```
1 from pathlib import Path
2 from data.utils.load_helpers import *
4 module_path = Path(__file__).parent
5 GRAPHICS = load_all_gfx((module_path / '../resources/graphics').resolve())
6 FONTS = load_all_fonts((module_path / '../resources/fonts').resolve())
7 SFX = load_all_sfx((module_path / '../resources/sfx').resolve())
8 MUSIC = load_all_music((module_path / '../resources/music').resolve())
10 DEFAULT_FONT=FONTS['vhs-gothic']
11 DEFAULT_FONT.strong = True
12 DEFAULT_FONT.strength = 0.05
1 import pygame
2 from enum import IntEnum, StrEnum, auto
_{4} BG_COLOUR = (0, 0, 0)
5 PAUSE_COLOUR = (50, 50, 50, 128)
6 OVERLAY_COLOUR_LIGHT = (*pygame.Color('0xf14e52').rgb, 128)
7 OVERLAY_COLOUR_DARK = (*pygame.Color('0x9b222b').rgb, 192)
8 SCREEN_SIZE = (1200, 600)
9 # SCREEN_SIZE = (600, 600)
10 SCREEN_FLAGS = pygame.HWSURFACE | pygame.DOUBLEBUF | pygame.RESIZABLE | pygame.
     OPENGL
11 STARTING_SQUARE_SIZE = (SCREEN_SIZE[1] * 0.64) / 8 #Board height divded by 8
12 EMPTY_BB = 0
13 A_FILE_MASK = 0
     14 J_FILE_MASK = 0
     15 ONE_RANK_MASK = 0
     16 EIGHT_RANK_MASK = 0
     17 \text{ TEST\_MASK} = 0
     18 GAMES_PER_PAGE = 10
19
20 class CursorMode(IntEnum):
21
     ARROW = auto()
     IBEAM = auto()
22
    OPENHAND = auto()
23
    CLOSEDHAND = auto()
    NO = auto()
25
26
27 class ShaderType(StrEnum):
    BASE = auto()
28
     SHAKE = auto()
29
     BLOOM = auto()
30
     GRAYSCALE = auto()
3.1
     CRT = auto()
     RAYS = auto()
33
     CHROMATIC_ABBREVIATION = auto()
34
     BACKGROUND_WAVES = auto()
35
     BACKGROUND_BALATRO = auto()
3.6
    BACKGROUND_LASERS = auto()
37
    BACKGROUND_GRADIENT = auto()
```

```
BACKGROUND NONE = auto()
          _BLUR = auto()
41
          _HIGHLIGHT_BRIGHTNESS = auto()
          _HIGHLIGHT_COLOUR = auto()
43
          _CALIBRATE = auto()
44
         _LIGHTMAP = auto()
         _SHADOWMAP = auto()
_OCCLUSION = auto()
46
47
         _BLEND = auto()
48
          _CROP = auto()
49
50
51 SHADER_MAP = {
         'default': [
52
                ShaderType.BL00M
53
5.4
          'retro':
55
56
                Shader Type . CRT
5.7
          'really_retro': [
                 Shader Type . CRT
59
                 Shader Type . GRAYSCALE
6.0
61
62 }
63
64 class TranspositionFlag(StrEnum):
          LOWER = auto()
EXACT = auto()
6.5
66
          UPPER = auto()
67
68
69 class Miscellaneous(StrEnum):
        PLACEHOLDER = auto()
7.0
          DRAW = auto()
7.1
72
73 class WidgetState(StrEnum):
          BASE = auto()
          HOVER = auto()
PRESS = auto()
75
7.6
78 BLUE_BUTTON_COLOURS = {
          WidgetState.BASE: ['0x1c2638', '0x23495d', '0x39707a', '0x95e0cc'],
79
          WidgetState.HOVER: ['0xdaf2e9', '0x23495d', '0x39707a', '0x95e0cc'], WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
80
8.1
82 }
83
84 INPUT COLOURS = {
          WidgetState.BASE: ['0x1c2638', '0x39707a', '0x23495d', '0x95e0cc'], WidgetState.HOVER: ['0xdaf2e9', '0x39707a', '0x23495d', '0x95e0cc'], WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0x1c2638', '0x39707a']
86
87
88 }
89
90 RED_BUTTON_COLOURS = {
          WidgetState.BASE: ['0x000000', '0x1c2638', '0x9b222b', '0xf14e52'], WidgetState.HOVER: ['0xdaf2e9', '0x1c2638', '0x9b222b', '0xf14e52'], WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0xf14e52', '0x95e0cc']
91
92
93
94 }
9.5
96 LOCKED_RED_BUTTON_COLOURS = {
          WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'], WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'], WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0xf14e52']
97
98
99
100 }
```

```
102 LOCKED_BLUE_BUTTON_COLOURS = {
        WidgetState.BASE: ['0x000000', '0x0000000', '0x1c2638', '0x23495d'],
103
       WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'], WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
105
106 }
107
108 class StatusText(StrEnum):
       PLAYER MOVE = auto()
        CPU_MOVE = auto()
110
       WIN = auto()
DRAW = auto()
112
113
114 class EditorEventType(StrEnum):
        MENU_CLICK = auto()
115
        PICK_PIECE_CLICK = auto()
116
        ROTATE_PIECE_CLICK = auto()
117
118
        COPY_CLICK = auto()
       EMPTY_CLICK = auto()
119
        RESET_CLICK = auto()
120
        BLUE_START_CLICK = auto()
121
       RED_START_CLICK = auto()
122
        START_CLICK = auto()
123
        CONFIG_CLICK = auto()
124
       ERASE_CLICK = auto()
125
       MOVE_CLICK = auto()
126
       HELP_CLICK = auto()
127
128
129 class ReviewEventType(StrEnum):
       MENU_CLICK = auto()
130
131
        PREVIOUS_CLICK = auto()
        NEXT_CLICK = auto()
132
       HELP_CLICK = auto()
133
134
135 class BrowserEventType(StrEnum):
        MENU_CLICK = auto()
        BROWSER_STRIP_CLICK = auto()
137
        COPY_CLICK = auto()
138
        DELETE_CLICK = auto()
       REVIEW_CLICK = auto()
140
       FILTER_COLUMN_CLICK = auto()
141
        FILTER_ASCEND_CLICK = auto()
142
        PAGE_CLICK = auto()
143
       HELP_CLICK = auto()
144
145
146 class GameEventType(StrEnum):
147
        BOARD_CLICK = auto()
        PIECE_CLICK = auto()
148
        PAUSE_CLICK = auto()
149
150
        MENU_CLICK = auto()
        GAME_CLICK = auto()
151
        HELP_CLICK = auto()
152
        TUTORIAL_CLICK = auto()
153
        RESIGN_CLICK = auto()
154
        DRAW_CLICK = auto()
        REVIEW_CLICK = auto()
156
       PIECE_DROP = auto()
157
        UPDATE_PIECES = auto()
158
        ROTATE_PIECE = auto()
159
160
       SET_LASER = auto()
        TIMER_END = auto()
161
162
```

```
163 class MenuEventType(StrEnum):
       CONFIG_CLICK = auto()
164
       SETTINGS_CLICK = auto()
       BROWSER_CLICK = auto()
       QUIT_CLICK = auto()
167
       CREDITS_CLICK = auto()
168
169
170 class SettingsEventType(StrEnum):
       RESET_DEFAULT = auto()
171
       RESET_USER = auto()
172
       MENU_CLICK = auto()
173
174
       COLOUR_SLIDER_SLIDE = auto()
       COLOUR_SLIDER_CLICK = auto()
175
       COLOUR_PICKER_HOVER = auto()
176
       PRIMARY_COLOUR_PICKER_CLICK = auto()
177
       SECONDARY_COLOUR_PICKER_CLICK = auto()
178
       PRIMARY_COLOUR_BUTTON_CLICK = auto()
179
180
       SECONDARY_COLOUR_BUTTON_CLICK = auto()
       VOLUME_SLIDER_SLIDE = auto()
181
       VOLUME_SLIDER_CLICK = auto()
       SHADER_PICKER_CLICK = auto()
183
       OPENGL\_CLICK = auto()
184
       DROPDOWN_CLICK = auto()
       PARTICLES_CLICK = auto()
186
187
188 class ConfigEventType(StrEnum):
       GAME_CLICK = auto()
189
       MENU_CLICK = auto()
190
       FEN_STRING_TYPE = auto()
191
       TIME_TYPE = auto()
192
       TIME_CLICK = auto()
       PVP_CLICK = auto()
194
       PVC_CLICK = auto()
195
       CPU_DEPTH_CLICK = auto()
196
       PRESET_CLICK = auto()
197
198
       SETUP_CLICK = auto()
       COLOUR_CLICK = auto()
199
       HELP_CLICK = auto()
200
201
202 class Colour(IntEnum):
       BLUE = 0
203
       RED = 1
204
205
206
       def get_flipped_colour(self):
           if self == Colour.BLUE:
207
                return Colour.RED
208
           elif self == Colour.RED:
209
               return Colour.BLUE
210
211
212 class Piece(StrEnum):
      SPHINX = 's'
213
       PYRAMID = 'p'
214
       ANUBIS = 'n'
215
       SCARAB = 'r'
216
       PHAROAH = 'f'
217
218
219 class Score(IntEnum):
     PHAROAH = 0
220
       SPHINX = 0
221
       PYRAMID = 100
222
       ANUBIS = 110
223
       SCARAB = 200
224
```

```
225
        MOVE = 4
226
        POSITION = 11
227
        PHAROAH_SAFETY = 31
228
        C\,H\,E\,C\,K\,M\,A\,T\,E \quad = \quad 1\,0\,0\,0\,0\,0
229
        INFINITE = 6969696969
230
231
232 class Rank(IntEnum):
233
        ONE = 0
        TWO = 1
234
        THREE = 2
        FOUR = 3
236
       FIVE = 4
237
        SIX = 5
238
239
        SEVEN = 6
        EIGHT = 7
240
241
242 class File(IntEnum):
       A = 0
243
244
        B = 1
        C = 2
245
       D = 3
246
247
        E = 4
        F = 5
248
        G = 6
249
        H = 7
250
       I = 8
251
        J = 9
252
253
254 class Rotation(StrEnum):
255
        UP = 'a'
        RIGHT = 'b'
256
        DOMN = 'c'
257
        LEFT = 'd'
258
259
260
        def to_angle(self):
            if self == Rotation.UP:
261
                return 0
262
263
            elif self == Rotation.RIGHT:
            return 270
elif self == Rotation.DOWN:
264
265
                return 180
266
            elif self == Rotation.LEFT:
267
                 return 90
268
269
       def get_clockwise(self):
270
271
             if self == Rotation.UP:
