    '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),
    )
}
```

1.16 data\states\game\components

1.16.1 bitboard_collection.py

See Section ??.

1.16.2 board.py

See Section ??.

1.16.3 capture_draw.py

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

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
```

1.16.4 father.py

```python
import pygame
from data.constants import CursorMode
from data.states.game.components.piece_sprite import PieceSprite
from data.managers.cursor import cursor
from data.managers.audio import audio
from data.assets import SFX

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)

    def handle_resize(self, board_position, board_size):
        if self.dragged_sprite:
            self.dragged_sprite.set_geometry(board_position, board_size[0] / 10)

        self._board_position = board_position
        self._board_size = board_size
```

1.16.5 fen_parser.py

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

```python

def parse_fen_string(fen_string):
    #sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa b
    piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char: EMPTY_BB for char in Piece}]
    rotation_bitboards = [EMPTY_BB, EMPTY_BB]
    combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
    combined_all_bitboard = 0
    part_1, part_2 = fen_string.split(' ')

    rank = 7
    file = 0

    piece_count = {char.lower(): 0 for char in Piece} | {char.upper(): 0 for char in Piece}

    for index, character in enumerate(part_1):
        square = rank * 10 + file

        if character.lower() in Piece:
            piece_count[character] += 1
            if character.isupper():
                piece_bitboards[Colour.BLUE][character.lower()] |= 1 << square

            else:
                piece_bitboards[Colour.RED][character.lower()] |= 1 << square

            rotation = part_1[index + 1]
            match rotation:
                case Rotation.UP:
                    pass
                case Rotation.RIGHT:
                    rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
                case Rotation.DOWN:
                    rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
                case Rotation.LEFT:
                    rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
                    rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
                case _:
                    raise ValueError('Invalid FEN String - piece character not followed by rotational character')

            file += 1
        elif character in '0123456789':
            if character == '1' and fen_string[index + 1] == '0':
                file += 10
                continue

            file += int(character)
        elif character == '/':
            rank = rank - 1
            file = 0
        elif character in Rotation:
            continue
        else:
            raise ValueError('Invalid FEN String - invalid character found:', character)

    if piece_count['s'] != 1 or piece_count['S'] != 1:
        raise ValueError('Invalid FEN string - invalid number of Sphinx pieces')
    # COMMENTED OUT AS NO PHAROAH PIECES IS OKAY IF PARSING FEN STRING FOR
    FINISHED GAME BOARD THUMBNAIL
    elif piece_count['f'] > 1 or piece_count['F'] > 1:
        raise ValueError('Invalid FEN string - invalid number of Pharoah pieces')

    if part_2 == 'b':
        colour = Colour.BLUE
    elif part_2 == 'r':
        colour = Colour.RED
    else:
        raise ValueError('Invalid FEN string - invalid active colour')

    for piece in Piece:
        combined_colour_bitboards[Colour.BLUE] |= piece_bitboards[Colour.BLUE][piece]
        combined_colour_bitboards[Colour.RED] |= piece_bitboards[Colour.RED][piece]

    combined_all_bitboard = combined_colour_bitboards[Colour.BLUE] | combined_colour_bitboards[Colour.RED]
    return (piece_bitboards, combined_colour_bitboards, combined_all_bitboard, rotation_bitboards, colour)

def encode_fen_string(bitboard_collection):
    blue_bitboards = bitboard_collection.piece_bitboards[Colour.BLUE]
    red_bitboards = bitboard_collection.piece_bitboards[Colour.RED]

    fen_string_list = [''] * 80

    for piece, bitboard in blue_bitboards.items():
        for individual_bitboard in occupied_squares(bitboard):
            index = bitboard_to_index(individual_bitboard)
            rotation = bitboard_collection.get_rotation_on(individual_bitboard)
            fen_string_list[index] = piece.upper() + rotation

    for piece, bitboard in red_bitboards.items():
        for individual_bitboard in occupied_squares(bitboard):
            index = bitboard_to_index(individual_bitboard)
            rotation = bitboard_collection.get_rotation_on(individual_bitboard)
            fen_string_list[index] = piece.lower() + rotation

    fen_string = ''
    row_string = ''
    empty_count = 0
    for index, square in enumerate(fen_string_list):
        if square == '':
            empty_count += 1
        else:
            if empty_count > 0:
                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
```

1.16.6 laser.py

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

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.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)

            if previous_direction != direction:
                pieces_on_trajectory.append(current_square)

            if next_square == EMPTY_BB:
                hit_square_bitboard = 0b0

                if piece_hit:
                    hit_square_bitboard = current_square

                return hit_square_bitboard, piece_hit, list(zip(trajectory_list,
    square_animation_states)), trajectory_bitboard, pieces_on_trajectory

            current_square = next_square
            previous_direction = direction

    def calculate_next_square(self, square, piece, rotation, previous_direction):
        match piece:
            case Piece.SPHINX:
                if previous_direction != rotation:
                    return EMPTY_BB, previous_direction, None

                next_square = self.next_square_bitboard(square, rotation)
                return next_square, previous_direction, Piece.SPHINX

            case Piece.PYRAMID:
                if previous_direction in [rotation, rotation.get_clockwise()]:
                    return EMPTY_BB, previous_direction, Piece.PYRAMID

                if previous_direction == rotation.get_anticlockwise():
                    new_direction = previous_direction.get_clockwise()
                else:
                    new_direction = previous_direction.get_anticlockwise()

                next_square = self.next_square_bitboard(square, new_direction)

                return next_square, new_direction, None

            case Piece.ANUBIS:
                if previous_direction == rotation.get_clockwise().get_clockwise():
                    return EMPTY_BB, previous_direction, None

                return EMPTY_BB, previous_direction, Piece.ANUBIS

            case Piece.SCARAB:
                if previous_direction in [rotation.get_clockwise(), rotation.get_anticlockwise()]:
                    new_direction = previous_direction.get_anticlockwise()
                else:
                    new_direction = previous_direction.get_clockwise()

                next_square = self.next_square_bitboard(square, new_direction)

                return next_square, new_direction, None

            case Piece.PHAROAH:
                return EMPTY_BB, previous_direction, Piece.PHAROAH

            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
```

1.16.7 laser_draw.py

See Section ??.

1.16.8 move.py

```python
from data.constants import MoveType, Colour, RotationDirection
from data.utils.bitboard_helpers import notation_to_bitboard, coords_to_bitboard,
    bitboard_to_coords, bitboard_to_notation, print_bitboard
import re
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 = ''
        coords_1 = '(' + chr(bitboard_to_coords(self.src)[0] + 65) + ',' + str(
    bitboard_to_coords(self.src)[1] + 1) + ')'

        if self.move_type == MoveType.ROTATE:
            rotate_text = ' ' + self.rotation_direction.name
            return f'{self.move_type.name}{rotate_text}: ON {coords_1}'

        elif self.move_type == MoveType.MOVE:
            coords_2 = '(' + chr(bitboard_to_coords(self.dest)[0] + 65) + ', ' +
    str(bitboard_to_coords(self.dest)[1] + 1) + ')'
            return f'{self.move_type.name}{rotate_text}: FROM {coords_1} TO {
    coords_2}'

        # (Rotation: {self.rotation_direction})

    @classmethod
    def instance_from_notation(move_cls, notation):
        try:
            notation = notation.split('x')[0]
            move_type = notation[0].lower()

        moves = notation[2:]
        letters = re.findall(r'[A-Za-z]+', moves)
        numbers = re.findall(r'\d+', moves)

        if move_type == MoveType.MOVE:
            src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
            dest_bitboard = notation_to_bitboard(letters[1] + numbers[1])

            return move_cls(move_type, src_bitboard, dest_bitboard)

        elif move_type == MoveType.ROTATE:
            src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
            rotation_direction = RotationDirection(letters[1])

            return move_cls(move_type, src_bitboard, src_bitboard,
rotation_direction)
        else:
            raise ValueError('(Move.instance_from_notation) Invalid move type:
', move_type)

    except Exception as error:
        logger.info('(Move.instance_from_notation) Error occured while parsing
:', error)
        raise error

@classmethod
def instance_from_input(move_cls, move_type, src, dest=None, rotation=None):
    try:
        if move_type == MoveType.MOVE:
            src_bitboard = notation_to_bitboard(src)
            dest_bitboard = notation_to_bitboard(dest)

        elif move_type == MoveType.ROTATE:
            src_bitboard = notation_to_bitboard(src)
            dest_bitboard = src_bitboard

        return move_cls(move_type, src_bitboard, dest_bitboard, rotation)
    except Exception as error:
        logger.info('Error (Move.instance_from):', error)
        raise error

@classmethod
def instance_from_coords(move_cls, move_type, src_coords, dest_coords=None,
rotation_direction=None):
    try:
        src_bitboard = coords_to_bitboard(src_coords)
        dest_bitboard = coords_to_bitboard(dest_coords)

        return move_cls(move_type, src_bitboard, dest_bitboard,
rotation_direction)
    except Exception as error:
        logger.info('Error (Move.instance_from_coords):', error)
        raise error

@classmethod
def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
None, rotation_direction=None):
    try:
        return move_cls(move_type, src_bitboard, dest_bitboard,
rotation_direction)
    except Exception as error:
            logger.info('Error (Move.instance_from_bitboards):', error)
            raise error
```

1.16.9 overlay_draw.py

```
import pygame
from data.constants import OVERLAY_COLOUR_LIGHT, OVERLAY_COLOUR_DARK
from data.utils.board_helpers import coords_to_screen_pos, screen_pos_to_coords,
    create_square_overlay, create_circle_overlay

class OverlayDraw:
    def __init__(self, board_position, board_size, limit_hover=True):
        self._board_position = board_position
        self._board_size = board_size

        self._hovered_coords = None
        self._selected_coords = None
        self._available_coords = None

        self._limit_hover = limit_hover

        self._selected_overlay = None
        self._hovered_overlay = None
        self._available_overlay = None

        self.initialise_overlay_surfaces()

