1 Technical Solution

1.1 File Tree Diagram

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

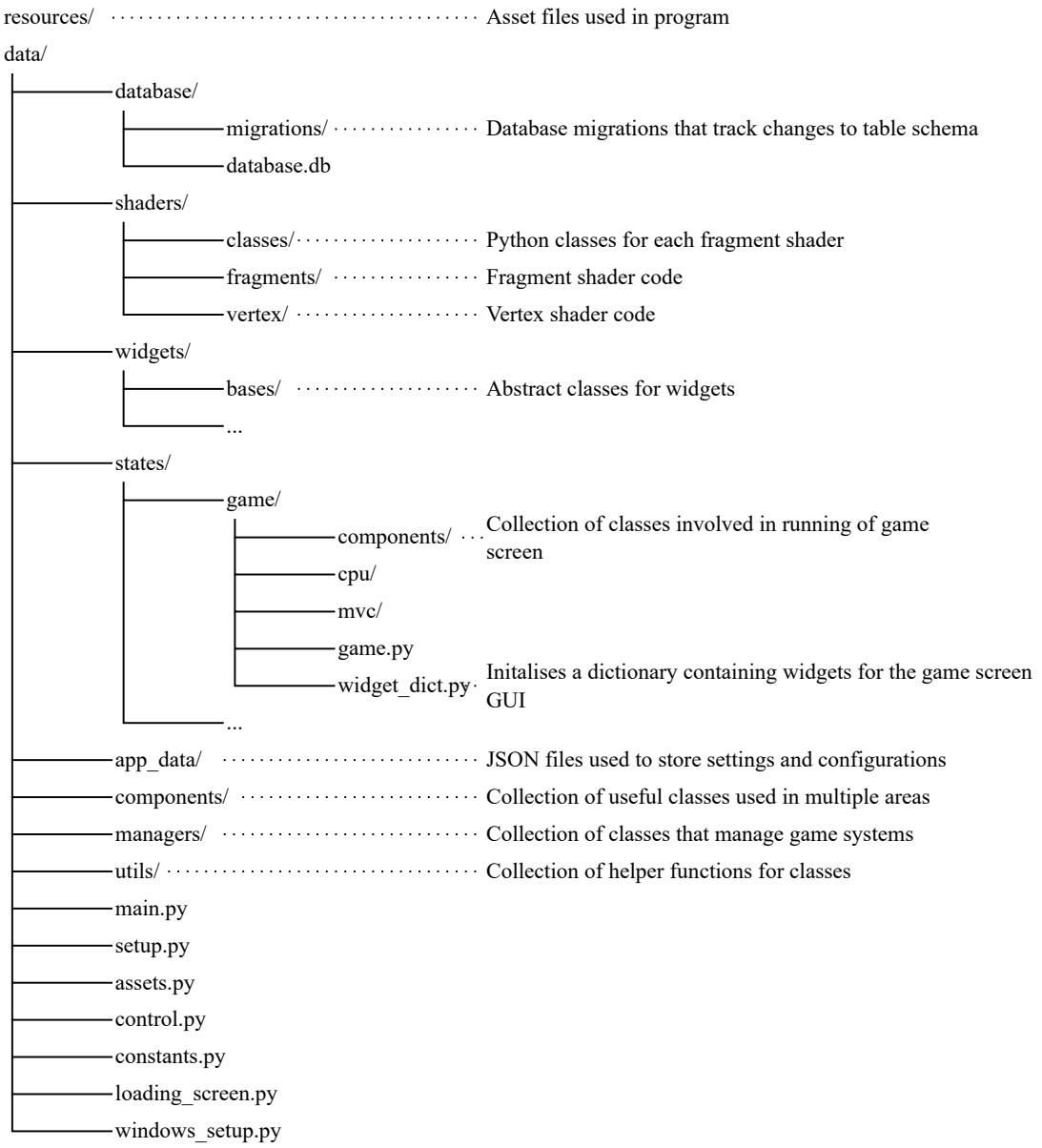

Figure 1: File tree diagram

1.2 Summary of Complexity

- Alpha-beta pruning and transposition table improvements for Minimax
- Shadow mapping and coordinate transformations
- Recursive Depth-First Search tree traversal
- Circular doubly-linked list and stack
- Multipass shaders and gaussian blur
- Aggregate and Window SQL functions
- OOP techniques
- Multithreading (Loading Screen)
- Bitboards
- (Dictionary recursion)
- (Dot product)

