# 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.

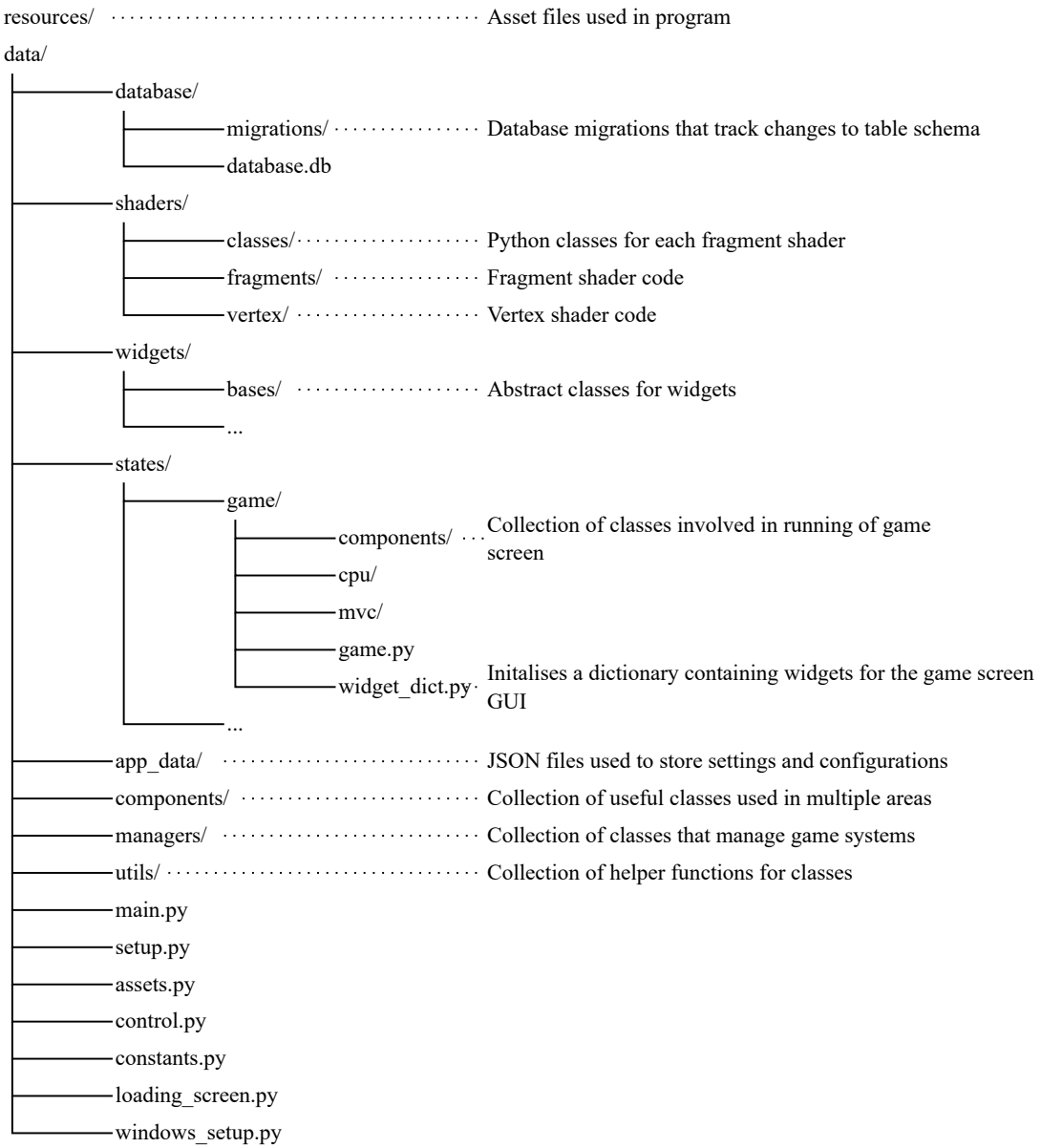


Figure 1: File tree diagram

## 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.

```
from sys import platform
import data.setup

if platform == 'win32':
    import data.windows_setup as win_setup # PUT BEFORE LOADINGSCREEN TO SCALE
    STARTUP WINDOW CORRECTLY

from data.loading_screen import LoadingScreen

states = [None, None]

def load_states():
    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():
    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.

```
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):
    c1 = 1.70158
    c3 = c1 + 1

    return 1 + c3 * pow(progress - 1, 3) + c1 * pow(progress - 1, 2)

class LoadingScreen:
    def __init__(self, target_func):
        self._clock = pygame.time.Clock()
        self._thread = threading.Thread(target=target_func)
        self._thread.start()

        self._logo_surface = load_gfx(logo_gfx_path)
        self._logo_surface = pygame.transform.scale(self._logo_surface, (96, 96))
        audio.play_sfx(load_sfx(sfx_path_1))
        audio.play_sfx(load_sfx(sfx_path_2))

        self.run()

```

```
34      @property
35      def logo_position(self):
36          duration = 1000
37          displacement = 50
38          elapsed_ticks = pygame.time.get_ticks() - start_ticks
39          progress = min(1, elapsed_ticks / duration)
40          center_pos = ((window.screen.size[0] - self._logo_surface.size[0]) / 2, (
    window.screen.size[1] - self._logo_surface.size[1]) / 2)
41
42          return (center_pos[0], center_pos[1] + displacement - displacement *
    easeOutBack(progress))
43
44      @property
45      def logo_opacity(self):
46          return min(255, (pygame.time.get_ticks() - start_ticks) / 5)
47
48      @property
49      def duration_not_over(self):
50          return (pygame.time.get_ticks() - start_ticks) < 1500
51
52      def event_loop(self):
53          for event in pygame.event.get():
54              if event.type == pygame.QUIT:
55                  pygame.quit()
56                  sys.exit()
57
58      def draw(self):
59          window.screen.fill((0, 0, 0))
60
61          self._logo_surface.set_alpha(self.logo_opacity)
62          window.screen.blit(self._logo_surface, self.logo_position)
63
64          window.update()
65
66      def run(self):
67          while self._thread.is_alive() or self.duration_not_over:
68              self.event_loop()
69              self.draw()
70              self._clock.tick(FPS)
```