                return Rotation.RIGHT
272
            elif self == Rotation.RIGHT:
273
274
                return Rotation.DOWN
            elif self == Rotation.DOWN:
275
276
                 return Rotation.LEFT
            elif self == Rotation.LEFT:
277
                 return Rotation.UP
278
279
       def get_anticlockwise(self):
280
            if self == Rotation.UP:
281
                 return Rotation.LEFT
282
            elif self == Rotation.RIGHT:
283
284
                 return Rotation.UP
             elif self == Rotation.DOWN:
285
                 return Rotation.RIGHT
286
```

```
elif self == Rotation.LEFT:
287
               return Rotation.DOWN
288
289
       def get_opposite(self):
290
           return self.get_clockwise().get_clockwise()
291
292
293 class RotationIndex(IntEnum):
       FIRSTBIT = 0
294
       SECONDBIT = 1
295
296
297 class RotationDirection(StrEnum):
298
       CLOCKWISE = 'cw'
       ANTICLOCKWISE = 'acw'
299
300
301
       def get_opposite(self):
           if self == RotationDirection.CLOCKWISE:
302
               return RotationDirection.ANTICLOCKWISE
303
304
           elif self == RotationDirection.ANTICLOCKWISE:
                return RotationDirection.CLOCKWISE
305
306
307 class MoveType(StrEnum):
       MOVE = 'm
308
       ROTATE = 'r'
309
310
311 class LaserType(IntEnum):
     END = 0
312
       STRAIGHT = 1
313
314
       CORNER = 2
315
316 class LaserDirection(IntEnum):
      FROM_TOP = 1
       FROM_RIGHT = 2
318
       FROM_BOTTOM = 3
319
       FROM_LEFT = 4
 1 import pygame
 2 from data.components.widget_group import WidgetGroup
 3 from data.managers.logs import initialise_logger
 4 from data.managers.cursor import CursorManager
 5 from data.managers.animation import animation
 6 from data.managers.window import window
 7 from data.managers.audio import audio
 8 from data.managers.theme import theme
 9 from data.assets import DEFAULT_FONT
11 logger = initialise_logger(__file__)
13 \text{ FPS} = 60
14 SHOW_FPS = False
15 start_ticks = pygame.time.get_ticks()
17 class Control:
      def __init__(self):
           self.done = False
19
           self._clock = pygame.time.Clock()
20
       def setup_states(self, state_dict, start_state):
22
23
           self.state_dict = state_dict
           self.state_name = start_state
24
2.5
           self.state = self.state_dict[self.state_name]
26
           self.state.startup()
```

```
def flip_state(self):
29
           self.state.done = False
3.0
           persist = self.state.cleanup()
31
32
           previous, self.state_name = self.state_name, self.state.next
33
34
           self.state = self.state_dict[self.state_name]
3.5
           self.state.previous = previous
36
           self.state.startup(persist)
37
38
39
      def update(self):
40
           if self.state.quit:
               self.done = True
41
           elif self.state.done:
42
              self.flip_state()
43
44
45
           self._clock.tick(FPS)
           animation.set_delta_time()
46
47
           self.state.update()
48
49
           if SHOW_FPS:
50
               self.draw_fps()
5.1
52
           window.update()
53
54
55
     def main_game_loop(self):
           while not self.done:
56
               self.event_loop()
5.7
58
               self.update()
59
     def update_window(self, resize=False):
60
           if resize:
61
               self.update_native_window_size()
62
63
               window.handle_resize()
               self.state.handle_resize()
64
6.5
           self.update()
66
67
      def draw_fps(self):
68
           fps = str(int(self._clock.get_fps()))
69
           DEFAULT_FONT.strength = 0.1
7.0
           DEFAULT_FONT.render_to(window.screen, (0, 0), fps, fgcolor=theme['
71
      textError'], size=15)
73
      def update_native_window_size(self):
           x, y = window.size
74
75
76
           max_window_x = 100000
          max_window_y = x / 1.4
7.7
           min_window_x = 400
78
79
          min_window_y = min_window_x/1.4
80
          if x / y < 1.4:
              min_window_x = x
82
83
           min_window_size = (min_window_x, min_window_y)
84
           max_window_size = (max_window_x, max_window_y)
8.5
86
           window.minimum_size = min_window_size
           window.maximum_size = max_window_size
87
88
```

```
def event_loop(self):
           for event in pygame.event.get():
90
                if event.type == pygame.QUIT:
91
                    self.done = True
93
               if event.type == pygame.MOUSEBUTTONDOWN and event.button != 1: # ONLY
94
       PROCESS LEFT CLICKS
9.5
96
                self.state.get_event(event)
97
98
99 class _State:
100
      def __init__(self):
           self.next = None
101
           self.previous = None
102
           self.done = False
103
           self.quit = False
104
           self.persist = {}
106
           self._cursor = CursorManager()
107
           self._widget_group = None
108
       def startup(self, widgets=None, music=None):
           if widgets:
                self._widget_group = WidgetGroup(widgets)
112
                self._widget_group.handle_resize(window.size)
113
114
115
           if music:
                audio.play_music(music)
116
118
           logger.info(f'starting {self.__class__._name__.lower()}.py')
119
120
       def cleanup(self):
           logger.info(f'cleaning {self.__class__._name__.lower()}.py')
121
       def draw(self):
123
           raise NotImplementedError
124
125
       def get_event(self, event):
126
           raise NotImplementedError
127
128
       def handle_resize(self):
129
           self._widget_group.handle_resize(window.size)
130
131
       def update(self, **kwargs):
132
133
           self.draw()
 1 import pygame
 2 import threading
 3 import sys
 4 from pathlib import Path
 5 from data.utils.load_helpers import load_gfx, load_sfx
 6 from data.managers.window import window
 7 from data.managers.audio import audio
 9 \text{ FPS} = 30
10 start_ticks = pygame.time.get_ticks()
11 logo_gfx_path = (Path(__file__).parent / '../resources/graphics/gui/icons/logo/
       logo.png').resolve()
12 sfx_path_1 = (Path(__file__).parent / '../resources/sfx/loading_screen/
       loading_screen_1.wav').resolve()
18 sfx_path_2 = (Path(__file__).parent / '../resources/sfx/loading_screen/
```

```
loading_screen_2.wav').resolve()
14
15 def easeOutBack(progress):
      Represents a cubic function for easing the logo position.
17
      Starts quickly and has small overshoot, then ends slowly.
18
19
2.0
          progress (float): x-value for cubic function ranging from 0-1.
2.1
22
      Returns:
23
      float: 2.70x^3 + 1.70x^2 + 0x + 1, where x is time elapsed.
24
25
      c2 = 1.70158
26
      c3 = 2.70158
27
28
      return c3 * ((progress - 1) ** 3) + c2 * ((progress - 1) ** 2) + 1
29
30
31 class LoadingScreen:
      def __init__(self, target_func):
33
          Creates new thread, and sets the load_state() function as its target.
3.4
          Then starts draw loop for the loading screen.
35
36
37
          Args:
          target_func (Callable): function to be run on thread.
38
39
40
          self._clock = pygame.time.Clock()
          self._thread = threading.Thread(target=target_func)
41
          self._thread.start()
42
43
          self._logo_surface = load_gfx(logo_gfx_path)
44
45
          self._logo_surface = pygame.transform.scale(self._logo_surface, (96, 96))
          audio.play_sfx(load_sfx(sfx_path_1))
46
          audio.play_sfx(load_sfx(sfx_path_2))
47
48
          self.run()
49
5.0
51
      @property
      def logo_position(self):
52
53
           duration = 1000
          displacement = 50
54
           elapsed_ticks = pygame.time.get_ticks() - start_ticks
5.5
56
          progress = min(1, elapsed_ticks / duration)
           center_pos = ((window.screen.size[0] - self._logo_surface.size[0]) / 2, (
57
      window.screen.size[1] - self._logo_surface.size[1]) / 2)
           return (center_pos[0], center_pos[1] + displacement - displacement *
59
      easeOutBack(progress))
60
6.1
      @property
      def logo_opacity(self):
62
          return min(255, (pygame.time.get_ticks() - start_ticks) / 5)
63
64
      def duration_not_over(self):
66
          return (pygame.time.get_ticks() - start_ticks) < 1500</pre>
67
68
6.9
      def event_loop(self):
70
          Handles events for the loading screen, no user input is taken except to
71
      quit the game.