1.3 Overview

1.3.1 Main

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

`main.py`

```
from sys import platform
# Initialises Pygame
import data.setup

# Windows OS requires some configuration for Pygame to scale GUI continuously
    while window is being resized
if platform == 'win32':
    import data.windows_setup as win_setup

from data.loading_screen import LoadingScreen

states = [None, None]

def load_states():
    """
    Initialises instances of all screens, executed on another thread with results
    being stored to the main thread by modifying a mutable such as the states list
    """
    from data.control import Control
    from data.states.game.game import Game
    from data.states.menu.menu import Menu
    from data.states.settings.settings import Settings
    from data.states.config.config import Config
    from data.states.browser.browser import Browser
    from data.states.review.review import Review
    from data.states.editor.editor import Editor

```

```
    state_dict = {
        'menu': Menu(),
        'game': Game(),
        'settings': Settings(),
        'config': Config(),
        'browser': Browser(),
        'review': Review(),
        'editor': Editor()
    }

    app = Control()

    states[0] = app
    states[1] = state_dict

loading_screen = LoadingScreen(load_states)

def main():
    """
    Executed by run.py, starts main game loop
    """
    app, state_dict = states

    if platform == 'win32':
        win_setup.set_win_resize_func(app.update_window)

    app.setup_states(state_dict, 'menu')
    app.main_game_loop()
```

1.3.2 Loading Screen

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

loading_screen.py

```
import pygame
import threading
import sys
from pathlib import Path
from data.utils.load_helpers import load_gfx, load_sfx
from data.managers.window import window
from data.managers.audio import audio

FPS = 30
start_ticks = pygame.time.get_ticks()
logo_gfx_path = (Path(__file__).parent / '../resources/graphics/gui/icons/logo/
    logo.png').resolve()
sfx_path_1 = (Path(__file__).parent / '../resources/sfx/loading_screen/
    loading_screen_1.wav').resolve()
sfx_path_2 = (Path(__file__).parent / '../resources/sfx/loading_screen/
    loading_screen_2.wav').resolve()

def easeOutBack(progress):
    """
    Represents a cubic function for easing the logo position
    Starts quickly and has small overshoot, then ends slowly

    Args:
        progress (float): x-value for cubic function ranging from 0-1
```

```
22
23      Returns:
24          float: 2.70x^3 + 1.70x^2 + 0x + 1, where x is time elapsed
25      """
26      c2 = 1.70158
27      c3 = 2.70158
28
29      return c3 * ((progress - 1) ** 3) + c2 * ((progress - 1) ** 2) + 1
30
31  class LoadingScreen:
32      def __init__(self, target_func):
33          """
34          Creates new thread, and sets the load_state() function as its target
35          Then starts draw loop for the loading screen
36
37          Args:
38              target_func (Callable): function to be run on thread
39          """
40          self._clock = pygame.time.Clock()
41          self._thread = threading.Thread(target=target_func)
42          self._thread.start()
43
44          self._logo_surface = load_gfx(logo_gfx_path)
45          self._logo_surface = pygame.transform.scale(self._logo_surface, (96, 96))
46          audio.play_sfx(load_sfx(sfx_path_1))
47          audio.play_sfx(load_sfx(sfx_path_2))
48
49          self.run()
50
51      @property
52      def logo_position(self):
53          duration = 1000
54          displacement = 50
55          elapsed_ticks = pygame.time.get_ticks() - start_ticks
56          progress = min(1, elapsed_ticks / duration)
57          center_pos = ((window.screen.size[0] - self._logo_surface.size[0]) / 2, (
    window.screen.size[1] - self._logo_surface.size[1]) / 2)
58
59          return (center_pos[0], center_pos[1] + displacement - displacement *
    easeOutBack(progress))
60
61      @property
62      def logo_opacity(self):
63          return min(255, (pygame.time.get_ticks() - start_ticks) / 5)
64
65      @property
66      def duration_not_over(self):
67          return (pygame.time.get_ticks() - start_ticks) < 1500
68
69      def event_loop(self):
70          """
71          Handles events for the loading screen, no user input is taken except to
    quit the game
72          """
73          for event in pygame.event.get():
74              if event.type == pygame.QUIT:
75                  pygame.quit()
76                  sys.exit()
77
78      def draw(self):
79          """
80          Draws logo to screen
```

```
        """
        window.screen.fill((0, 0, 0))

        self._logo_surface.set_alpha(self.logo_opacity)
        window.screen.blit(self._logo_surface, self.logo_position)

        window.update()

    def run(self):
        """
        Runs while the thread is still setting up our screens, or the minimum loading screen duration is not reached yet
        """
        while self._thread.is_alive() or self.duration_not_over:
            self.event_loop()
            self.draw()
            self._clock.tick(FPS)
```