    @property
    def square_size(self):
        return self._board_size[0] / 10

    def initialise_overlay_surfaces(self):
        self._selected_overlay = create_square_overlay(self.square_size,
    OVERLAY_COLOUR_DARK)
        self._hovered_overlay = create_square_overlay(self.square_size,
    OVERLAY_COLOUR_LIGHT)
        self._available_overlay = create_circle_overlay(self.square_size,
    OVERLAY_COLOUR_LIGHT)

    def set_hovered_coords(self, mouse_pos):
        self._hovered_coords = screen_pos_to_coords(mouse_pos, self.
    _board_position, self._board_size)

    def set_selected_coords(self, coords):
        self._selected_coords = coords

    def set_available_coords(self, coords_list):
        self._available_coords = coords_list

    def set_hover_limit(self, new_limit):
        self._limit_hover = new_limit

    def draw(self, screen):
        self.set_hovered_coords(pygame.mouse.get_pos())

        if self._selected_coords:
            screen.blit(self._selected_overlay, coords_to_screen_pos(self.
    _selected_coords, self._board_position, self.square_size))

        if self._available_coords:
            for coords in self._available_coords:
                screen.blit(self._available_overlay, coords_to_screen_pos(coords, self._board_position, self.square_size))

        if self._hovered_coords:
            if self._hovered_coords is None:
                return

            if self._limit_hover and ((self._available_coords is None) or (self._hovered_coords not in self._available_coords)):
                return

            screen.blit(self._hovered_overlay, coords_to_screen_pos(self._hovered_coords, self._board_position, self.square_size))

    def handle_resize(self, board_position, board_size):
        self._board_position = board_position
        self._board_size = board_size

        self.initialise_overlay_surfaces()
```

1.16.10 particles_draw.py

See Section ??.

1.16.11 piece_group.py

```
import pygame
from data.constants import EMPTY_BB, Colour, Piece
from data.states.game.components.piece_sprite import PieceSprite
from data.utils.board_helpers import coords_to_screen_pos
from data.utils import bitboard_helpers as bb_helpers

class PieceGroup(pygame.sprite.Group):
    def __init__(self):
        # self.square_list = []
        # self.valid_square_list_positions = []
        super().__init__()

    def initialise_pieces(self, piece_list, board_position, board_size):
        self.empty()

        for index, piece_and_rotation in enumerate(piece_list):
            x = index % 10
            y = index // 10

            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()

    # def handle_resize_end(self):
    #     for sprite in self.sprites():
    #         sprite.handle_resize_end()

    # def clear_square(self, src_bitboard):
    #     list_position = bb_helpers.bitboard_to_index(src_bitboard)
    #     self.square_list[list_position].clear_piece()

    # def update_squares_move(self, src, dest, new_piece_symbol, new_colour,
    rotation):
    #     self.square_list[src].clear_piece()
    #     self.square_list[dest].clear_piece()
    #     self.square_list[dest].set_piece(piece_symbol=new_piece_symbol, colour=
    new_colour, rotation=rotation)

    # def update_squares_rotate(self, src, piece_symbol, colour, new_rotation):
    #     self.square_list[src].clear_piece()
    #     self.square_list[src].set_piece(piece_symbol=piece_symbol, colour=colour
    , rotation=new_rotation)

    # def add_valid_square_overlays(self, valid_bitboard):
    #     if valid_bitboard == EMPTY_BB:
    #         return

    #     list_positions = self.bitboard_to_list_positions(valid_bitboard)
    #     self.valid_square_list_positions = list_positions

    #     for square_position in list_positions:
    #         square = self.square_list[square_position]
    #         square.selected = True

    # def remove_valid_square_overlays(self):
    #     for square_position in self.valid_square_list_positions:
    #         square = self.square_list[square_position]
    #         square.selected = False
    #         square.remove_overlay()

    #     self.valid_square_list_positions = []

    # def draw_valid_square_overlays(self):
    #     for square_position in self.valid_square_list_positions:
    #         square = self.square_list[square_position]
    #         square.draw_overlay()

    # def bitboard_to_list_positions(self, bitboard):
    #     list_positions = []

    #     for square in bb_helpers.occupied_squares(bitboard):
    #         list_positions.append(bb_helpers.bitboard_to_index(square))

    #     return list_positions
```

1.16.12 piece_sprite.py

```python
import pygame
from data.assets import GRAPHICS
from data.constants import Colour, Piece
from data.utils.asset_helpers import scale_and_cache
from data.utils.board_helpers import coords_to_screen_pos

class EmptyPiece(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()

        self.image = pygame.Surface((1, 1))
        self.rect = self.image.get_rect()
        self.rect.topleft = (0, 0)

    def set_image(self, type):
        pass

    def set_rect(self):
        pass

    def set_geometry(self, anchor_position, size):
        pass

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
```

1.16.13 psqt.py

```python
from data.constants 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.PHAROAH: [
        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,
    ],
}
```

1.17 data\states\game\cpu

1.17.1 arena.py

```python
from data.states.game.cpu.engines import *
from data.states.game.components.board import Board
from data.constants import Colour, Miscellaneous
from data.managers.logs import initialise_logger

logger = initialise_logger(__name__)
# sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa
    b
# scfaRa7/RaRaRaFa6/RaRaRa7/10/10/10/10/9Sa b
# scfa8/10/10/10/10/10/10/8FaSa b

def compare(cls1, cls2, depth, rounds):
    wins = [0, 0]

    board = Board()
    def callback(move):
        board.apply_move(move, add_hash=True)

    cpu1 = cls1(callback=callback, max_depth=depth, verbose='compact')
    cpu2 = cls2(callback=callback, max_depth=depth, verbose='compact')

    for i in range(rounds):
        board = Board(fen_string="scfa8/10/10/10/10/10/10/8FaSa b")
        ply = 0

        if i % 2 == 0:
            players = { Colour.BLUE: cpu1, Colour.RED: cpu2, Miscellaneous.DRAW: '
    DRAW' }
        else:
            players = { Colour.BLUE: cpu2, Colour.RED: cpu1, Miscellaneous.DRAW: '
    DRAW' }

        while (winner := board.check_win()) is None:
            players[board.get_active_colour()].find_move(board, None)
            ply += 1
            logger.debug('PLY:', ply)

        if winner == Miscellaneous.DRAW:
            wins[0] += 0.5
            wins[1] += 0.5
        else:
            if players[winner] == cpu1:
                wins[0] += 1
            else:
                wins[1] += 1

        logger.debug(f'ROUND {i + 1} | WINNER: {players[winner]} | PLY: {ply}')

    logger.debug(f'{cpu1} SCORE: {wins[0]} | {cpu2} SCORE: {wins[1]}')

compare(TTNegamaxCPU, TTNegamaxCPU, 2, 1)
```

1.17.2 base.py

```python
import time
from pprint import PrettyPrinter
from data.constants 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)

```

```python
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}')

    def find_move(self, board, stop_event=None):
        raise NotImplementedError

    def search(self, board, depth, stop_event, absolute=False, **kwargs):
        if stop_event and stop_event.is_set():
            raise Exception(f'Thread killed - stopping minimax function ({self.
    __str__}.search)')

        self._stats['nodes'] += 1

        if (winner := board.check_win()) is not None:
            self._stats['leaf_nodes'] += 1
            return self.process_win(winner, depth, absolute)

        if depth == 0:
            self._stats['leaf_nodes'] += 1
            return self._evaluator.evaluate(board, absolute), None

    def process_win(self, winner, depth, absolute):
        self._stats['leaf_nodes'] += 1

        if winner == Miscellaneous.DRAW:
            self._stats['draws'] += 1
            return 0, None
        elif winner == Colour.BLUE or absolute:
            self._stats['mates'] += 1
            return Score.CHECKMATE + depth, None
        elif winner == Colour.RED:
            self._stats['mates'] += 1
            return -Score.CHECKMATE - depth, None

    def __str__(self):
        return self.__class__.__name__
```

1.17.3 cpu_thread.py

See Section ??.

1.17.4 evaluator.py

See Section ??.

1.17.5 move_orderer.py

```
from data.states.game.cpu.evaluator import Evaluator
from data.constants import Colour
from data.utils.bitboard_helpers import print_bitboard, pop_count

class SimpleEvaluator:
    def __init__(self):
        self._evaluator = Evaluator(verbose=False)
        self._cache = {}

    def evaluate(self, board):
        if (hashed := board.to_hash()) in self._cache:
            return self._cache[hashed]

        score = self._evaluator.evaluate_material(board, board.get_active_colour()
    )
        self._cache[hashed] = score

        return score

class MoveOrderer:
    def __init__(self):
        self._evaluator = SimpleEvaluator()

    # def get_eval(self, board, move):
    #     laser_result = board.apply_move(move)
    #     score = self._evaluator.evaluate(board)
    #     board.undo_move(move, laser_result)
    #     return score

    # def score_moves(self, board, moves):
    #     for i in range(len(moves)):
    #         score = self.get_eval(board, moves[i])
```

```python
    #         moves[i] = (moves[i], score)

    #     return moves

    def best_move_to_front(self, moves, start_idx, hint):
        for i in range(start_idx + 1, len(moves)):
            if moves[i].src in hint:
                moves[i], moves[start_idx] = moves[start_idx], moves[i]
                return

    def get_moves(self, board, hint=None):
        colour = board.get_active_colour()
        moves = list(board.generate_all_moves(colour))

        for i in range(len(moves)):
            if hint:
                self.best_move_to_front(moves, i, hint)

            yield moves[i]
```

1.17.6 temp.py

```python
from data.constants import Score, Colour
from data.states.game.cpu.transposition_table import TranspositionTable
from data.states.game.cpu.base import BaseCPU
from pprint import pprint

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

    def find_move(self, board, stop_event):
        # No bit_length bug as None type returned, so Move __str__ called on NoneType I think (just deal with None being returned)
        try:
            best_move = self.search(board, self._max_depth, -Score.INFINITE, Score.INFINITE, stop_event)

            if self._verbose:
                print('\nCPU Search Results:')
                pprint(self._stats)
                print('Best move:', best_move, '\n')

                self._callback(self._best_move)
        except Exception as error:
            print('(MinimaxBase.find_move) Error has occured:')
            raise error

    def search(self, board, depth, alpha, beta, stop_event):
        if stop_event.is_set():
            raise Exception('Thread killed - stopping minimax function (CPU.minimax)')