```

```
0.00
72
73
           for event in pygame.event.get():
               if event.type == pygame.QUIT:
7.4
                   pygame.quit()
75
                    sys.exit()
76
7.7
      def draw(self):
78
7.9
           Draws logo to screen.
80
81
           window.screen.fill((0, 0, 0))
82
83
           self._logo_surface.set_alpha(self.logo_opacity)
84
           window.screen.blit(self._logo_surface, self.logo_position)
8.5
86
           window.update()
87
88
89
      def run(self):
9.0
           Runs while the thread is still setting up our screens, or the minimum
91
      loading screen duration is not reached yet.
92
           while self._thread.is_alive() or self.duration_not_over:
93
               self.event_loop()
94
95
               self.draw()
               self._clock.tick(FPS)
1 from sys import platform
2 # Initialises Pygame
3 import data.setup
5 # Windows OS requires some configuration for Pygame to scale GUI continuously
      while window is being resized
6 if platform == 'win32':
      import data.windows_setup as win_setup
{\tt 9} \  \, \textbf{from} \  \, \textbf{data.loading\_screen} \  \, \textbf{import} \  \, \textbf{LoadingScreen}
10
states = [None, None]
13 def load_states():
14
      Initialises instances of all screens, executed on another thread with results
15
      being stored to the main thread by modifying a mutable such as the states list
      from data.control import Control
17
18
      from data.states.game.game import Game
      from data.states.menu.menu import Menu
      from data.states.settings.settings import Settings
20
21
      from data.states.config.config import Config
22
      from data.states.browser.browser import Browser
      from data.states.review.review import Review
23
      from data.states.editor.editor import Editor
24
25
       state_dict = {
26
           'menu': Menu(),
           'game': Game(),
28
           'settings': Settings(),
29
           'config': Config(),
30
           'browser': Browser()
3.1
           'review': Review(),
32
           'editor': Editor()
33
```

```
}
34
       app = Control()
36
37
       states[0] = app
38
       states[1] = state_dict
3.9
40
41 loading_screen = LoadingScreen(load_states)
43 def main():
44
45
       Executed by run.py, starts main game loop
46
      app, state_dict = states
47
       if platform == 'win32':
49
           win_setup.set_win_resize_func(app.update_window)
50
51
       app.setup_states(state_dict, 'menu')
52
       app.main_game_loop()
1 import pygame
3 pygame.mixer.init()
4 pygame.init()
6 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MAJOR_VERSION, 3)
\label{eq:context_minor_version} \texttt{7} \;\; \texttt{pygame.display.gl\_set\_attribute(pygame.GL\_CONTEXT\_MINOR\_VERSION\,, \,\, 3)}
{\tt 8} \  \  \, {\tt pygame.display.gl\_set\_attribute(pygame.GL\_CONTEXT\_PROFILE\_MASK, pygame.} \\
       GL_CONTEXT_PROFILE_CORE)
9 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_FORWARD_COMPATIBLE_FLAG, True)
1 import win32gui
2 import win32con
3 import os
4 import ctypes
5 import sys
7 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)
12
13
       return win32gui.CallWindowProc(oldWndProc, hWnd, message, wParam, 1Param)
14
1.5
16 def set_win_resize_func(resize_function):
       oldWndProc = win32gui.SetWindowLong(win32gui.GetForegroundWindow(), win32con.
17
       GWL_WNDPROC, lambda *args: wndProc(oldWndProc, resize_function, *args))
user32 = ctypes.windll.user32
20 user32.SetProcessDPIAware() # To deal with Windows High Text Size / Low Display
       Resolution Settings
22 if os.name != 'nt' or sys.getwindowsversion()[0] < 6:
       raise NotImplementedError("Incompatible OS!")
```

```
1 {
       "primaryBoardColour": "0xB98766",
2
       "secondaryBoardColour": "0xF3D8B8",
       "laserColourBlue": "0x0000ff",
       "laserColourRed": "0xff0000",
5
       "displayMode": "windowed",
6
       "musicVolume": 0.5,
       "sfxVolume": 0.5,
       "particles": true,
9
       "opengl": true,
10
       "shader": "default"
11
12 }
1 {
       "version": 1,
2
       "disable_existing_loggers": false,
       "formatters": {
         "simple": {
5
          "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
           "datefmt": "%Y - %m - %d %H: %M: %S"
        }
9
10
      "handlers": {
11
        "console": {
    "class": "logging.StreamHandler",
12
13
           "formatter": "simple",
14
           "stream": "ext://sys.stdout"
15
        }
16
      },
17
18
      "root": {
19
        "level": "INFO",
20
         "handlers": ["console"],
"propagate": false
21
22
     }
23
    }
24
1 {
       "version": 1,
       "disable_existing_loggers": false,
       "formatters": {
         "simple": {
           "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
6
      },
8
9
10
       "handlers": {
        "console": {
11
           "class": "logging.StreamHandler",
"level": "DEBUG",
12
13
           "formatter": "simple",
14
           "stream": "ext://sys.stdout"
15
16
         "info_file_handler": {
           "class": "logging.handlers.RotatingFileHandler",
"level": "INFO",
19
20
           "formatter": "simple",
21
           "filename": "info.log",
22
           "maxBytes": 10485760,
23
           "backupCount": 20,
24
```

```
"encoding": "utf8"
25
26
27
         "error_file_handler": {
           "class": "logging.handlers.RotatingFileHandler",
"level": "ERROR",
29
3.0
           "formatter": "simple",
"filename": "errors.log",
31
32
           "maxBytes": 10485760,
33
           "backupCount": 20,
34
           "encoding": "utf8"
3.5
36
      },
37
38
39
       "loggers": {
         "my_module": \{
40
           "level": "ERROR",
41
           "handlers": ["console"],
42
           "propagate": false
43
44
45
46
      "root": {
         "level": "INFO",
48
         "handlers": ["console", "info_file_handler", "error_file_handler"]
49
50
    }
51
1 {
       "colours": {
           "text": {
                "primary": "0xdaf2e9",
                "secondary": "0xf14e52",
                "error": "0xf14e52"
6
            "fill": {
                "primary": "0x1c2638",
9
                "secondary": "0xf14e52",
"tertiary": "0xdaf2e9",
"error": "0x9b222b"
10
11
12
13
            "border": {
14
                "primary": "0x9b222b",
15
                "secondary": ""
16
17
      "dimensions": {
19
           "borderRadius": 3,
20
           "borderWidth": 5,
21
           "margin": 10
22
23
24 }
1 {
       "primaryBoardColour": "0xB98766",
       "secondaryBoardColour": "0xF3D8B8",
       "laserColourBlue": "0x0000ff",
       "laserColourRed": "0xff0000",
       "displayMode": "windowed",
       "musicVolume": 0.085,
       "sfxVolume": 0.336,
       "particles": true,
```

```
"opengl": true,
"shader": "default"
10
12 }
1 class Node:
     def __init__(self, data):
           self.data = data
          self.next = None
          self.previous = None
7 class CircularLinkedList:
      def __init__(self, list_to_convert=None):
           Initialises a CircularLinkedList object.
10
11
12
          Args:
              list_to_convert (list, optional): Creates a linked list from existing
1.3
      items. Defaults to None.
14
          self._head = None
15
17
           if list_to_convert:
18
               for item in list_to_convert:
                   self.insert_at_end(item)
19
2.0
      def __str__(self):
21
22
           Returns a string representation of the circular linked list.
23
24
           Returns:
25
           str: Linked list formatted as string.
26
27
           if self._head is None:
28
29
              return '| empty | '
30
          characters = ' | -> '
31
           current_node = self._head
          while True:
33
               characters += str(current_node.data) + ' -> '
34
               current_node = current_node.next
36
37
               if current_node == self._head:
                   characters += '|'
38
                   return characters
39
40
      def insert_at_beginning(self, data):
41
42
          Inserts a node at the beginning of the circular linked list.
43
44
45
          data: The data to insert.
46
47
48
          new_node = Node(data)
49
           if self._head is None:
5.0
              self._head = new_node
               new_node.next = self._head
52
53
               new_node.previous = self._head
54
           else:
               new_node.next = self._head
5.5
56
               new_node.previous = self._head.previous
               self._head.previous.next = new_node
57
```

```
self._head.previous = new_node
58
59
                self._head = new_node
60
       def insert_at_end(self, data):
62
63
           Inserts a node at the end of the circular linked list.
64
6.5
66
           data: The data to insert.
67
68
69
           new_node = Node(data)
70
           if self._head is None:
71
                self._head = new_node
72
                new_node.next = self._head
7.3
               new_node.previous = self._head
74
75
           else:
               new_node.next = self._head
76
77
                new_node.previous = self._head.previous
                self._head.previous.next = new_node
78
                self._head.previous = new_node
7.9
80
      def insert_at_index(self, data, index):
8.1
82
           Inserts a node at a specific index in the circular linked list.