1.3.3 Helper functions

These files provide useful functions for different classes.
`asset_helpers.py (Functions used for assets and pygame Surfaces)`

```
import pygame
from PIL import Image
from functools import cache
from random import sample, randint
import math

@cache
def scale_and_cache(image, target_size):
    """
    Caches image when resized repeatedly

    Args:
        image (pygame.Surface): Image surface to be resized
        target_size (tuple[float, float]): New image size

    Returns:
        pygame.Surface: Resized image surface
    """
    return pygame.transform.scale(image, target_size)

@cache
def smoothscale_and_cache(image, target_size):
    """
    Same as scale_and_cache, but with the Pygame smoothscale function

    Args:
        image (pygame.Surface): Image surface to be resized
        target_size (tuple[float, float]): New image size

    Returns:
        pygame.Surface: Resized image surface
    """
    return pygame.transform.smoothscale(image, target_size)

def gif_to_frames(path):
    """
    Uses the PIL library to break down GIFs into individual frames

```

```
    Args:
        path (str): Directory path to GIF file

    Yields:
        PIL.Image: Single frame
    """
    try:
        image = Image.open(path)

        first_frame = image.copy().convert('RGBA')
        yield first_frame
        image.seek(1)

        while True:
            current_frame = image.copy()
            yield current_frame
            image.seek(image.tell() + 1)
    except EOFError:
        pass

def get_perimeter_sample(image_size, number):
    """
    Used for particle drawing class, generates roughly equally distributed points around a rectangular image surface's perimeter

    Args:
        image_size (tuple[float, float]): Image surface size
        number (int): Number of points to be generated

    Returns:
        list[tuple[int, int], ...]: List of random points on perimeter of image surface
    """
    perimeter = 2 * (image_size[0] + image_size[1])
    # Flatten perimeter to a single number representing the distance from the top-middle of the surface going clockwise, and create a list of equally spaced points
    perimeter_offsets = [(image_size[0] / 2) + (i * perimeter / number) for i in range(0, number)]
    pos_list = []

    for perimeter_offset in perimeter_offsets:
        # For every point, add a random offset
        max_displacement = int(perimeter / (number * 4))
        perimeter_offset += randint(-max_displacement, max_displacement)

        if perimeter_offset > perimeter:
            perimeter_offset -= perimeter

        # Convert 1D distance back into 2D points on image surface perimeter
        if perimeter_offset < image_size[0]:
            pos_list.append((perimeter_offset, 0))
        elif perimeter_offset < image_size[0] + image_size[1]:
            pos_list.append((image_size[0], perimeter_offset - image_size[0]))
        elif perimeter_offset < image_size[0] + image_size[1] + image_size[0]:
            pos_list.append((perimeter_offset - image_size[0] - image_size[1], image_size[1]))
        else:
            pos_list.append((0, perimeter - perimeter_offset))
    return pos_list

def get_angle_between_vectors(u, v, deg=True):
```

```
95      """
96      Uses the dot product formula to find the angle between two vectors
97
98      Args:
99          u (list[int, int]): Vector 1
100         v (list[int, int]): Vector 2
101         deg (bool, optional): Return results in degrees. Defaults to True.
102
103     Returns:
104         float: Angle between vectors
105     """
106     dot_product = sum(i * j for (i, j) in zip(u, v))
107     u_magnitude = math.sqrt(u[0] ** 2 + u[1] ** 2)
108     v_magnitude = math.sqrt(v[0] ** 2 + v[1] ** 2)
109
110     cos_angle = dot_product / (u_magnitude * v_magnitude)
111     radians = math.acos(min(max(cos_angle, -1), 1))
112
113     if deg:
114         return math.degrees(radians)
115     else:
116         return radians
117
118 def get_rotational_angle(u, v, deg=True):
119     """
120     Get bearing angle relative to positive x-axis centered on second vector
121
122     Args:
123         u (list[int, int]): Vector 1
124         v (list[int, int]): Vector 2, set as center of axes
125         deg (bool, optional): Return results in degrees. Defaults to True.
126
127     Returns:
128         float: Bearing angle between vectors
129     """
130     radians = math.atan2(u[1] - v[1], u[0] -v[0])
131
132     if deg:
133         return math.degrees(radians)
134     else:
135         return radians
136
137 def get_vector(src_vertex, dest_vertex):
138     """
139     Get vector describing translation between two points
140
141     Args:
142         src_vertex (list[int, int]): Source vertex
143         dest_vertex (list[int, int]): Destination vertex
144
145     Returns:
146         tuple[int, int]: Vector between the two points
147     """
148     return (dest_vertex[0] - src_vertex[0], dest_vertex[1] - src_vertex[1])
149
150 def get_next_corner(vertex, image_size):
151     """
152     Used in particle drawing system, finds coordinates of the next corner going
        clockwise, given a point on the perimeter
153
154     Args:
155         vertex (list[int, int]): Point on perimeter
```