        # cached_move, cached_score = self._transposition_table.get_entry(hash_key=board.bitboards.get_hash(), depth=depth, alpha=alpha, beta=beta)
        # if cached_move or cached_score:
        #     if depth == self._max_depth:
        #         self._best_move = cached_move
        #     return cached_score


    if depth == 0:
        return self.evaluate(board)

    is_maximiser = board.get_active_colour() == Colour.BLUE

    if is_maximiser:
        score = -Score.INFINITE

        for move in board.generate_all_moves(board.get_active_colour()):
            before, before_score = board.bitboards.get_rotation_string(), self
.evaluate(board)

            laser_result = board.apply_move(move)
            new_score = self.minimax(board, depth - 1, alpha, beta, False,
stop_event)

            if new_score >= score:
                score = new_score

                if depth == self._max_depth:
                    self._best_move = move

            board.undo_move(move, laser_result)

            alpha = max(alpha, score)
            if depth == self._max_depth: # https://stackoverflow.com/questions
/31429974/alphabeta-pruning-alpha-equals-or-greater-than-beta-why-equals
                if beta < alpha:
                    break
            else:
                if beta <= alpha:
                    break

            after, after_score = board.bitboards.get_rotation_string(), self.
evaluate(board)
            if (before != after or before_score != after_score):
                print('shit\n\n')

        return score

    else:
        score = Score.INFINITE

        for move in board.generate_all_moves(board.get_active_colour()):
            bef, before_score = board.bitboards.get_rotation_string(), self.
evaluate(board)

            laser_result = board.apply_move(move)
            new_score = self.minimax(board, depth - 1, alpha, beta, False,
stop_event)

            if new_score <= score:
                score = new_score
                if depth == self._max_depth:
                    self._best_move = move

            board.undo_move(move, laser_result)

            beta = min(beta, score)
            if depth == self._max_depth:
                if beta < alpha:
                    break
```

```python
                else:
                    if beta <= alpha:
                        break

                after, after_score = board.bitboards.get_rotation_string(), self.evaluate(board)
                if (bef != after or before_score != after_score):
                    print('shit\n\n')
                    raise ValueError

            return score
```

1.17.7 transposition_table.py

See Section ??.

1.17.8 zobrist_hasher.py

See Section ??.

1.18 data\states\game\cpu\engines

1.18.1 alpha_beta.py

See Section ??.

1.18.2 iterative_deepening.py

```python
from data.states.game.cpu.engines.transposition_table import TranspositionTableMixin
from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU, ABNegamaxCPU
from data.constants import Score

class IterativeDeepeningMixin:
    def find_move(self, board, stop_event):
        best_move = None

        for depth in range(1, self._max_depth + 1):
            self.initialise_stats()
            self._stats['ID_depth'] = depth

            best_score, best_move = self.search(board, depth, -Score.INFINITE, Score.INFINITE, stop_event)

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

        self._callback(best_move)

class IDMinimaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABMinimaxCPU):
    def initialise_stats(self):
        super().initialise_stats()
        self._stats['cache_hits'] = 0

    def print_stats(self, score, move):
        self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] / self._stats['nodes'], 3)
        self._stats['cache_entries'] = len(self._table._table)
        super().print_stats(score, move)

class IDNegamaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABNegamaxCPU)
    :
    def initialise_stats(self):
        super().initialise_stats()
        self._stats['cache_hits'] = 0

    def print_stats(self, score, move):
        self._stats['cache_hits_percentage'] = self._stats['cache_hits'] / self.
    _stats['nodes']
        self._stats['cache_entries'] = len(self._table._table)
        super().print_stats(score, move)
```

1.18.3 minimax.py

See Section ??.

1.18.4 negamax.py

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

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
```

1.18.5 simple.py

```
from data.states.game.cpu.base import BaseCPU
from data.constants import Colour, Score

class SimpleCPU(BaseCPU):
    def __init__(self, callback, verbose=True):
        super().__init__(callback, verbose)

    def find_move(self, board, stop_event=None):
        self.initialise_stats()
        best_score, best_move = self.search(board, stop_event)

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

        self._callback(best_move)

    def search(self, board, stop_event):
        if stop_event and stop_event.is_set():
            raise Exception('Thread killed - stopping simple function (SimpleCPU.search)')

        active_colour = board.bitboards.active_colour
        best_score = -Score.INFINITE if active_colour == Colour.BLUE else Score.INFINITE
        best_move = None

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

            self._stats['nodes'] += 1

            if winner := board.check_win() is not None:
                self.process_win(winner)
            else:
                self._stats['leaf_nodes'] += 1

            score = self._evaluator.evaluate(board)

            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
```

1.18.6 transposition_table.py

See Section ??.

1.18.7 __init__.py

```
from data.states.game.cpu.engines.simple import SimpleCPU
from data.states.game.cpu.engines.negamax import NegamaxCPU
```

```python
from data.states.game.cpu.engines.minimax import MinimaxCPU
from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU, ABNegamaxCPU
from data.states.game.cpu.engines.iterative_deepening import IDMinimaxCPU,
    IDNegamaxCPU
from data.states.game.cpu.engines.transposition_table import TTMinimaxCPU,
    TTNegamaxCPU
```

1.19 data\states\game\mvc

1.19.1 game_controller.py

See Section ??.

1.19.2 game_model.py

See Section ??.

1.19.3 game_view.py

See Section ??.

1.19.4 pause_view.py

```python
import pygame
from data.states.game.widget_dict import PAUSE_WIDGETS
from data.constants import GameEventType, PAUSE_COLOUR
from data.components.widget_group import WidgetGroup
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)
```

1.19.5 win_view.py

```python
from data.constants 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)
```

1.20 data\states\menu

1.20.1 menu.py

```
import pygame
import sys
from random import randint
from data.utils.asset_helpers import get_rotational_angle
from data.states.menu.widget_dict import MENU_WIDGETS
from data.constants import MenuEventType, ShaderType
from data.utils.asset_helpers import scale_and_cache
from data.managers.logs import initialise_logger
from data.managers.animation import animation
from data.assets import GRAPHICS, MUSIC, SFX
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()

        window.clear_apply_arguments(ShaderType.BLOOM)
        window.clear_apply_arguments(ShaderType.SHAKE)
        window.clear_effect(ShaderType.CHROMATIC_ABBREVIATION)

        return None

    def startup(self, persist=None):
        super().startup(MENU_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
        window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
    BACKGROUND_BALATRO)
        window.set_effect(ShaderType.CHROMATIC_ABBREVIATION)

        MENU_WIDGETS['credits'].kill()

        self._fire_laser = False
        self._bloom_mask = None
        self._laser_mask = None

        self.draw()
```

```python
        self.update_masks()

    @property
    def sphinx_center(self):
        return (window.size[0] - self.sphinx_size[0] / 2, window.size[1] - self.
    sphinx_size[1] / 2)

    @property
    def sphinx_size(self):
        return (min(window.size) * 0.1, min(window.size) * 0.1)

    @property
    def sphinx_rotation(self):
        mouse_pos = (pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1] + 0.01)
        return -get_rotational_angle(mouse_pos, self.sphinx_center)

    def get_event(self, event):
        if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
            MENU_WIDGETS['credits'].kill()

        if event.type == pygame.MOUSEBUTTONDOWN:
            self._fire_laser = True
            audio.play_sfx(SFX['menu_laser_windup'])
            audio.play_sfx(SFX['menu_laser_loop'], loop=True)
            animation.set_timer(SFX['menu_laser_loop'].get_length() * 1000 / 2,
    lambda: audio.play_sfx(SFX['menu_laser_loop'], loop=True) if self._fire_laser
    else ...) # OVERLAP TWO LOOPS TO HIDE TRANSITION

        elif event.type == pygame.MOUSEBUTTONUP:
            self._fire_laser = False

            window.clear_effect(ShaderType.RAYS)
            animation.set_timer(300, lambda: window.clear_effect(ShaderType.SHAKE)
    )
            audio.stop_sfx(1000)

        widget_event = self._widget_group.process_event(event)

        if widget_event is None:
            return

        match widget_event.type:
            case None:
                return

            case MenuEventType.CONFIG_CLICK:
                self.next = 'config'
                self.done = True
            case MenuEventType.SETTINGS_CLICK:
                self.next = 'settings'
                self.done = True
            case MenuEventType.BROWSER_CLICK:
                self.next = 'browser'
                self.done = True
            case MenuEventType.QUIT_CLICK:
                pygame.quit()
                sys.exit()
                logger.info('quitting...')
            case MenuEventType.CREDITS_CLICK:
                self._widget_group.add(MENU_WIDGETS['credits'])

    def draw_sphinx(self):
        sphinx_surface = scale_and_cache(GRAPHICS['sphinx_0_b'], self.sphinx_size)
        sphinx_surface = pygame.transform.rotate(sphinx_surface, self.
    sphinx_rotation)
        sphinx_rect = pygame.FRect(0, 0, *self.sphinx_size)
        sphinx_rect.center = self.sphinx_center

        window.screen.blit(sphinx_surface, sphinx_rect)

    def update_masks(self):
        self.draw()

        widget_mask = window.screen.copy()
        laser_mask = pygame.mask.from_surface(widget_mask)
        laser_mask = laser_mask.to_surface(setcolor=(255, 0, 0, 255), unsetcolor
    =(0, 0, 0, 255))
        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)
    )
        pygame.draw.rect(widget_mask, (0, 0, 0, 255), (window.screen.width - 50,
    0, 50, 50))

        self._bloom_mask = widget_mask
        self._laser_mask = laser_mask

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

        if self._fire_laser:
            window.set_apply_arguments(ShaderType.RAYS, occlusion=self._laser_mask
    , softShadow=0.1)

        window.set_apply_arguments(ShaderType.BLOOM, highlight_surface=self.
    _bloom_mask, surface_intensity=0.3, brightness_intensity=0.6)

    def update(self, **kwargs):
        random_offset = lambda: randint(-5, 5) / 40
        if self._fire_laser:
            window.clear_effect(ShaderType.RAYS)
            window.set_effect(ShaderType.RAYS, lights=[[
                (self.sphinx_center[0] / window.size[0], self.sphinx_center[1] /
    window.size[1]),
                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()
```

1.20.2 widget_dict.py

```python
from data.components.custom_event import CustomEvent
from data.constants import MenuEventType
from data.managers.theme import theme
from data.assets import GRAPHICS
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,
        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
        )
    ]
}
```

1.21 data\states\review

1.21.1 review.py

See Section ??.