83
           The head node is taken as index 0.
84
85
86
           Args:
                data: The data to insert.
87
88
                index (int): The index to insert the data at.
89
90
           Raises:
               ValueError: Index is out of range.
91
92
           if index < 0:</pre>
               raise ValueError('Invalid index! (CircularLinkedList.insert_at_index)'
94
      )
           if index == 0 or self._head is None:
96
97
               self.insert_at_beginning(data)
98
               new_node = Node(data)
99
100
                current_node = self._head
               count = 0
101
102
                while count < index - 1 and current_node.next != self._head:</pre>
                   current_node = current_node.next
104
105
                    count += 1
106
                if count == (index - 1):
107
108
                    new_node.next = current_node.next
                    new_node.previous = current_node
109
                    current_node.next = new_node
                    raise ValueError('Index out of range! (CircularLinkedList.
112
       insert_at_index)')
113
114
       def delete(self, data):
115
           Deletes a node with the specified data from the circular linked list.
116
117
```

```
118
           Args:
               data: The data to delete.
119
120
121
           Raises:
           ValueError: No nodes in the list contain the specified data.
122
123
           if self._head is None:
124
125
               return
126
           current_node = self._head
127
128
129
           while current_node.data != data:
               current_node = current_node.next
130
131
                if current_node == self._head:
132
                    raise ValueError('Data not found in circular linked list! (
133
       CircularLinkedList.delete)')
134
            if self._head.next == self._head:
135
136
               self._head = None
            else:
137
                current_node.previous.next = current_node.next
138
                current_node.next.previous = current_node.previous
139
140
       def data_in_list(self, data):
141
142
            Checks if the specified data is in the circular linked list.
143
144
145
           Args:
               data: The data to check.
146
147
           Returns:
148
           bool: True if the data is in the list, False otherwise.
149
150
           if self._head is None:
151
152
                return False
153
           current_node = self._head
154
            while True:
               if current_node.data == data:
156
157
                    return True
158
                current_node = current_node.next
if current_node == self._head:
159
160
                    return False
161
162
163
       def shift_head(self):
164
            Shifts the head of the circular linked list to the next node.
165
166
            self._head = self._head.next
167
168
169
       def unshift_head(self):
            Shifts the head of the circular linked list to the previous node.
171
172
            self._head = self._head.previous
173
174
       def get_head(self):
175
176
           Gets the head node of the circular linked list.
177
```

```
179
           Returns:
               Node: The head node.
181
           return self._head
 1 import pygame
 3 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()
11
12
       def get_sprite_collision(self, mouse_pos, square_group):
13
            self.rect.center = mouse_pos
14
            sprite = pygame.sprite.spritecollideany(self, square_group)
15
16
1.7
           return sprite
 1 from data.constants import GameEventType, SettingsEventType, ConfigEventType,
       BrowserEventType, EditorEventType
 3 required_args = {
       {\tt GameEventType.BOARD\_CLICK: ['coords'],}
       GameEventType.ROTATE_PIECE: ['rotation_direction'],
       GameEventType.SET_LASER: ['laser_result'],
       GameEventType.UPDATE_PIECES: ['move_notation'],
       GameEventType.TIMER_END: ['active_colour'],
       GameEventType.PIECE_DROP: ['coords', 'piece', 'colour', 'rotation', '
       remove_overlay'],
       SettingsEventType.COLOUR_SLIDER_SLIDE: ['colour'],
       SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK: ['colour'],
11
       SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK: ['colour'],
       SettingsEventType.DROPDOWN_CLICK: ['selected_word'],
13
       SettingsEventType.VOLUME_SLIDER_CLICK: ['volume', 'volume_type'],
14
15
       SettingsEventType.SHADER_PICKER_CLICK: ['data'],
       SettingsEventType.PARTICLES_CLICK: ['toggled'],
16
       SettingsEventType.OPENGL_CLICK: ['toggled'],
       ConfigEventType.TIME_TYPE: ['time'],
18
       ConfigEventType.FEN_STRING_TYPE: ['time'],
19
       ConfigEventType.CPU_DEPTH_CLICK: ['data'],
       ConfigEventType.PVC_CLICK: ['data'],
21
       ConfigEventType.PRESET_CLICK: ['fen_string'],
22
       BrowserEventType.BROWSER_STRIP_CLICK: ['selected_index'],
23
       BrowserEventType.PAGE_CLICK: ['data'],
24
       EditorEventType.PICK_PIECE_CLICK: ['piece', 'active_colour'], EditorEventType.ROTATE_PIECE_CLICK: ['rotation_direction'],
25
26
27 }
29 class CustomEvent():
30
       def __init__(self, type, **kwargs):
           self.__dict__.update(kwargs)
self.type = type
31
32
       {\tt @classmethod}
34
       def create_event(event_cls, event_type, **kwargs):
3.5
```

```
@classmethod Factory method used to instance CustomEvent object, to check
      for required keyword arguments
38
               event_cls (CustomEvent): Reference to own class.
40
               \verb| event_type: The state EventType|.
4.1
42
           Raises:
43
               ValueError: If required keyword argument for passed event type not
44
               ValueError: If keyword argument passed is not required for passed
45
      event type.
46
47
           Returns:
           CustomEvent: Initialised CustomEvent instance.
49
           if event_type in required_args:
50
51
               for required_arg in required_args[event_type]:
52
                   if required_arg not in kwargs:
                       raise ValueError(f"Argument '{required_arg}' required for {
54
      event_type.name} event (GameEvent.create_event)")
               for kwarg in kwargs:
56
                   if kwarg not in required_args[event_type]:
57
                       raise ValueError(f"Argument '{kwarg}' not included in
58
      required_args dictionary for event '{event_type}'! (GameEvent.create_event)")
               return event_cls(event_type, **kwargs)
60
6.1
           else:
               return event cls(event type)
63
1 from data.constants import Colour
2 from data.states.game.components.move import Move
4 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):
9
           return f'''
10
11 < GameEntry > :>
      CPU_ENABLED: {self._game_states['CPU_ENABLED']}
      CPU_DEPTH: {self._game_states['CPU_DEPTH']},
13
      WINNER: {self._game_states['WINNER']},
TIME_ENABLED: {self._game_states['TIME_ENABLED']},
14
      TIME: {self._game_states['TIME']},
16
      NUMBER_OF_PLY: {len(self._game_states['MOVES'])},
17
18
      MOVES: {self.convert_moves(self._game_states['MOVES'])}
      FINAL FEN_STRING: {self._final_fen_string}
19
      START FEN STRING: {self._game_states['START_FEN_STRING']}
20
21 </GameEntry>
22
      def convert_to_row(self):
24
           return (self._game_states['CPU_ENABLED'], self._game_states['CPU_DEPTH'],
25
       self._game_states['WINNER'], self._game_states['TIME_ENABLED'], self.