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

```
212         frame_index = calculate_frame_index(current_time, 0, len(background), fps
    =8)
213         scaled_background = scale_and_cache(background[frame_index], screen.size)
214         screen.blit(scaled_background, (0, 0))
215     else:
216         scaled_background = scale_and_cache(background, screen.size)
217         screen.blit(scaled_background, (0, 0))
218 
219 def get_highlighted_icon(icon):
220     """
221     Used for pressable icons, draws overlay on icon to show as pressed
222 
223     Args:
224         icon (pygame.Surface): Icon surface
225 
226     Returns:
227         pygame.Surface: Icon with overlay drawn on top
228     """
229     icon_copy = icon.copy()
230     overlay = pygame.Surface((icon.get_width(), icon.get_height()), pygame.
    SRCALPHA)
231     overlay.fill((0, 0, 0, 128))
232     icon_copy.blit(overlay, (0, 0))
233     return icon_copy
```

data_helpers.py (Functions used for file handling and JSON parsing)

```
1  import json
2  from pathlib import Path
3  
4  module_path = Path(__file__).parent
5  default_file_path = (module_path / '../app_data/default_settings.json').resolve()
6  user_file_path = (module_path / '../app_data/user_settings.json').resolve()
7  themes_file_path = (module_path / '../app_data/themes.json').resolve()
8  
9  def load_json(path):
10     """
11     Args:
12         path (str): Path to JSON file
13 
14     Raises:
15         Exception: Invalid file
16 
17     Returns:
18         dict: Parsed JSON file
19     """
20     try:
21         with open(path, 'r') as f:
22             file = json.load(f)
23 
24         return file
25     except:
26         raise Exception('Invalid JSON file (data_helpers.py)')
27 
28 def get_user_settings():
29     return load_json(user_file_path)
30 
31 def get_default_settings():
32     return load_json(default_file_path)
33 
34 def get_themes():
```

```
    return load_json(themes_file_path)

def update_user_settings(data):
    """
    Rewrites JSON file for user settings with new data

    Args:
        data (dict): Dictionary storing updated user settings

    Raises:
        Exception: Invalid file
    """
    try:
        with open(user_file_path, 'w') as f:
            json.dump(data, f, indent=4)
    except:
        raise Exception('Invalid JSON file (data_helpers.py)')
```