1.21.2 widget_dict.py

```
from data.widgets import *
from data.components.custom_event import CustomEvent
from data.constants import ReviewEventType, Colour
from data.assets import GRAPHICS

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),
        relative_size=(1, 0.3),
        anchor_y='bottom',
        text='MOVE NO:',
        fit_vertical=False,
        margin=10,
        border_width=0,
        fill_colour=(0, 0, 0, 0),
    ),
    'move_colour_text':
    Text(
        parent=info_container,
        relative_size=(1, 0.3),
        relative_position=(0, 0),
        anchor_y='center',
        text='TO MOVE',
        fit_vertical=False,
        margin=10,
        border_width=0,
        fill_colour=(0, 0, 0, 0),
    ),
    'winner_text':
    Text(
        parent=info_container,
        relative_size=(1, 0.3),
        relative_position=(0, 0),
        text='WINNER:',
        fit_vertical=False,
        margin=10,
        border_width=0,
        fill_colour=(0, 0, 0, 0),
    ),
    'blue_timer':
    Timer(
        relative_position=(0.05, 0.05),
        anchor_y='center',
        relative_size=(0.1, 0.1),
        active_colour=Colour.BLUE,
    ),
    'red_timer':
    Timer(
        relative_position=(0.05, -0.05),
        anchor_y='center',
        relative_size=(0.1, 0.1),
        active_colour=Colour.RED
    ),
    'timer_disabled_text':
    Text(
        relative_size=(0.2, 0.1),
        relative_position=(0.05, 0),
        anchor_y='center',
        fit_vertical=False,
        text='TIMER DISABLED',
    ),
    'blue_piece_display':
    PieceDisplay(
        relative_position=(0.05, 0.05),
        relative_size=(0.2, 0.1),
        anchor_y='bottom',
        active_colour=Colour.BLUE
    ),
    'red_piece_display':
    PieceDisplay(
        relative_position=(0.05, 0.05),
        relative_size=(0.2, 0.1),
        active_colour=Colour.RED
    ),
}
```

1.22 data\states\settings

1.22.1 settings.py

```
import pygame
```

```python
from random import randint
from data.utils.data_helpers import get_default_settings, get_user_settings, update_user_settings
from data.constants import SettingsEventType, WidgetState, ShaderType, SHADER_MAP
from data.states.settings.widget_dict import SETTINGS_WIDGETS
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.control import _State
from data.assets import MUSIC

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]),
            relative_width=0.15,
            selected_colour=selected_colour,
            event_type=event_type
        )
        self._widget_group.add(self._colour_picker)

    def remove_colour_picker(self):
        self._colour_picker.kill()

    def reload_display_mode(self):
        relative_mouse_pos = (pygame.mouse.get_pos()[0] / window.size[0], pygame.mouse.get_pos()[1] / window.size[1])

        if self._settings['displayMode'] == 'fullscreen':
            window.set_fullscreen(desktop=True)
            window.handle_resize()

        elif self._settings['displayMode'] == 'windowed':
            window.set_windowed()
            window.handle_resize()
            window.restore()

        self._widget_group.handle_resize(window.size)

        new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
    relative_mouse_pos[1] * window.size[1])
        pygame.mouse.set_pos(new_mouse_pos)

    def reload_shaders(self):
        window.clear_all_effects()

        for shader_type in SHADER_MAP[self._settings['shader']]:
            window.set_effect(shader_type)

    def reload_settings(self):
        SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self.
    _settings['primaryBoardColour'])
        SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self.
    _settings['secondaryBoardColour'])
        SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.
    BASE)
        SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.
    BASE)
        SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['
    musicVolume'])
        SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume
    '])
        SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings
    ['displayMode'])
        SETTINGS_WIDGETS['shader_carousel'].set_to_key(self._settings['shader'])
        SETTINGS_WIDGETS['particles_switch'].set_toggle_state(self._settings['
    particles'])
        SETTINGS_WIDGETS['opengl_switch'].set_toggle_state(self._settings['opengl'
    ])

        self.reload_shaders()
        self.reload_display_mode()

    def get_event(self, event):
        widget_event = self._widget_group.process_event(event)

        if widget_event is None:
            if event.type == pygame.MOUSEBUTTONDOWN and self._colour_picker:
                self.remove_colour_picker()
            return

        match widget_event.type:
            case SettingsEventType.VOLUME_SLIDER_SLIDE:
                return

            case SettingsEventType.VOLUME_SLIDER_CLICK:
                if widget_event.volume_type == 'music':
                    audio.set_music_volume(widget_event.volume)
                    self._settings['musicVolume'] = widget_event.volume
                elif widget_event.volume_type == 'sfx':
```

```python
                    audio.set_sfx_volume(widget_event.volume)
                    self._settings['sfxVolume'] = widget_event.volume

            case SettingsEventType.DROPDOWN_CLICK:
                selected_word = SETTINGS_WIDGETS['display_mode_dropdown'].get_selected_word()

                if selected_word is None or selected_word == self._settings['displayMode']:
                    return

                self._settings['displayMode'] = selected_word

                self.reload_display_mode()

            case SettingsEventType.MENU_CLICK:
                self.next = 'menu'
                self.done = True

            case SettingsEventType.RESET_DEFAULT:
                self._settings = get_default_settings()
                self.reload_settings()

            case SettingsEventType.RESET_USER:
                self._settings = get_user_settings()
                self.reload_settings()

            case SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK | SettingsEventType.SECONDARY_COLOUR_BUTTON_CLICK:
                if self._colour_picker:
                    self.remove_colour_picker()

                self.create_colour_picker(event.pos, widget_event.type)

            case SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK | SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK:
                if widget_event.colour:
                    r, g, b = widget_event.colour.rgb
                    hex_colour = f'0x{hex(r)[2:].zfill(2)}{hex(g)[2:].zfill(2)}{hex(b)[2:].zfill(2)}'

                    if widget_event.type == SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK:
                        SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(widget_event.colour)
                        SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.BASE)
                        self._settings['primaryBoardColour'] = hex_colour
                    elif widget_event.type == SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK:
                        SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(widget_event.colour)
                        SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.BASE)
                        self._settings['secondaryBoardColour'] = hex_colour

            case SettingsEventType.SHADER_PICKER_CLICK:
                self._settings['shader'] = widget_event.data
                self.reload_shaders()

            case SettingsEventType.OPENGL_CLICK:
                self._settings['opengl'] = widget_event.toggled
                self.reload_shaders()

            case SettingsEventType.PARTICLES_CLICK:
                self._settings['particles'] = widget_event.toggled

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

1.22.2 widget_dict.py

```
from data.widgets import *
from data.components.custom_event import CustomEvent
from data.constants import SettingsEventType, SHADER_MAP
from data.utils.data_helpers import get_user_settings
from data.assets import GRAPHICS, DEFAULT_FONT
from data.managers.theme import theme
from data.utils.font_helpers import text_width_to_font_size
from data.managers.window import window

user_settings = get_user_settings()
# font_size = text_width_to_font_size('Shaders (OPENGL GPU REQUIRED)',
    DEFAULT_FONT, 0.4 * window.screen.width)
FONT_SIZE = 21

carousel_widgets = {
    key: Text(
        relative_position=(0, 0),
        relative_size=(0.25, 0.04),
        margin=0,
        text=key.replace('_', ' ').upper(),
        fit_vertical=True,
        border_width=0,
        fill_colour=(0, 0, 0, 0),
    ) for key in SHADER_MAP.keys()
}

reset_container = Rectangle(
    relative_size=(0.2, 0.2),
    relative_position=(0, 0),
    fixed_position=(5, 5),
    anchor_x='right',
    anchor_y='bottom',
)

SETTINGS_WIDGETS = {
    'default': [
        reset_container,
        ReactiveIconButton(
            relative_position=(0, 0),
            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)
    ),
}
```

1.23 data\utils

1.23.1 asset_helpers.py

See Section ??.

1.23.2 bitboard_helpers.py

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

logger = initialise_logger(__name__)

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)

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

def bitboard_to_coords_list(bitboard):
    list_positions = []

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

    return list_positions

def occupied_squares(bitboard):
    while bitboard:
        lsb_square = bitboard & -bitboard
        bitboard = bitboard ^ lsb_square

        yield lsb_square

def pop_count(bitboard):
    count = 0
```

```python
    while bitboard:
        count += 1
        lsb_square = bitboard & -bitboard
        bitboard = bitboard ^ lsb_square

    return count

# def pop_count(bitboard):
#     count = 0
#     while bitboard:
#         count += 1
#         bitboard &= bitboard - 1

#     return count

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

#Solar
def get_LSB_value(bitboard: int):
    return bitboard & -bitboard

def pop_count_2(bitboard):
    count = 0
    while bitboard > 0:
        lsb_value = get_LSB_value(bitboard)
        count += 1
        bitboard ^= lsb_value

    return count
```

1.23.3 board_helpers.py

```python
import pygame
from data.utils.data_helpers import get_user_settings
from data.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
        else:
            square_colour = secondary_colour

        square_x = x * square_size
        square_y = y * square_size

        pygame.draw.rect(board_surface, square_colour, (square_x, square_y,
    square_size + 1, square_size + 1)) # +1 to fill in black lines

        if y == 7:
            text_position = (square_x + square_size * 0.7, square_y + square_size
    * 0.55)
```

```
            text_size = square_size / 3
            font.render_to(board_surface, text_position, str(chr(x + 1 + 96)), fgcolor=(10, 10, 10, 175), size=text_size)
        if x == 0:
            text_position = (square_x + square_size * 0.1, square_y + square_size * 0.1)
            text_size = square_size / 3
            font.render_to(board_surface, text_position, str(7-y + 1), fgcolor=(10, 10, 10, 175), size=text_size)

    return board_surface

def create_square_overlay(square_size, colour):
    overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
    overlay.fill(colour)

    return overlay

def create_circle_overlay(square_size, colour):
    overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
    pygame.draw.circle(overlay, colour, (square_size / 2, square_size / 2), square_size / 4)

    return overlay

def coords_to_screen_pos(coords, board_position, square_size):
    x = board_position[0] + (coords[0] * square_size)
    y = board_position[1] + ((7 - coords[1]) * square_size)

    return (x, y)

def screen_pos_to_coords(mouse_position, board_position, board_size):
    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]):
        x = (mouse_position[0] - board_position[0]) // (board_size[0] / 10)
        y = (board_size[1] - (mouse_position[1] - board_position[1])) // (board_size[0] / 10)
        return (int(x), int(y))

    return None
```

1.23.4 browser_helpers.py

```
from data.constants import Miscellaneous, Colour

def get_winner_string(winner):
    if winner is None:
        return 'UNFINISHED'
    elif winner == Miscellaneous.DRAW:
        return 'DRAW'
    else:
        return Colour(winner).name
```

1.23.5 database_helpers.py

See Section ??.