      _game_states['TIME'], len(self._game_states['MOVES']), self.convert_moves(self
       . \verb| _game_states['MOVES']|), \verb| self._game_states['START_FEN_STRING']|, \verb| self.|
      _final_fen_string)
```

```
def convert_moves(self, moves):
27
           return '|'.join([
28
               f'{round(move['time'][Colour.BLUE], 4)};{round(move['time'][Colour.RED
      ], 4)};{move['move']}'
3.0
               for move in moves
           1)
31
32
33
      @staticmethod
      def parse_moves(move_str):
34
           moves = move_str.split('|')
3.5
           return [
36
37
               {
                    'blue_time': move.split(';')[0],
3.8
                    'red_time': move.split(';')[1],
39
                    'move': Move.instance_from_notation(move.split(';')[2]),
40
                    'unparsed_move': move.split(';')[2],
41
42
               } for move in moves if move != ''
           1
43
44
45 # self.states = {
         'CPU_ENABLED': game_config['CPU_ENABLED'],
46 #
        'CPU_DEPTH': game_config['CPU_DEPTH'],
47 #
48 #
         'AWAITING_CPU': False,
        'WINNER': None,
49 #
50 #
        'PAUSED': False,
        'ACTIVE_COLOUR': Colour.BLUE,
'TIME_ENABLED': game_config['TIME_ENABLED'],
51 #
52 #
53 #
        'TIME': game_config['TIME'],
        'MOVES': []
54 #
55 # }
56
57
58 #
         move_item = {
         'time': {
59 #
             Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
60 #
61 #
             Colour.RED: GAME_WIDGETS['red_timer'].get_time()
         },
62 #
63 #
         'move': move_notation,
64 #
         'laserResult': laser_result
65 # }
1 import pygame
2 from data.managers.window import window
4 class WidgetGroup(pygame.sprite.Group):
      def __init__(self, widget_dict):
           super().__init__()
           for value in widget_dict.values():
               if isinstance(value, list):
9
10
                   for widget in value:
11
                        self.add(widget)
               elif isinstance(value, dict):
12
                    for widget in value.values():
13
                        self.add(widget)
               else:
15
16
                    self.add(value)
17
18
      def handle_resize(self, new_surface_size):
19
           for sprite in self.sprites():
               sprite.set_surface_size(new_surface_size)
20
```

```
21
               sprite.set_image()
               sprite.set_geometry()
22
23
     def process_event(self, event):
          for sprite in self.sprites():
25
26
               widget_event = sprite.process_event(event)
27
               if widget_event:
28
29
                   return widget_event
30
          return None
31
     def draw(self):
33
          sprites = self.sprites()
34
           for spr in sprites:
35
               surface = spr._surface or window.screen
36
               self.spritedict[spr] = surface.blit(spr.image, spr.rect)
37
38
           self.lostsprites = []
          dirty = self.lostsprites
3.9
40
          return dirty
41
42
     def on_widget(self, mouse_pos):
43
          test_sprite = pygame.sprite.Sprite()
44
           test_sprite.rect = pygame.FRect(*mouse_pos, 1, 1)
45
           return pygame.sprite.spritecollideany(test_sprite, self)
1 import sqlite3
2 from pathlib import Path
4 database_path = (Path(__file__).parent / '../database.db').resolve()
6 def upgrade():
      connection = sqlite3.connect(database_path)
      cursor = connection.cursor()
      cursor.execute('''
      ALTER TABLE games ADD COLUMN created_dt TIMESTAMP NOT NULL
10
11
12
13
14
      connection.commit()
      connection.close()
15
16
17 def downgrade():
      connection = sqlite3.connect(database_path)
18
      cursor = connection.cursor()
19
20
      cursor.execute('''
     ALTER TABLE games DROP COLUMN created_dt
21
22
23
24
      connection.commit()
25
26
      connection.close()
28 upgrade()
29 # downgrade()
1 import sqlite3
2 from pathlib import Path
4 database_path = (Path(__file__).parent / '../database.db').resolve()
```

```
6 def upgrade():
      connection = sqlite3.connect(database_path)
      cursor = connection.cursor()
     cursor.execute('''
     ALTER TABLE games ADD COLUMN fen_string TEXT NOT NULL
10
11
1.3
     connection.commit()
14
      connection.close()
15
16
17 def downgrade():
     connection = sqlite3.connect(database_path)
18
      cursor = connection.cursor()
19
20
     cursor.execute('''
2.1
     ALTER TABLE games DROP COLUMN fen_string
22
23
24
      connection.commit()
     connection.close()
26
2.7
28 upgrade()
1 import sqlite3
2 from pathlib import Path
4 database_path = (Path(__file__).parent / '../database.db').resolve()
6 def upgrade():
      connection = sqlite3.connect(database_path)
      cursor = connection.cursor()
      cursor.execute('''
      ALTER TABLE games ADD COLUMN start_fen_string TEXT NOT NULL
10
11
12
13
14
      connection.commit()
      connection.close()
15
16
17 def downgrade():
     connection = sqlite3.connect(database_path)
18
19
     cursor = connection.cursor()
20
     cursor.execute('''
2.1
     ALTER TABLE games DROP COLUMN start_fen_string
23
24
     connection.commit()
25
      connection.close()
26
27
28 upgrade()
29 # downgrade()
1 import sqlite3
2 from pathlib import Path
4 database_path = (Path(__file__).parent / '../database.db').resolve()
6 def upgrade():
      Upgrade function to rename fen_string column.
```

```
9
      connection = sqlite3.connect(database_path)
10
      cursor = connection.cursor()
      cursor.execute('''
      ALTER TABLE games RENAME COLUMN fen_string TO final_fen_string
13
14
15
16
      connection.commit()
17
      connection.close()
18
19
20 def downgrade():
21
      Downgrade function to revert fen_string column renaming.
22
23
      connection = sqlite3.connect(database_path)
24
25
      cursor = connection.cursor()
26
      cursor.execute('''
27
      ALTER TABLE games RENAME COLUMN final_fen_string TO fen_string
29
3.0
      connection.commit()
31
      connection.close()
32
33
34 upgrade()
35 # downgrade()
1 import sqlite3
2 from pathlib import Path
4 database_path = (Path(__file__).parent / '../database.db').resolve()
6 def upgrade():
      Upgrade function to create games table.
      connection = sqlite3.connect(database_path)
10
11
      cursor = connection.cursor()
      cursor.execute('''
13
          CREATE TABLE games (
14
              id INTEGER PRIMARY KEY,
15
               cpu_enabled INTEGER NOT NULL,
16
17
               cpu_depth INTEGER,
               winner INTEGER,
18
               time_enabled INTEGER NOT NULL,
19
               time REAL,
              number_of_ply INTEGER NOT NULL,
21
22
               moves TEXT NOT NULL
23
     111)
24
25
      connection.commit()
26
      connection.close()
27
29 def downgrade():
30
      Downgrade function to revert table creation.
31
32
      connection = sqlite3.connect(database_path)
33
      cursor = connection.cursor()
34
```

```
cursor.execute('''
36
        DROP TABLE games
3.7
      111)
39
      connection.commit()
40
     connection.close()
41
42
43 upgrade()
44 # downgrade()
1 import pygame
2 from data.utils.asset_helpers import scale_and_cache
_{4} FPS = 60
6 class AnimationManager:
      def __init__(self):
           self._current_ms = 0
          self._timers = []
10
      def set_delta_time(self):
11
          self._current_ms = pygame.time.get_ticks()
12
          for timer in self._timers:
14
               start_ms, target_ms, callback = timer
15
16
               if self._current_ms - start_ms >= target_ms:
                   callback()
17
18
                   self._timers.remove(timer)
      def calculate_frame_index(self, start_index, end_index, fps):
20
          ms_per_frame = int(1000 / fps)
21
           return start_index + ((self._current_ms // ms_per_frame) % (end_index -
22
      start_index))
23
24
      def draw_animation(self, screen, animation, position, size, fps=8):
          frame_index = self.calculate_frame_index(0, len(animation), fps)
2.5
26
           scaled_animation = scale_and_cache(animation[frame_index], size)
          screen.blit(scaled_animation, position)
27
28
29
      def draw_image(self, screen, image, position, size):
          scaled_background = scale_and_cache(image, size)
30
           screen.blit(scaled_background, position)
31
32
      def set_timer(self, target_ms, callback):
3.3
           self._timers.append((self._current_ms, target_ms, callback))
35
36 animation = AnimationManager()
1 import pygame
2 from data.utils.data_helpers import get_user_settings
3 from data.managers.logs import initialise_logger
5 logger = initialise_logger(__name__)
6 user_settings = get_user_settings()
8 class AudioManager:
      def __init__(self, num_channels=16):
          pygame.mixer.set_num_channels(num_channels)
10
          self._music_volume = user_settings['musicVolume']
12
          self._sfx_volume = user_settings['sfxVolume']
13
```

```
self._current_song = None
15
          self._current_channels = []
16
     def set_sfx_volume(self, volume):
18
19
          self._sfx_volume = volume
20
          for channel in self._current_channels:
2.1
22
               channel.set_volume(self._sfx_volume)
23
     def set_music_volume(self, volume):
24
25
          self._music_volume = volume
26
          pygame.mixer.music.set_volume(self._music_volume)
2.7
28
      def pause_sfx(self):
29
30
          pygame.mixer.pause()
31
      def unpause_sfx(self):
32
          pygame.mixer.unpause()
33
34
      def stop_sfx(self, fadeout=0):
3.5
          pygame.mixer.fadeout(fadeout)
36
37
      def remove_unused_channels(self):
38
          unused_channels = []
39
          40
41
              if channel.get_busy() is False:
                  unused_channels.append(channel)
42
43
44
          return unused_channels
45
46
     def play_sfx(self, sfx, loop=False):
          unused_channels = self.remove_unused_channels()
47
48
49
          if len(unused_channels) == 0:
              channel = pygame.mixer.find_channel()
50
          else:
5.1
              channel = unused_channels.pop(0)
53
          if channel is None:
54
              logger.warning('No available channel for SFX')
55
              return
56
57
          self._current_channels.append(channel)
58
          channel.set_volume(self._sfx_volume)
5.9
60
61
          if loop:
              channel.play(sfx, loops=-1)
62
63
          else:
              channel.play(sfx)
64
65
66
     def play_music(self, music_path):
          if 'menu' in str(music_path) and 'menu' in str(self._current_song):
6.7
              return
69
          if music_path == self._current_song:
70