widget_helpers.py (Files used for creating widgets)

```
import pygame
from math import sqrt

def create_slider(size, fill_colour, border_width, border_colour):
    """
    Creates surface for sliders

    Args:
        size (list[int, int]): Image size
        fill_colour (pygame.Color): Fill (inner) colour
        border_width (float): Border width
        border_colour (pygame.Color): Border colour

    Returns:
        pygame.Surface: Slider image surface
    """
    gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
    border_rect = pygame.FRect((0, 0, gradient_surface.width, gradient_surface.
    height))

    # Draws rectangle with a border radius half of image height, to draw an
    rectangle with semicurclar cap (obround)
    pygame.draw.rect(gradient_surface, fill_colour, border_rect, border_radius=int
    (size[1] / 2))
    pygame.draw.rect(gradient_surface, border_colour, border_rect , width=int(
    border_width), border_radius=int(size[1] / 2))

    return gradient_surface

def create_slider_gradient(size, border_width, border_colour):
    """
    Draws surface for colour slider, with a full colour gradient as fill colour

    Args:
        size (list[int, int]): Image size
        border_width (float): Border width
        border_colour (pygame.Color): Border colour

    Returns:
        pygame.Surface: Slider image surface
    """
```

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

```
        width=int(border_width))
95      pygame.draw.circle(thumb_surface, colour, (radius, radius), (radius -
        border_width))
96
97      return thumb_surface
98
99  def create_square_gradient(side_length, colour):
100     """
101     Creates a square gradient for the colour picker widget, gradient transitioning
         between saturation and value
102     Uses smoothscale to blend between colour values for individual pixels
103
104     Args:
105         side_length (float): Length of a square side
106         colour (pygame.Color): Colour with desired hue value
107
108     Returns:
109         pygame.Surface: Square gradient surface
110     """
111     square_surface = pygame.Surface((side_length, side_length))
112
113     mix_1 = pygame.Surface((1, 2))
114     mix_1.fill((255, 255, 255))
115     mix_1.set_at((0, 1), (0, 0, 0))
116     mix_1 = pygame.transform.smoothscale(mix_1, (side_length, side_length))
117
118     hue = colour.hsva[0]
119     saturated_rgb = pygame.Color(0)
120     saturated_rgb.hsva = (hue, 100, 100)
121
122     mix_2 = pygame.Surface((2, 1))
123     mix_2.fill((255, 255, 255))
124     mix_2.set_at((1, 0), saturated_rgb)
125     mix_2 = pygame.transform.smoothscale(mix_2,(side_length, side_length))
126
127     mix_1.blit(mix_2, (0, 0), special_flags=pygame.BLEND_MULT)
128
129     square_surface.blit(mix_1, (0, 0))
130
131     return square_surface
132
133 def create_switch(size, colour):
134     """
135     Creates surface for switch toggle widget
136
137     Args:
138         size (list[int, int]): Image size
139         colour (pygame.Color): Fill colour
140
141     Returns:
142         pygame.Surface: Switch surface
143     """
144     switch_surface = pygame.Surface((size[0], size[1]), pygame.SRCALPHA)
145     pygame.draw.rect(switch_surface, colour, (0, 0, size[0], size[1]),
        border_radius=int(size[1] / 2))
146
147     return switch_surface
148
149 def create_text_box(size, border_width, colours):
150     """
151     Creates bordered textbox with shadow, flat, and highlighted vertical regions
152
```

```
153     Args:
154         size (list[int, int]): Image size
155         border_width (float): Border width
156         colours (list[pygame.Color, ...]): List of 4 colours, representing border
    colour, shadow colour, flat colour and highlighted colour
157 
158     Returns:
159         pygame.Surface: Textbox surface
160     """
161     surface = pygame.Surface(size, pygame.SRCALPHA)
162 
163     pygame.draw.rect(surface, colours[0], (0, 0, *size))
164     pygame.draw.rect(surface, colours[2], (border_width, border_width, size[0] - 2
     * border_width, size[1] - 2 * border_width))
165     pygame.draw.rect(surface, colours[3], (border_width, border_width, size[0] - 2
     * border_width, border_width))
166     pygame.draw.rect(surface, colours[1], (border_width, size[1] - 2 *
    border_width, size[0] - 2 * border_width, border_width))
167 
168     return surface
```

1.3.4 Theme

1.4 GUI

1.4.1 Laser

1.4.2 Particles

1.5 Game

1.5.1 Database