1.23.6 data_helpers.py

See Section ??.

1.23.7 font_helpers.py

```python
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
```

1.23.8 input_helpers.py

```python
from data.constants 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())
```

1.23.9 load_helpers.py

```python
import pygame
from pathlib import Path

import pygame.freetype
from data.utils.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)

    #     return image

def load_gfx(path, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
    file_path = Path(path)
    name, extension = file_path.stem, file_path.suffix

    if extension.lower() in accept:
        if extension.lower() == '.gif':
            frames_list = []

            for frame in gif_to_frames(path):
                image_surface = pil_image_to_surface(frame)
                frames_list.append(image_surface)

            return frames_list

        if extension.lower() == '.svg':
            low_quality_image = pygame.image.load_sized_svg(path, (200, 200))
            image = pygame.image.load(path)
            image = convert_gfx_alpha(image, colorkey)

            return [image, low_quality_image]

        else:
            image = pygame.image.load(path)
            return convert_gfx_alpha(image, colorkey)

def load_all_gfx(directory, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
    graphics = {}

    for file in Path(directory).rglob('*'):
        name, extension = file.stem, file.suffix
        path = Path(directory / file)

        if extension.lower() in accept and 'old' not in name:
            if name == 'piece_spritesheet':
                data = load_spritesheet(
                    path,
                    (16, 16),
                    ['pyramid_1', 'scarab_1', 'anubis_1', 'pharoah_1', 'sphinx_1', 'pyramid_0', 'scarab_0', 'anubis_0', 'pharoah_0', 'sphinx_0'],
                    ['_a', '_b', '_c', '_d'])

                graphics = graphics | data
                continue

            data = load_gfx(path, colorkey, accept)

            if isinstance(data, list):
                graphics[name] = data[0]
                graphics[f'{name}_lq'] = data[1]
            else:
                graphics[name] = data

    return graphics

def load_spritesheet(path, sprite_size, col_names, row_names):
    spritesheet = load_gfx(path)
    col_count = int(spritesheet.width / sprite_size[0])
    row_count = int(spritesheet.height / sprite_size[1])

    sprite_dict = {}

    for column in range(col_count):
        for row in range(row_count):
            surface = pygame.Surface(sprite_size, pygame.SRCALPHA)
            name = col_names[column] + row_names[row]

            surface.blit(spritesheet, (0, 0), (column * sprite_size[0], row * sprite_size[1], *sprite_size))
            sprite_dict[name] = surface

    return sprite_dict
```

```python

def load_all_fonts(directory, accept=(".ttf", ".otf")):
    fonts = {}

    for file in Path(directory).rglob('*'):
        name, extension = file.stem, file.suffix
        path = Path(directory / file)

        if extension.lower() in accept:
            font = pygame.freetype.Font(path)
            fonts[name] = font

    return fonts

def load_all_sfx(directory, accept=(".mp3", ".wav", ".ogg")):
    sound_effects = {}

    for file in Path(directory).rglob('*'):
        name, extension = file.stem, file.suffix
        path = Path(directory / file)

        if extension.lower() in accept and 'old' not in name:
            sound_effects[name] = load_sfx(path)

    return sound_effects

def load_sfx(path, accept=(".mp3", ".wav", ".ogg")):
    file_path = Path(path)
    name, extension = file_path.stem, file_path.suffix

    if extension.lower() in accept:
        sfx = pygame.mixer.Sound(path)
        return sfx

def load_all_music(directory, accept=(".mp3", ".wav", ".ogg")):
    music_paths = {}
    for file in Path(directory).rglob('*'):
        name, extension = file.stem, file.suffix
        path = Path(directory / file)

        if extension.lower() in accept:
            music_paths[name] = path

    return music_paths
```

1.23.10 widget_helpers.py

See Section ??.

1.24 data\widgets

1.24.1 board_thumbnail.py

```python
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()

    def initialise_board(self, fen_string):
        if len(fen_string) == 0:
            piece_list = []
        else:
            piece_list = BitboardCollection(fen_string).convert_to_piece_list()

        self._piece_group = PieceGroup()
        self._piece_group.initialise_pieces(piece_list, (0, 0), self.size)

        self._board.refresh_board()
        self.set_image()

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

        self._board.set_image()
        self.image.blit(self._board.image, (0, 0))

        self._piece_group.draw(self.image)

    def set_geometry(self):
        super().set_geometry()
        self._board.set_geometry()

    def set_surface_size(self, new_surface_size):
        super().set_surface_size(new_surface_size)
        self._board.set_surface_size(new_surface_size)
        self._piece_group.handle_resize((0, 0), self.size)

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

1.24.2 board_thumbnail_button.py

```
import pygame
from data.widgets.bases.pressable import _Pressable
from data.widgets.board_thumbnail import BoardThumbnail
from data.constants import WidgetState
from data.components.custom_event import CustomEvent

class BoardThumbnailButton(_Pressable, BoardThumbnail):
    def __init__(self, event, **kwargs):
        _Pressable.__init__(
            self,
            event=CustomEvent(**vars(event), fen_string=kwargs.get('fen_string')),
            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),
        )
        BoardThumbnail.__init__(self, **kwargs)

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

1.24.3 browser_item.py

```
import pygame
from data.utils.font_helpers import text_width_to_font_size
from data.utils.browser_helpers import get_winner_string
from data.widgets.board_thumbnail import BoardThumbnail
from data.utils.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
```

1.24.4 browser_strip.py

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

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
```

```python
        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()
            browser_list.append((item.image, (index * (self.item_width + self.margin) + self.margin, self.margin)))

        self.image.blits(browser_list)

        if self._selected_index is not None:
            border_position = (self._selected_index * (self.item_width + self.margin), 0)
            border_size = (self.item_width + 2 * self.margin, self.size[1])
            pygame.draw.rect(self.image, (255, 255, 255), (*border_position, *border_size), width=int(self.item_width / 20))

    def set_geometry(self):
        super().set_geometry()
        for item in self._items_list:
            item.set_geometry()

    def set_surface_size(self, new_surface_size):
        super().set_surface_size(new_surface_size)

        for item in self._items_list:
            item.set_surface_size(new_surface_size)

    def process_event(self, event, scrolled_pos):
        parent_pos = self._get_rect().topleft
        self.rect.topleft = parent_pos

        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
            self._selected_index = None
            self.set_image()
            return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK, selected_index=None)

        if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.pos):
            relative_mouse_pos = (event.pos[0] - parent_pos[0], event.pos[1] - parent_pos[1])
            self._selected_index = int(max(0, (relative_mouse_pos[0] - self.margin) // (self.item_width + self.margin)))
            self.set_image()
            return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK, selected_index=self._selected_index)
```

1.24.5 carousel.py

```python
import pygame
```

```python
from data.widgets.reactive_icon_button import ReactiveIconButton
from data.components.custom_event import CustomEvent
from data.widgets.bases.circular import _Circular
from data.widgets.bases.widget import _Widget
from data.constants import Miscellaneous
from data.assets import GRAPHICS, SFX

class Carousel(_Circular, _Widget):
    def __init__(self, event, widgets_dict, **kwargs):
        _Circular.__init__(self, items_dict=widgets_dict)
        _Widget.__init__(self, relative_size=None, **kwargs)

        max_widget_size = (
            max([widget.rect.width for widget in widgets_dict.values()]),
            max([widget.rect.height for widget in widgets_dict.values()])
        )

        self._relative_max_widget_size = (max_widget_size[0] / self.surface_size[1], max_widget_size[1] / self.surface_size[1])
        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])

        self._left_arrow = ReactiveIconButton(
            relative_position=(0, 0),
            relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
            scale_mode='height',
            base_icon=GRAPHICS['left_arrow_base'],
            hover_icon=GRAPHICS['left_arrow_hover'],
            press_icon=GRAPHICS['left_arrow_press'],
            event=CustomEvent(Miscellaneous.PLACEHOLDER),
            sfx=SFX['carousel_click']
        )
        self._right_arrow = ReactiveIconButton(
            relative_position=(0, 0),
            relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
            scale_mode='height',
            base_icon=GRAPHICS['right_arrow_base'],
            hover_icon=GRAPHICS['right_arrow_hover'],
            press_icon=GRAPHICS['right_arrow_press'],
            event=CustomEvent(Miscellaneous.PLACEHOLDER),
            sfx=SFX['carousel_click']
        )

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

        self.set_image()
        self.set_geometry()

    @property
    def max_widget_size(self):
        return (self._relative_max_widget_size[0] * self.surface_size[1], self._relative_max_widget_size[1] * self.surface_size[1])

    @property
    def arrow_size(self):
        height = self.max_widget_size[1] * 0.75
        width = (GRAPHICS['left_arrow_base'].width / GRAPHICS['left_arrow_base'].height) * height
        return (width, height)

    @property
    def size(self):
        return ((self.arrow_size[0] + self.margin) * 2 + self.max_widget_size[0], self.max_widget_size[1])

    @property
    def left_arrow_position(self):
        return (0, (self.size[1] - self.arrow_size[1]) / 2)

    @property
    def right_arrow_position(self):
        return (self.size[0] - self.arrow_size[0], (self.size[1] - self.arrow_size[1]) / 2)