              return
71
72
73
          pygame.mixer.music.stop()
          pygame.mixer.music.unload()
74
          pygame.mixer.music.load(music_path)
75
```

```
pygame.mixer.music.set_volume(self._music_volume)
           pygame.mixer.music.play(loops=-1)
7.8
           self._current_song = music_path
80
81 audio = AudioManager()
1 import pygame
_{\rm 2} from data.assets import GRAPHICS
3 from data.constants import CursorMode
5 class CursorManager:
      def __init__(self):
           self._mode = CursorMode.ARROW
           self.set_mode(CursorMode.ARROW)
9
      def set_mode(self, mode):
10
          pygame.mouse.set_visible(True)
11
12
13
          match mode:
              case CursorMode.ARROW:
14
                  pygame.mouse.set_cursor((7, 5), pygame.transform.scale(GRAPHICS['
1.5
      arrow'], (32, 32)))
               case CursorMode.IBEAM:
16
                  pygame.mouse.set_cursor((15, 5), pygame.transform.scale(GRAPHICS['
17
      ibeam'], (32, 32)))
               case CursorMode.OPENHAND:
18
                  pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['
19
      hand_open'], (32, 32)))
               case CursorMode.CLOSEDHAND:
20
                   pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['
21
      hand_closed'], (32, 32)))
              case CursorMode.NO:
22
                   pygame.mouse.set_visible(False)
24
           self._mode = mode
2.5
26
     def get_mode(self):
2.7
28
           return self._mode
29
30 cursor = CursorManager()
1 import logging.config
2 from data.utils.data_helpers import load_json
3 from pathlib import Path
4 import logging
6 config_path = (Path(__file__).parent / '../app_data/logs_config.json').resolve()
7 config = load_json(config_path)
8 logging.config.dictConfig(config)
10 def initialise_logger(file_path):
      return logging.getLogger(Path(file_path).name)
1 from pathlib import Path
2 from array import array
3 import moderngl
4 from data.shaders.classes import shader_pass_lookup
5 from data.shaders.protocol import SMProtocol
6 from data.constants import ShaderType
```

```
8 shader_path = (Path(__file__).parent / '../shaders/').resolve()
10 SHADER_PRIORITY = [
      {\tt ShaderType.CRT}\;,
      ShaderType.SHAKE,
12
13
      ShaderType.BLOOM,
      ShaderType.CHROMATIC_ABBREVIATION,
14
      ShaderType.RAYS,
1.5
      ShaderType.GRAYSCALE,
16
      ShaderType.BASE,
17
18
20 pygame_quad_array = array('f', [
      -1.0 , 1.0 , 0.0 , 0.0 ,
21
      1.0, 1.0, 1.0, 0.0,
22
      -1.0, -1.0, 0.0, 1.0,
23
      1.0, -1.0, 1.0, 1.0,
24
25 ])
26
27 opengl_quad_array = array('f', [
      -1.0, -1.0, 0.0, 0.0,
1.0, -1.0, 1.0, 0.0,
28
29
      -1.0, 1.0, 0.0, 1.0,
30
      1.0, 1.0, 1.0, 1.0,
3.1
32 ])
33
34 class ShaderManager(SMProtocol):
35
      def __init__(self, ctx: moderngl.Context, screen_size):
           self._ctx = ctx
36
           self._ctx.gc_mode = 'auto'
3.7
38
           self._screen_size = screen_size
3.9
40
           self._opengl_buffer = self._ctx.buffer(data=opengl_quad_array)
           self._pygame_buffer = self._ctx.buffer(data=pygame_quad_array)
41
           self._shader_list = [ShaderType.BASE]
42
           self._vert_shaders = {}
44
           self._frag_shaders = {}
45
           self._programs = {}
           self._vaos = {}
47
           self._textures = {}
48
           self._shader_passes = {}
49
           self.framebuffers = {}
5.0
51
           self.load_shader(ShaderType.BASE)
52
           self.load_shader(ShaderType._CALIBRATE)
5.3
           self.create_framebuffer(ShaderType._CALIBRATE)
55
56
      def load_shader(self, shader_type, **kwargs):
57
           Loads a given shader by creating a VAO reading the corresponding .frag
58
      file.
59
           Args:
60
               shader_type (ShaderType): The type of shader to load.
               **kwargs: Additional arguments passed when initialising the fragment
62
      shader class.
           0.00
64
           self._shader_passes[shader_type] = shader_pass_lookup[shader_type](self,
      **kwargs)
           self.create_vao(shader_type)
66
```

```
def clear_shaders(self):
67
68
           Clears the shader list, leaving only the base shader.
6.9
70
           self._shader_list = [ShaderType.BASE]
71
72
73
      def create_vao(self, shader_type):
7.4
           Creates a vertex array object (VAO) for the given shader type.
7.5
76
7.7
           Args:
           shader_type (ShaderType): The type of shader.
78
79
           frag_name = shader_type[1:] if shader_type[0] == '_' else shader_type
80
           vert_path = Path(shader_path / 'vertex/base.vert').resolve()
81
           frag_path = Path(shader_path / f'fragments/{frag_name}.frag').resolve()
82
83
           self._vert_shaders[shader_type] = vert_path.read_text()
84
           self._frag_shaders[shader_type] = frag_path.read_text()
85
           program = self._ctx.program(vertex_shader=self._vert_shaders[shader_type],
87
        fragment_shader=self._frag_shaders[shader_type])
           self._programs[shader_type] = program
89
           if shader_type == ShaderType._CALIBRATE:
90
               self._vaos[shader_type] = self._ctx.vertex_array(self._programs[
91
       shader_type], [(self._pygame_buffer, '2f 2f', 'vert', 'texCoords')])
92
               self._vaos[shader_type] = self._ctx.vertex_array(self._programs[
93
       shader_type], [(self._opengl_buffer, '2f 2f', 'vert', 'texCoords')])
94
       def create_framebuffer(self, shader_type, size=None, filter=moderngl.NEAREST):
9.5
96
           Creates a framebuffer for the given shader type.
97
98
99
           Args:
               shader_type (ShaderType): The type of shader.
100
               size (tuple[int, int], optional): The size of the framebuffer.
101
       Defaults to screen size.
               filter (moderngl.Filter, optional): The texture filter. Defaults to
       NEAREST.
           0.000
           texture_size = size or self._screen_size
104
           texture = self._ctx.texture(size=texture_size, components=4)
           texture.filter = (filter, filter)
106
107
           self._textures[shader_type] = texture
           {\tt self.framebuffers[shader\_type] = self.\_ctx.framebuffer(color\_attachments=[shader\_type]}
       self._textures[shader_type]])
       def render_to_fbo(self, shader_type, texture, output_fbo=None, program_type=
       None, use_image=True, **kwargs):
           Applies the shaders and renders the resultant texture to a framebuffer
       object (FBO).
114
115
           Args:
               shader_type (ShaderType): The type of shader.
116
               texture (moderngl.Texture): The texture to render.
               output_fbo (moderngl.Framebuffer, optional): The output framebuffer.
118
       Defaults to None.
               program_type (ShaderType, optional): The program type. Defaults to
119
```

```
None.
               use_image (bool, optional): Whether to use the image uniform. Defaults
        to True.
               **kwargs: Additional uniforms for the fragment shader.
121
           fbo = output_fbo or self.framebuffers[shader_type]
123
           program = self._programs[program_type] if program_type else self._programs
124
       [shader_type]
           vao= self._vaos[program_type] if program_type else self._vaos[shader_type]
125
126
           fbo.use()
127
128
           texture.use(0)
129
130
           if use_image:
               program['image'] = 0
131
           for uniform, value in kwargs.items():
132
               program[uniform] = value
133
134
           vao.render(mode=moderngl.TRIANGLE_STRIP)
135
136
       def apply_shader(self, shader_type, **kwargs):
137
138
           Applies a shader of the given type and adds it to the list.
140
141
           Args:
               shader_type (ShaderType): The type of shader to apply.
142
143
144
           ValueError: If the shader is already being applied.
145
146
147
           if shader_type in self._shader_list:
               return
148
149
           self.load_shader(shader_type, **kwargs)
           self._shader_list.append(shader_type)
151
152
           # Sort shader list based on the order in SHADER_PRIORITY, so that more
153
       important shaders are applied first
           self._shader_list.sort(key=lambda shader: -SHADER_PRIORITY.index(shader))
154
       def remove_shader(self, shader_type):
156
157
           Removes a shader of the given type from the list.
158
159
160
           Args:
           shader_type (ShaderType): The type of shader to remove.
161
162
           if shader_type in self._shader_list:
163
               self._shader_list.remove(shader_type)
164
       def render_output(self):
166
167
           Renders the final output to the screen.
168
169
           # Render to the screen framebuffer
           self._ctx.screen.use()
172
           # Take the texture of the last framebuffer to be rendered to, and render
173
       that to the screen framebuffer
174
           output_shader_type = self._shader_list[-1]
175
           self.get_fbo_texture(output_shader_type).use(0)
           self._programs[output_shader_type]['image'] = 0
176
```

```
177
           self._vaos[output_shader_type].render(mode=moderngl.TRIANGLE_STRIP)
178
       def get_fbo_texture(self, shader_type):
181
182
           Gets the texture from the specified shader type's FBO.
183
184
           Args:
               shader_type (ShaderType): The type of shader.
185
186
           Returns:
187
           moderngl.Texture: The texture from the FBO.
189
           return self.framebuffers[shader_type].color_attachments[0]
190
191
       def calibrate_pygame_surface(self, pygame_surface):
192
193
194
           Converts the Pygame window surface into an OpenGL texture.
195
196
               pygame_surface (pygame.Surface): The finished Pygame surface.
197
198
199
              moderngl.Texture: The calibrated texture.
200
201
202
           texture = self._ctx.texture(pygame_surface.size, 4)
203
           texture.filter = (moderngl.NEAREST, moderngl.NEAREST)
           texture.swizzle = 'BGRA'
204
           # Take the Pygame surface's pixel array and draw it to the new texture
205
           texture.write(pygame_surface.get_view('1'))
206
207
           # ShaderType._CALIBRATE has a VAO containing the pygame_quad_array
208
       coordinates, as Pygame uses different texture coordinates than ModernGL
           self.render_to_fbo(ShaderType._CALIBRATE, texture)
           return self.get_fbo_texture(ShaderType._CALIBRATE)
210
211
       def draw(self, surface, arguments):
213
           Draws the Pygame surface with shaders applied to the screen.