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

        if self.border_width:
            pygame.draw.rect(self.image, self._border_colour, (0, 0, *self.size), width=int(self.border_width), border_radius=int(self.border_radius))

        self._left_arrow.set_image()
        self.image.blit(self._left_arrow.image, self.left_arrow_position)

        self.current_item.set_image()
        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))

        self._right_arrow.set_image()
        self.image.blit(self._right_arrow.image, self.right_arrow_position)

    def set_geometry(self):
        super().set_geometry()

        self.current_item.set_geometry()
        self._left_arrow.set_geometry()
        self._right_arrow.set_geometry()

        self.current_item.rect.center = self.rect.center
        self._left_arrow.rect.topleft = (self.position[0] + self.left_arrow_position[0], self.position[1] + self.left_arrow_position[1])
        self._right_arrow.rect.topleft = (self.position[0] + self.right_arrow_position[0], self.position[1] + self.right_arrow_position[1])

    def set_surface_size(self, new_surface_size):
        super().set_surface_size(new_surface_size)
        self._left_arrow.set_surface_size(new_surface_size)
        self._right_arrow.set_surface_size(new_surface_size)

        for item in self._items_dict.values():
            item.set_surface_size(new_surface_size)

    def process_event(self, event):
        self.current_item.process_event(event)
        left_arrow_event = self._left_arrow.process_event(event)
        right_arrow_event = self._right_arrow.process_event(event)

        if left_arrow_event:
            self.set_previous_item()
            self.current_item.set_surface_size(self._raw_surface_size)

        elif right_arrow_event:
            self.set_next_item()
            self.current_item.set_surface_size(self._raw_surface_size)

        if left_arrow_event or right_arrow_event:
            self.set_image()
            self.set_geometry()

            return CustomEvent(**vars(self._event), data=self.current_key)

        elif event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.MOUSEMOTION]:
            self.set_image()
            self.set_geometry()
```

1.24.6 chessboard.py

```python
import pygame
from data.widgets.bases.widget import _Widget
from data.utils.board_helpers import create_board
from data.utils.data_helpers import get_user_settings
from data.constants 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)
```

1.24.7 colour_button.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.bases.pressable import _Pressable
from data.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))
```

1.24.8 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
```

1.24.9 colour_picker.py

```python
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)
        self.rect = self.image.get_rect()

        self._square = _ColourSquare(
            parent=self,
            relative_position=(0.1, 0.1),
            relative_width=0.5,
            event_type=event_type
        )
        self._square.set_colour(kwargs.get('selected_colour'))

        self._slider = _ColourSlider(
            parent=self,
            relative_position=(0.0, 0.7),
            relative_width=1.0,
            border_width=self.border_width,
            border_colour=self._border_colour
        )
        self._slider.set_colour(kwargs.get('selected_colour'))

        self._display = _ColourDisplay(
            parent=self,
            relative_position=(0.7, 0.1),
            relative_size=(0.2, 0.5)
        )
        self._display.set_colour(kwargs.get('selected_colour'))

        self._event_type = event_type
        self._hover_event_type = event_type

        self.set_image()
        self.set_geometry()

    def global_to_relative_pos(self, global_pos):
        return (global_pos[0] - self.position[0], global_pos[1] - self.position
    [1])

    def set_image(self):
        self.image = pygame.Surface(self.size)
        self.image.fill(self._fill_colour)

        self._square.set_image()
        self._square.set_geometry()
        self.image.blit(self._square.image, self.global_to_relative_pos(self.
    _square.position))

        self._slider.set_image()
        self._slider.set_geometry()
        self.image.blit(self._slider.image, self.global_to_relative_pos(self._slider.position))

        self._display.set_image()
        self._display.set_geometry()
        self.image.blit(self._display.image, self.global_to_relative_pos(self._display.position))

        pygame.draw.rect(self.image, self._border_colour, (0, 0, self.size[0], self.size[1]), width=int(self.border_width))

    def set_surface_size(self, new_surface_size):
        super().set_surface_size(new_surface_size)
        self._square.set_surface_size(self.size)
        self._slider.set_surface_size(self.size)
        self._display.set_surface_size(self.size)

    def get_picker_position(self):
        return self.position

    def process_event(self, event):
        slider_colour = self._slider.process_event(event)
        square_colour = self._square.process_event(event)

        if square_colour:
            self._display.set_colour(square_colour)
            self.set_image()

        if slider_colour:
            self._square.set_colour(slider_colour)
            self.set_image()

        if event.type in [pygame.MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN, pygame.MOUSEMOTION] and self.rect.collidepoint(event.pos):
            return CustomEvent(self._event_type, colour=square_colour)
```

1.24.10 colour_slider.py

See Section ??.

1.24.11 colour_square.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.utils.widget_helpers import create_square_gradient

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

        self._colour = None

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

    def get_colour(self):
        return self._colour

    def set_image(self):
        self.image = create_square_gradient(side_length=self.size[0], colour=self._colour)

    def process_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
            relative_mouse_pos = (event.pos[0] - self.position[0], event.pos[1] - self.position[1])

            if (
                0 > relative_mouse_pos[0] or
                self.size[0] < relative_mouse_pos[0] or
                0 > relative_mouse_pos[1] or
                self.size[1] < relative_mouse_pos[1]
            ): return None

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

            return self._colour

        return None
```

1.24.12 dropdown.py

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

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)
        return (all_words_rect.size[0] + max_word_rect.size[1], all_words_rect.size[1])

    def get_selected_word(self):
        return self._word_list[0].lower()

    def toggle_expanded(self):
        if self._expanded:
            self._word_list = [self._word_list_copy[0]]
        else:
            self._word_list = [*self._word_list_copy]

        self._expanded = not(self._expanded)

    def hover_func(self):
        mouse_position = pygame.mouse.get_pos()
        relative_position = (mouse_position[0] - self.position[0], mouse_position[1] - self.position[1])
        self._hovered_index = self.calculate_hovered_index(relative_position)
        self.set_state_colour(WidgetState.HOVER)

    def set_selected_word(self, word):
        index = self._word_list_copy.index(word.capitalize())
        selected_word = self._word_list_copy.pop(index)
        self._word_list_copy.insert(0, selected_word)

        if self._expanded:
            self._word_list.pop(index)
            self._word_list.insert(0, selected_word)
        else:
            self._word_list = [selected_word]

        self.set_image()

    def up_func(self):
        if self.get_widget_state() == WidgetState.PRESS:
            if self._expanded and self._hovered_index is not None:
                self.set_selected_word(self._word_list_copy[self._hovered_index])

            self.toggle_expanded()

        self._hovered_index = None

        self.set_state_colour(WidgetState.BASE)
        self.set_geometry()

```

```python
    def calculate_hovered_index(self, mouse_pos):
        return int(mouse_pos[1] // (self.size[1] / len(self._word_list)))

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

        fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
        pygame.draw.rect(self.image, self._background_colour, fill_rect)
        pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int(self.border_width))

        word_box_height = (self.size[1] - (2 * self.margin) - ((len(self._word_list) - 1) * self.margin)) / len(self._word_list)

        arrow_size = (GRAPHICS['dropdown_arrow_open'].width / GRAPHICS['dropdown_arrow_open'].height * word_box_height, word_box_height)
        open_arrow_surface = pygame.transform.scale(GRAPHICS['dropdown_arrow_open'], arrow_size)
        closed_arrow_surface = pygame.transform.scale(GRAPHICS['dropdown_arrow_close'], arrow_size)
        arrow_position = (self.size[0] - arrow_size[0] - self.margin, (word_box_height) / 3)

        if self._expanded:
            self.image.blit(closed_arrow_surface, arrow_position)
        else:
            self.image.blit(open_arrow_surface, arrow_position)

        for index, word in enumerate(self._word_list):
            word_position = (self.margin, self.margin + (word_box_height + self.margin) * index)
            self._font.render_to(self.image, word_position, word, fgcolor=self._text_colour, size=self.font_size)

        if self._hovered_index is not None:
            overlay_surface = pygame.Surface((self.size[0], word_box_height + 2 * self.margin), pygame.SRCALPHA)
            overlay_surface.fill((*self._fill_colour.rgb, 128))
            overlay_position = (0, (word_box_height + self.margin) * self._hovered_index)
            self.image.blit(overlay_surface, overlay_position)
```

1.24.13 icon.py

```python
import pygame
from data.widgets.bases.widget import _Widget
from data.utils.widget_helpers import create_text_box

class Icon(_Widget):
    def __init__(self, icon, stretch=False, is_mask=False, smooth=False, fit_icon=False, box_colours=None, **kwargs):
        super().__init__(**kwargs)

        if fit_icon:
            aspect_ratio = icon.width / icon.height
            self._relative_size = (self._relative_size[1] * aspect_ratio, self._relative_size[1])

        self._icon = icon
        self._is_mask = is_mask
        self._stretch = stretch
        self._smooth = smooth
        self._box_colours = box_colours

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

        self.set_image()
        self.set_geometry()

    def set_icon(self, icon):
        self._icon = icon
        self.set_image()

    def set_image(self):
        if self._box_colours:
            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
```

1.24.14 icon_button.py

```python
from data.widgets.bases.pressable import _Pressable
from data.widgets.bases.box import _Box
from data.widgets.icon import Icon
from data.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)

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

1.24.15 move_list.py

```python
import pygame
from data.widgets.bases.widget import _Widget
from data.utils.font_helpers import width_to_font_size

class MoveList(_Widget):
    def __init__(self, relative_width, minimum_height=0, move_list=[], **kwargs):
        super().__init__(relative_size=None, **kwargs)

        self._relative_width = relative_width * self.surface_size[0] / self.
    surface_size[1]
        self._relative_minimum_height = minimum_height / self.surface_size[1]
        self._move_list = move_list
        self._relative_font_size = width_to_font_size(self._font, self.
    surface_size[0] / 3.5) / self.surface_size[1]

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

        self.set_image()
        self.set_geometry()

    @property
    def size(self):
        font_metrics = self._font.get_metrics('j', size=self.font_size)

        width = self._relative_width * self.surface_size[1]
        minimum_height = self._relative_minimum_height * self.surface_size[1]
        row_gap = font_metrics[0][3] - font_metrics[0][2]
        number_of_rows = 2 * ((len(self._move_list) + 1) // 2) + 1

        return (width, max(minimum_height, row_gap * number_of_rows))

    def register_get_rect(self, get_rect_func):
        pass

    def reset_move_list(self):
        self._move_list = []
        self.set_image()
        self.set_geometry()

    def append_to_move_list(self, new_move):
        self._move_list.append(new_move)
        self.set_image()
        self.set_geometry()

    def pop_from_move_list(self):
        self._move_list.pop()
        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)

        font_metrics = self._font.get_metrics('j', size=self.font_size)
        row_gap = font_metrics[0][3] - font_metrics[0][2]

        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
```

1.24.16 multiple_icon_button.py

```
import pygame
from data.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)

  def set_next_icon(self):
    super().set_next_item()
    self._icon = self.current_item
    self.set_image()

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

    if widget_event:
      return CustomEvent(**vars(widget_event), data=self.current_key)
```

1.24.17 piece_display.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.states.game.components.piece_sprite import PieceSprite
from data.constants import Score, Rotation, WidgetState, Colour,
    BLUE_BUTTON_COLOURS, RED_BUTTON_COLOURS
from data.utils.widget_helpers import create_text_box
from data.utils.asset_helpers import scale_and_cache

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

        self._active_colour = active_colour
        self._piece_list = []
        self._piece_surface = None
        self._box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour
     == Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]

        self.initialise_piece_surface()

        self.set_image()
        self.set_geometry()

    def add_piece(self, piece):
        self._piece_list.append(piece)
        self._piece_list.sort(key=lambda piece: Score[piece.name])
        self.initialise_piece_surface()

    def remove_piece(self, piece):
        self._piece_list.remove(piece)
        self.initialise_piece_surface()

    def reset_piece_list(self):
        self._piece_list = []
        self.initialise_piece_surface()

    def initialise_piece_surface(self):
        self._piece_surface = pygame.Surface((self.size[0] - 2 * self.margin, self
.size[1] - 2 * self.margin), pygame.SRCALPHA)

        if (len(self._piece_list) == 0):
            self.set_image()
            return

        piece_width = min(self.size[1] - 2 * self.margin, (self.size[0] - 2 * self
.margin) / len(self._piece_list))
        piece_list = []

        for index, piece in enumerate(self._piece_list):
            piece_instance = PieceSprite(piece, self._active_colour.
get_flipped_colour(), Rotation.UP)
            piece_instance.set_geometry((0, 0), piece_width)
            piece_instance.set_image()
            piece_list.append((piece_instance.image, (piece_width * index, (self.
_piece_surface.height - piece_width) / 2)))

        self._piece_surface.fblits(piece_list)

        self.set_image()

    def set_image(self):
        self.image = create_text_box(self.size, self.border_width, self.
_box_colours)

        resized_piece_surface = scale_and_cache(self._piece_surface, (self.size[0]
 - 2 * self.margin, self.size[1] - 2 * self.margin))
        self.image.blit(resized_piece_surface, (self.margin, self.margin))

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

1.24.18 reactive_button.py

See Section ??.

1.24.19 reactive_icon_button.py

See Section ??.

1.24.20 rectangle.py

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

class Rectangle(_Widget):
    def __init__(self, visible=False, **kwargs):
        super().__init__(**kwargs)

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

        self.set_image()
        self.set_geometry()

    def set_image(self):
        self.image = pygame.transform.scale(self._empty_surface, self.size)
        if self._visible:
            pygame.draw.rect(self.image, self._fill_colour, self.image.get_rect(),
 border_radius=int(self.border_radius))

            if 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
```

1.24.21 scrollbar.py

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

# self.set_state_colour(WidgetState.HOVER)
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))

    def process_event(self, event):
        before_state = self.get_widget_state()
        widget_event = super().process_event(event)
        after_state = self.get_widget_state()

        if event.type == pygame.MOUSEMOTION and self._last_mouse_px:
            if self._vertical:
                offset_from_last_frame = event.pos[1] - self._last_mouse_px
                self._last_mouse_px = event.pos[1]

                return offset_from_last_frame
            else:
                offset_from_last_frame = event.pos[0] - self._last_mouse_px
                self._last_mouse_px = event.pos[0]

                return offset_from_last_frame


        if widget_event or before_state != after_state:
            return 0
```

1.24.22 scroll_area.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.scrollbar import _Scrollbar
from data.managers.theme import theme

SCROLLBAR_WIDTH_FACTOR = 0.05

class ScrollArea(_Widget):
    def __init__(self, widget, vertical, scroll_factor=15, **kwargs):
        super().__init__(**kwargs)
        if vertical is False:
            self._relative_size = kwargs.get('relative_size')

        self._relative_scroll_factor = scroll_factor / self.surface_size[1]

        self._scroll_percentage = 0
        self._widget = widget
        self._vertical = vertical

        self._widget.register_get_rect(self.calculate_widget_rect)

        if self._vertical:
            anchor_x = 'right'
```

```python
            anchor_y = 'top'
            scale_mode = 'height'
        else:
            anchor_x = 'left'
            anchor_y = 'bottom'
            scale_mode = 'width'

        self._scrollbar = _Scrollbar(
            parent=self,
            relative_position=(0, 0),
            relative_size=None,
            anchor_x=anchor_x,
            anchor_y=anchor_y,
            fill_colour=theme['borderPrimary'],
            scale_mode=scale_mode,
            vertical=vertical,
        )

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

        self.set_image()
        self.set_geometry()

    @property
    def scroll_factor(self):
        return self._relative_scroll_factor * self.surface_size[1]

    @property
    def scrollbar_size(self):
        if self._vertical:
            return (self.size[0] * SCROLLBAR_WIDTH_FACTOR, min(1, self.size[1] / self._widget.rect.height) * self.size[1])
        else:
            return (min(1, self.size[0] / (self._widget.rect.width + 0.001)) * self.size[0], self.size[1] * SCROLLBAR_WIDTH_FACTOR)

    @property
    def size(self):
        if self._vertical is False:
            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
        else:
            return super().size

    def calculate_scroll_percentage(self, offset, scrollbar=False):
        if self._vertical:
            widget_height = self._widget.rect.height

            if widget_height < self.size[1]:
                return 0

            if scrollbar:
                self._scroll_percentage += offset / (self.size[1] - self.scrollbar_size[1] + 0.001)
            else:
                max_scroll_height = widget_height - self.size[1]
                current_scroll_height = self._scroll_percentage * max_scroll_height
                self._scroll_percentage = (current_scroll_height + offset) / (max_scroll_height + 0.001)
        else:
            widget_width = self._widget.rect.width

            if widget_width < self.size[0]:
                return 0

            if scrollbar:
                self._scroll_percentage += offset / (self.size[0] - self.scrollbar_size[0] + 0.001)
            else:
                max_scoll_width = widget_width - self.size[0]
                current_scroll_width = self._scroll_percentage * max_scoll_width
                self._scroll_percentage = (current_scroll_width + offset) / max_scoll_width

        return min(1, max(0, self._scroll_percentage))

    def calculate_widget_rect(self):
        widget_position = self.calculate_widget_position()
        return pygame.FRect(widget_position[0] - self.position[0], self.position[1] + widget_position[1], self.size[0], self.size[1])

    def calculate_widget_position(self):
        if self._vertical:
            return (0, -self._scroll_percentage * (self._widget.rect.height - self.size[1]))
        else:
            return (-self._scroll_percentage * (self._widget.rect.width - self.size[0]), 0)

    def calculate_relative_scrollbar_position(self):
        if self._vertical:
            vertical_offset = (self.size[1] - self.scrollbar_size[1]) * self._scroll_percentage
            scrollbar_position = (0, vertical_offset)
        else:
            horizontal_offset = (self.size[0] - self.scrollbar_size[0]) * self._scroll_percentage
            scrollbar_position = (horizontal_offset, 0)

        return (scrollbar_position[0] / self.size[0], scrollbar_position[1] / self.size[1])

    def set_widget(self, new_widget):
        self._widget = new_widget
        self.set_image()
        self.set_geometry()

    def set_image(self):
        self.image = pygame.transform.scale(self._empty_surface, self.size)
        self.image.fill(theme['fillPrimary'])

        self._widget.set_image()
        self.image.blit(self._widget.image, self.calculate_widget_position())

        self._scrollbar.set_relative_position(self.calculate_relative_scrollbar_position()) # WRONG USING RELATIVE
        self._scrollbar.set_relative_size((self.scrollbar_size[0] / self.size[1], self.scrollbar_size[1] / self.size[1]))
        self._scrollbar.set_image()
        relative_scrollbar_position = (self._scrollbar.rect.left - self.position[0], self._scrollbar.rect.top - self.position[1])
        self.image.blit(self._scrollbar.image, relative_scrollbar_position)

    def set_geometry(self):
        super().set_geometry()
        self._widget.set_geometry()
        self._scrollbar.set_geometry()

    def set_surface_size(self, new_surface_size):
        super().set_surface_size(new_surface_size)
        self._widget.set_surface_size(new_surface_size)
        self._scrollbar.set_surface_size(new_surface_size)

    def process_event(self, event):
        # WAITING FOR PYGAME-CE 2.5.3 TO RELEASE TO FIX SCROLL FLAGS
        # self.image.scroll(0, SCROLL_FACTOR)
        # self.image.scroll(0, -SCROLL_FACTOR)

        offset = self._scrollbar.process_event(event)

        if offset is not None:
            self.set_image()

            if abs(offset) > 0:
                self._scroll_percentage = self.calculate_scroll_percentage(offset,
 scrollbar=True)

        if self.rect.collidepoint(pygame.mouse.get_pos()):
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 4:
                    self._scroll_percentage = self.calculate_scroll_percentage(-
self.scroll_factor)
                    self.set_image()
                    return
                elif event.button == 5:
                    if self._scroll_percentage == 100:
                        return

                    self._scroll_percentage = self.calculate_scroll_percentage(
self.scroll_factor)
                    self.set_image()
                    return

        widget_event = self._widget.process_event(event, scrolled_pos=self.
calculate_widget_position())
        if widget_event is not None:
            self.set_image()
        return widget_event
```

1.24.23 slider_thumb.py

```
import pygame
from data.widgets.bases.pressable import _Pressable
from data.constants import WidgetState
from data.utils.widget_helpers import create_slider_thumb
from data.managers.theme import theme

class _SliderThumb(_Pressable):
    def __init__(self, radius, border_colour=theme['borderPrimary'], fill_colour=
theme['fillPrimary']):
        super().__init__(
            event=None,
            down_func=self.down_func,
            up_func=self.up_func,
            hover_func=self.hover_func,
            prolonged=True,
            sfx=None
        )
        self._border_colour = border_colour
        self._radius = radius
        self._percent = None

        self.state = WidgetState.BASE
        self.initialise_new_colours(fill_colour)

    def get_position(self):
        return (self.rect.x, self.rect.y)

    def set_position(self, position):
        self.rect = self._thumb_surface.get_rect()
        self.rect.topleft = position

    def get_surface(self):
        return self._thumb_surface

    def set_surface(self, radius, border_width):
        self._thumb_surface = create_slider_thumb(radius, self._colours[self.state
    ], self._border_colour, border_width)

    def get_pressed(self):
        return self._pressed

    def down_func(self):
        self.state = WidgetState.PRESS

    def up_func(self):
        self.state = WidgetState.BASE

    def hover_func(self):
        self.state = WidgetState.HOVER
```

1.24.24 switch.py