214
215
216
           Args:
               surface (pygame.Surface): The final Pygame surface.
217
218
               arguments (dict): A dict of { ShaderType: Args } items, containing
      keyword arguments for every fragment shader.
           self._ctx.viewport = (0, 0, *self._screen_size)
220
           texture = self.calibrate_pygame_surface(surface)
221
222
223
           for shader_type in self._shader_list:
               self._shader_passes[shader_type].apply(texture, **arguments.get(
224
       shader_type, {}))
               texture = self.get_fbo_texture(shader_type)
226
           self.render_output()
227
228
       def __del__(self):
229
230
231
           Cleans up ModernGL resources when the ShaderManager object is deleted.
232
           self.cleanup()
234
```

```
def cleanup(self):
235
236
            Cleans up resources used by the {\tt ModernGL}\,.
237
            Probably unnecessary as the 'auto' garbage collection mode is used.
238
239
            self._pygame_buffer.release()
240
            self._opengl_buffer.release()
241
            for program in self._programs:
242
243
                 self._programs[program].release()
            for texture in self._textures:
244
                 self._textures[texture].release()
245
246
            for vao in self._vaos:
                 self._vaos[vao].release()
247
            \begin{array}{ll} \textbf{for} & \textbf{framebuffer} & \textbf{in} & \textbf{self.framebuffers} : \end{array}
248
                 self.framebuffers[framebuffer].release()
249
250
251
       def handle resize(self. new screen size):
252
            Handles resizing of the screen.
253
254
            Args:
            new_screen_size (tuple[int, int]): The new screen size.
256
257
            self._screen_size = new_screen_size
258
259
            # Recreate all framebuffers to prevent scaling issues
260
261
            \begin{array}{lll} \textbf{for} & \textbf{shader\_type} & \textbf{in} & \textbf{self.framebuffers} : \end{array}
262
                 filter = self._textures[shader_type].filter[0]
                 self.create_framebuffer(shader_type, size=self._screen_size, filter=
263
 1 from data.utils.data_helpers import get_themes, get_user_settings
 3 themes = get_themes()
 4 user_settings = get_user_settings()
 6 def flatten_dictionary_generator(dictionary, parent_key=None):
        Recursive depth-first search to yield all items in a dictionary.
 10
        Args:
            dictionary (dict): Dictionary to be iterated through.
            parent_key (str, optional): Prefix added to every key. Defaults to None.
 12
 13
 14
        Yields:
        dict | tuple[str, str]: Another dictionary or key, value pair.
 15
 16
       for key, value in dictionary.items():
 17
            if parent_key:
 18
                new_key = parent_key + key.capitalize()
 19
 20
            else:
                 new_key = key
 21
 22
            if isinstance(value, dict):
 23
                yield from flatten_dictionary(value, new_key).items()
 9.4
            else:
                 yield new_key, value
 26
 27
28 def flatten_dictionary(dictionary, parent_key=''):
29
        return dict(flatten_dictionary_generator(dictionary, parent_key))
30
31 class ThemeManager:
```

```
def __init__(self):
32
           self.__dict__.update(flatten_dictionary(themes['colours']))
33
           \verb|self.__dict__.update(flatten_dictionary(themes['dimensions']))|\\
3.4
       def __getitem__(self, arg):
36
3.7
       Override default class's \_getitem\_\_ dunder method, to make retrieving an instance attribute nicer with [] notation.
38
39
40
           Args:
               arg (str): Attribute name.
41
42
           Raises:
43
               KeyError: Instance does not have requested attribute.
44
45
           Returns:
46
           str | int: Instance attribute.
47
48
           item = self.__dict__.get(arg)
49
           if item is None:
51
               raise KeyError('(ThemeManager.__getitem__) Requested theme item not
5.2
      found: ', arg)
53
           return item
54
56 theme = ThemeManager()
1 import pygame
2 import moderngl
_{\rm 3} from data.constants <code>import</code> <code>ShaderType</code> , <code>SCREEN_SIZE</code> , <code>SHADER_MAP</code>
4 from data.utils.data_helpers import get_user_settings
5 from data.utils.asset_helpers import draw_background
6 from data.managers.shader import ShaderManager
8 user_settings = get_user_settings()
9 is_opengl = user_settings['opengl']
is_fullscreen = user_settings['displayMode'] == 'fullscreen'
12 class WindowManager(pygame.Window):
      def __init__(self, **kwargs):
13
           super().__init__(**kwargs)
14
           self._native_screen = self.get_surface() # Initialise convert format
15
           self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
16
17
           if is opengl:
18
19
                self._ctx = moderngl.create_context()
                self._shader_manager = ShaderManager(self._ctx, screen_size=self.size)
21
                self.shader_arguments = {
22
                    ShaderType.BASE: {},
23
24
                    ShaderType.SHAKE: {},
25
                    ShaderType.BLOOM: {},
                    ShaderType.GRAYSCALE: {},
26
27
                    ShaderType.CRT: {},
                    ShaderType.RAYS: {}
29
30
                if (selected_shader := get_user_settings()['shader']) is not None:
31
                    for shader_type in SHADER_MAP[selected_shader]:
32
33
                         self.set_effect(shader_type)
           else:
34
```

```
{\tt from} \  \, {\tt data.assets} \  \, {\tt import} \  \, {\tt GRAPHICS}
35
               self._background_image = GRAPHICS['temp_background']
36
37
      def set_effect(self, effect, **kwargs):
          if is_opengl:
39
40
               self._shader_manager.apply_shader(effect, **kwargs)
41
      def set_apply_arguments(self, effect, **kwargs):
42
43
           if is_opengl:
               self.shader_arguments[effect] = kwargs
44
45
46
      def clear_apply_arguments(self, effect):
47
           if is_opengl:
               self.shader_arguments[effect] = {}
48
49
      def clear_effect(self, effect):
5.0
51
           if is_opengl:
52
               self._shader_manager.remove_shader(effect)
               self.clear_apply_arguments(effect)
53
      def clear_all_effects(self, clear_arguments=False):
55
           if is_opengl:
56
               self._shader_manager.clear_shaders()
57
5.8
59
               if clear_arguments:
                   for shader_type in self.shader_arguments:
60
61
                        self.shader_arguments[shader_type] = {}
62
      def draw(self):
63
64
          if is_opengl:
65
               self._shader_manager.draw(self.screen, self.shader_arguments)
66
67
               self._native_screen.blit(self.screen, (0, 0))
68
           self.flip()
6.9
70
           if is_opengl:
71
               self.screen.fill((0, 0, 0, 0))
73
               self.screen.fill((0, 0, 0))
74
               draw_background(self.screen, self._background_image)
75
76
     def update(self):
7.7
78
           self.draw()
79
      def handle_resize(self):
80
           self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
81
           if is_opengl:
82
               self._shader_manager.handle_resize(self.size)
83
           else:
               draw_background(self.screen, self._background_image)
8.5
87 window = WindowManager(size=SCREEN_SIZE, resizable=True, opengl=is_opengl,
      fullscreen_desktop=is_fullscreen)
1 import pygame
2 import moderngl
3 from typing import Protocol, Optional
4 from data.constants import ShaderType
6 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: ...
14
       def render_output(self, texture: moderngl.Texture) -> None: ...
       def get_fbo_texture(self, shader_type: ShaderType) -> moderngl.Texture: ...
1.5
       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: ...
18
19
       def handle_resize(self, new_screen_size: tuple[int]) -> None: ...