```
import pygame
from data.widgets.bases.widget import _Widget
from data.widgets.bases.pressable import _Pressable
from data.constants import WidgetState
from data.utils.widget_helpers import create_switch
from data.components.custom_event import CustomEvent
from data.managers.theme import theme

class Switch(_Pressable, _Widget):
    def __init__(self, relative_height, event, fill_colour=theme['fillTertiary'],
    on_colour=theme['fillSecondary'], off_colour=theme['fillPrimary'], **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,
        )
        _Widget.__init__(self, relative_size=(relative_height * 2, relative_height
    ), scale_mode='height',fill_colour=fill_colour, **kwargs)

        self._on_colour = on_colour
        self._off_colour = off_colour
```

```python
        self._background_colour = None

        self._is_toggled = None
        self.set_toggle_state(False)

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

        self.set_image()
        self.set_geometry()

    def hover_func(self):
        self.set_state_colour(WidgetState.HOVER)

    def set_toggle_state(self, is_toggled):
        self._is_toggled = is_toggled
        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)
```

1.24.25 text.py

```python
import pygame
from data.widgets.bases.widget import _Widget
from data.constants import WidgetState
from data.utils.font_helpers import text_width_to_font_size, text_height_to_font_size, height_to_font_size
from data.utils.widget_helpers import create_text_box
from data.assets import GRAPHICS
```

```python

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]

        if font_size:
            self._relative_font_size = font_size / self.surface_size[1]

        self._center = center
        self.rect = self._font.get_rect(self._text, size=self.font_size)
        self.rect.topleft = self.position

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

        self.set_image()
        self.set_geometry()

    def resize_text(self):
        if self._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:
            ideal_font_size = height_to_font_size(self._font, target_height=(self.size[1] - (self.margin + self.border_width))) / self.surface_size[1]
            new_font_size = text_width_to_font_size(self._text, self._font, (self.size[0] - (self.margin + self.border_width))) / self.surface_size[1]

            if new_font_size < ideal_font_size:
                self._relative_font_size = new_font_size
            else:
                self._relative_font_size = ideal_font_size

    def set_text(self, new_text):
        self._text = new_text

        self.resize_text()
        self.set_image()

    def set_image(self):
        if self._box_colours:
            self.image = create_text_box(self.size, self.border_width, self._box_colours)
        else:
            text_surface = pygame.transform.scale(self._empty_surface, self.size)
            self.image = text_surface

            if self._fill_colour:
                fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
                pygame.draw.rect(self.image, self._fill_colour, fill_rect, border_radius=int(self.border_radius))

        self._font.strength = self._strength
        font_rect_size = self._font.get_rect(self._text, size=self.font_size).size
        if self._center:
            font_position = ((self.size[0] - font_rect_size[0]) / 2, (self.size[1] - font_rect_size[1]) / 2)
        else:
            font_position = (self.margin / 2, (self.size[1] - font_rect_size[1]) / 2)
        self._font.render_to(self.image, font_position, self._text, fgcolor=self._text_colour, size=self.font_size)

        if self._box_colours is None and self.border_width:
            fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
            pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int(self.border_width), border_radius=int(self.border_radius))

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

1.24.26 text_button.py

```
from data.widgets.bases.pressable import _Pressable
from data.widgets.bases.box import _Box
from data.widgets.text import Text
from data.constants import WidgetState, BLUE_BUTTON_COLOURS

class TextButton(_Box, _Pressable, Text):
    def __init__(self, event, **kwargs):
        _Box.__init__(self, box_colours=BLUE_BUTTON_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),
        )
        Text.__init__(self, box_colours=BLUE_BUTTON_COLOURS[WidgetState.BASE], **kwargs)

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

1.24.27 text_input.py

See Section ??.

1.24.28 timer.py

```
import pygame
from data.constants import WidgetState, Colour, BLUE_BUTTON_COLOURS, RED_BUTTON_COLOURS
from data.components.custom_event import CustomEvent
from data.managers.animation import animation
from data.widgets.text import Text

class Timer(Text):
    def __init__(self, active_colour, event=None, start_mins=60, **kwargs):
        box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour == 
    Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]

        self._current_ms = float(start_mins) * 60 * 1000
        self._active_colour = active_colour
        self._active = False
        self._timer_running = False
        self._event = event

        super().__init__(text=self.format_to_text(), fit_vertical=False, 
    box_colours=box_colours, **kwargs)

    def set_active(self, is_active):
        if self._active == is_active:
            return

        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)
```

1.24.29 volume_slider.py

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

```python
from data.components.custom_event import CustomEvent
from data.constants import SettingsEventType
from data.constants import WidgetState
from data.utils.widget_helpers import create_slider
from data.utils.asset_helpers import scale_and_cache
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

        self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self._border_colour, fill_colour=thumb_colour)
        self._gradient_surface = create_slider(self.calculate_slider_size(), self._fill_colour, self.border_width, self._border_colour)

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

    @property
    def position(self):
        '''Minus so easier to position slider by starting from the left edge of the slider instead of the thumb'''
        return (self._relative_position[0] * self.surface_size[0] - (self.size[1] / 2), self._relative_position[1] * self.surface_size[1])

    def calculate_slider_position(self):
        return (self.size[1] / 2, self.size[1] / 4)

    def calculate_slider_size(self):
        return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)

    def calculate_selected_percent(self, mouse_pos):
        if self._last_mouse_x is None:
            return

        x_change = (mouse_pos[0] - self._last_mouse_x) / (self.calculate_slider_size()[0] - 2 * self.border_width)
        return max(0, min(self._selected_percent + x_change, 1))

    def calculate_thumb_position(self):
        gradient_size = self.calculate_slider_size()
        x = gradient_size[0] * self._selected_percent
        y = 0

        return (x, y)

    def relative_to_global_position(self, position):
        relative_x, relative_y = position
        return (relative_x + self.position[0], relative_y + self.position[1])

    def set_image(self):
        gradient_scaled = scale_and_cache(self._gradient_surface, self.calculate_slider_size())
        gradient_position = self.calculate_slider_position()

        self.image = pygame.transform.scale(self._empty_surface, (self.size))
        self.image.blit(gradient_scaled, gradient_position)

        thumb_position = self.calculate_thumb_position()
        self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.border_width)
        self._thumb.set_position(self.relative_to_global_position((thumb_position[0], thumb_position[1])))

        thumb_surface = self._thumb.get_surface()
        self.image.blit(thumb_surface, thumb_position)

    def set_volume(self, volume):
        self._selected_percent = volume
        self.set_image()

    def process_event(self, event):
        if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:
            return

        before_state = self._thumb.state
        self._thumb.process_event(event)
        after_state = self._thumb.state

        if before_state != after_state:
            self.set_image()

            if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:
                self._last_mouse_x = None
                return CustomEvent(SettingsEventType.VOLUME_SLIDER_CLICK, volume=round(self._selected_percent, 3), volume_type=self._volume_type)

        if self._thumb.state == WidgetState.PRESS:
            selected_percent = self.calculate_selected_percent(event.pos)
            self._last_mouse_x = event.pos[0]

            if selected_percent:
                self._selected_percent = selected_percent
                self.set_image()
                return CustomEvent(SettingsEventType.VOLUME_SLIDER_SLIDE)
```

1.24.30 __init__.py

```
from data.widgets.bases.widget import _Widget
from data.widgets.bases.pressable import _Pressable
from data.widgets.bases.circular import _Circular
from data.widgets.bases.box import _Box
from data.widgets.colour_display import _ColourDisplay
from data.widgets.colour_square import _ColourSquare
from data.widgets.colour_slider import _ColourSlider
from data.widgets.slider_thumb import _SliderThumb
from data.widgets.scrollbar import _Scrollbar

from data.widgets.board_thumbnail_button import BoardThumbnailButton
from data.widgets.multiple_icon_button import MultipleIconButton
from data.widgets.reactive_icon_button import ReactiveIconButton
from data.widgets.board_thumbnail import BoardThumbnail
from data.widgets.reactive_button import ReactiveButton
from data.widgets.volume_slider import VolumeSlider
from data.widgets.colour_picker import ColourPicker
from data.widgets.colour_button import ColourButton
from data.widgets.browser_strip import BrowserStrip
```

```python
from data.widgets.piece_display import PieceDisplay
from data.widgets.browser_item import BrowserItem
from data.widgets.text_button import TextButton
from data.widgets.icon_button import IconButton
from data.widgets.scroll_area import ScrollArea
from data.widgets.chessboard import Chessboard
from data.widgets.text_input import TextInput
from data.widgets.rectangle import Rectangle
from data.widgets.move_list import MoveList
from data.widgets.dropdown import Dropdown
from data.widgets.carousel import Carousel
from data.widgets.switch import Switch
from data.widgets.timer import Timer
from data.widgets.text import Text
from data.widgets.icon import Icon

__all__ = ['Text', 'TextButton', 'ColourPicker', 'ColourButton', 'Switch', '
    Dropdown', 'IconButton', 'Icon', 'VolumeSlider', 'TextInput', '
    MultipleIconButton', 'Carousel', 'Timer', 'Rectangle', 'Chessboard', '
    ScrollArea', 'MoveList', 'BoardThumbnail', 'BrowserStrip', 'BrowserItem', '
    PieceDisplay', 'BoardThumbnailButton', 'ReactiveButton', 'ReactiveIconButton']
```

1.25 data\widgets\bases

1.25.1 box.py

```python
from data.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)
```

1.25.2 circular.py

See Section ??.

1.25.3 pressable.py

```python
import pygame
from data.constants import WidgetState
from data.managers.audio import audio
from data.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()
```

1.25.4 widget.py

See Section ??.