20
21
       _ctx: moderngl.Context
22
      _screen_size: tuple[int]
       _opengl_buffer: moderngl.Buffer
24
       _pygame_buffer: moderngl.Buffer
2.5
       _shader_stack: list[ShaderType]
26
27
28
       _vert_shaders: dict
      _frag_shaders: dict
29
3.0
      _programs: dict
31
       _vaos: dict
      _textures: dict
32
       _shader_passes: dict
3.3
       framebuffers: dict
1 import pygame
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
5 class Base:
      def __init__(self, shader_manager: SMProtocol):
           self._shader_manager = shader_manager
           self._shader_manager.create_framebuffer(ShaderType.BASE)
9
           self._shader_manager.create_vao(ShaderType.BACKGROUND_WAVES)
10
           self._shader_manager.create_vao(ShaderType.BACKGROUND_BALATRO)
11
           \verb|self._shader_manager.create_vao(ShaderType.BACKGROUND_LASERS)|
12
13
           \verb|self._shader_manager.create_vao(ShaderType.BACKGROUND_GRADIENT)| \\
           self._shader_manager.create_vao(ShaderType.BACKGROUND_NONE)
14
15
      def apply(self, texture, background_type=None):
           base_texture = self._shader_manager.get_fbo_texture(ShaderType.BASE)
17
           match background_type:
                case ShaderType.BACKGROUND_WAVES:
20
                    self._shader_manager.render_to_fbo(
21
                         ShaderType . BASE ,
22
                         texture=base_texture,
                         program_type=ShaderType.BACKGROUND_WAVES,
                         use_image=False,
25
                         time=pygame.time.get_ticks() / 1000
26
27
                    )
                {\tt case \ ShaderType.BACKGROUND\_BALATRO:}
28
                    self._shader_manager.render_to_fbo(
29
                         ShaderType.BASE,
3.0
```

```
texture=base_texture,
31
                       program_type = ShaderType . BACKGROUND_BALATRO ,
32
33
                       use_image=False,
                       time=pygame.time.get_ticks() / 1000,
                       screenSize=base_texture.size
35
                   )
36
               case ShaderType.BACKGROUND_LASERS:
37
                   self._shader_manager.render_to_fbo(
38
39
                       ShaderType.BASE,
40
                       texture=base_texture,
                       program_type=ShaderType.BACKGROUND_LASERS,
41
42
                       use_image=False,
                       time=pygame.time.get_ticks() / 1000,
43
                      screenSize=base_texture.size
44
                   )
45
               case ShaderType.BACKGROUND_GRADIENT:
46
47
                   self._shader_manager.render_to_fbo(
48
                       ShaderType . BASE,
49
                       texture=base_texture,
                       program_type=ShaderType.BACKGROUND_GRADIENT,
50
51
                       use_image=False,
                       time=pygame.time.get_ticks() / 1000,
5.2
                      screenSize=base_texture.size
53
                   )
54
               case None:
55
56
                   self._shader_manager.render_to_fbo(
57
                       {\tt ShaderType.BASE} \ ,
58
                       texture=base_texture,
                       program_type = ShaderType . BACKGROUND_NONE ,
59
                       use_image=False,
60
61
                   )
62
               case _:
                   raise ValueError('(shader.py) Unknown background type:',
63
      background_type)
64
           self._shader_manager.get_fbo_texture(ShaderType.BASE).use(1)
          self._shader_manager.render_to_fbo(ShaderType.BASE, texture, background=1)
66
1 import moderngl
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
5 class _Blend:
      def __init__(self, shader_manager: SMProtocol):
           self._shader_manager = shader_manager
           self._shader_manager.create_framebuffer(ShaderType._BLEND)
9
11
      def apply(self, texture, texture_2, texture_2_pos):
          self._shader_manager._ctx.blend_func = (moderngl.SRC_ALPHA, moderngl.ONE)
12
13
          relative_size = (texture_2.size[0] / texture.size[0], texture_2.size[1] /
14
      texture.size[1])
          opengl_pos = (texture_2_pos[0], 1 - texture_2_pos[1] - relative_size[1])
16
          texture_2.use(1)
17
          self._shader_manager.render_to_fbo(ShaderType._BLEND, texture, image2=1,
18
      image2Pos=opengl_pos, relativeSize=relative_size)
           self._shader_manager._ctx.blend_func = moderngl.DEFAULT_BLENDING
1 from data.shaders.classes.highlight_brightness import _HighlightBrightness
2 from data.shaders.classes.highlight_colour import _HighlightColour
```

```
3 from data.shaders.protocol import SMProtocol
4 from data.shaders.classes.blur import _Blur
5 from data.constants import ShaderType
7 BLOOM_INTENSITY = 0.6
9 class Bloom:
      def __init__(self, shader_manager: SMProtocol):
1.0
11
           self._shader_manager = shader_manager
12
           shader_manager.load_shader(ShaderType._BLUR)
13
           \verb| shader_manager.load_shader(ShaderType._HIGHLIGHT_BRIGHTNESS)| \\
           shader_manager.load_shader(ShaderType._HIGHLIGHT_COLOUR)
15
16
17
           shader_manager.create_framebuffer(ShaderType.BLOOM)
           shader_manager.create_framebuffer(ShaderType._BLUR)
1.8
           \verb| shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_BRIGHTNESS)| \\
19
20
           shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_COLOUR)
21
      def apply(self, texture, highlight_surface=None, highlight_colours=[],
      surface_intensity=BLOOM_INTENSITY, brightness_intensity=BLOOM_INTENSITY,
      colour_intensity=BLOOM_INTENSITY):
           Applies a bloom effect to a given texture.
24
26
27
               {\tt texture \ (moderngl.Texture): \ Texture \ to \ apply \ bloom \ to.}
               highlight_surface (pygame.Surface, optional): Surface to use as the
      highlights. Defaults to None.
               highlight_colours (list[list[int, int, int], ...], optional): Colours
      to use as the highlights. Defaults to [].
               surface_intensity (_type_, optional): Intensity of bloom applied to
3.0
      the highlight surface. Defaults to BLOOM_INTENSITY.
               brightness_intensity (_type_, optional): Intensity of bloom applied to
       the highlight brightness. Defaults to BLOOM_INTENSITY.
               colour_intensity (_type_, optional): Intensity of bloom applied to the
       \label{light} \mbox{highlight colours. Defaults to BLOOM\_INTENSITY.}
33
34
           if highlight_surface:
               # Calibrate Pygame surface and apply blur
35
               glare_texture = self._shader_manager.calibrate_pygame_surface(
36
      highlight_surface)
               _Blur(self._shader_manager).apply(glare_texture)
37
38
               self._shader_manager.get_fbo_texture(ShaderType._BLUR).use(1)
39
               {\tt self.\_shader\_manager.render\_to\_fbo} \ (\, {\tt ShaderType.BLOOM} \, , \ \ {\tt texture} \, , \\
40
      blurredImage=1, intensity=surface_intensity)
41
               \# Set bloom-applied texture as the base texture
42
43
               texture = self._shader_manager.get_fbo_texture(ShaderType.BLOOM)
44
           # Extract bright colours (highlights) from the texture
45
           _HighlightBrightness(self._shader_manager).apply(texture, intensity=
46
      brightness intensity)
           highlight_texture = self._shader_manager.get_fbo_texture(ShaderType.
       _HIGHLIGHT_BRIGHTNESS)
48
           # Use colour as highlights
49
5.0
           for colour in highlight_colours:
               _HighlightColour(self._shader_manager).apply(texture, old_highlight=
51
      highlight_texture, colour=colour, intensity=colour_intensity)
               \verb|highlight_texture| = self._shader_manager.get_fbo_texture| (ShaderType.
52
```

```
_HIGHLIGHT_COLOUR)
53
           # Apply Gaussian blur to highlights
54
          _Blur(self._shader_manager).apply(highlight_texture)
56
          # Add the pixel values for the highlights onto the base texture
5.7
           self._shader_manager.get_fbo_texture(ShaderType._BLUR).use(1)
           \verb|self._shader_manager.render_to_fbo(ShaderType.BLOOM, texture, blurredImage)| \\
59
      =1, intensity=BLOOM_INTENSITY)
1 from data.shaders.protocol import SMProtocol
2 from data.constants import ShaderType
4 BLUR_ITERATIONS = 4
6 class _Blur:
      def __init__(self, shader_manager: SMProtocol):
          self._shader_manager = shader_manager
           shader_manager.create_framebuffer(ShaderType._BLUR)
10
11
           shader_manager.create_framebuffer("blurPing")
          shader_manager.create_framebuffer("blurPong")
13
14
15
      def apply(self, texture):
16
17
          Applies Gaussian blur to a given texture.
18
19
              texture (moderngl.Texture): Texture to blur.
20
21
          self._shader_manager.get_fbo_texture("blurPong").write(texture.read())
22
23
          for _ in range(BLUR_ITERATIONS):
24
25
               # Apply horizontal blur
26
               self._shader_manager.render_to_fbo(
                   {\tt ShaderType.\_BLUR} \ ,
27
28
                   texture=self._shader_manager.get_fbo_texture("blurPong"),
                   output_fbo=self._shader_manager.framebuffers["blurPing"],
29
30
                   passes=5,
                   horizontal = True
31
32
               # Apply vertical blur
33
               self._shader_manager.render_to_fbo(
34
                   ShaderType._BLUR,
3.5
                   texture=self._shader_manager.get_fbo_texture("blurPing"), # Use
36
      horizontal blur result as input texture
                   output_fbo=self._shader_manager.framebuffers["blurPong"],
37
38
                   passes=5,
                   horizontal=False
39
40
               )
41
           self._shader_manager.render_to_fbo(ShaderType._BLUR, self._shader_manager.
42
      get_fbo_texture("blurPong"))
1 import pygame
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
5 CHROMATIC ABBREVIATION INTENSITY = 2.0
7 class ChromaticAbbreviation:
```

```
def __init__(self, shader_manager: SMProtocol):
           self._shader_manager = shader_manager
9
10
           self._shader_manager.create_framebuffer(ShaderType.CHROMATIC_ABBREVIATION)
12
1.3
      def apply(self, texture):
          mouse_pos = (pygame.mouse.get_pos()[0] / texture.size[0], pygame.mouse.
14
      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 from data.constants import ShaderType
{\tiny 2} \quad \textbf{from} \quad \textbf{data.shaders.protocol} \quad \textbf{import} \quad \textbf{SMProtocol}
4 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])
9
          pixel_size = (int(relative_size[0] * texture.size[0]), int(relative_size
1.0
      [1] * texture.size[1]))
          self._shader_manager.create_framebuffer(ShaderType._CROP, size=pixel_size)
12
          self._shader_manager.render_to_fbo(ShaderType._CROP, texture, relativePos=
14
      opengl_pos, relativeSize=relative_size)
1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
4 class CRT:
      def __init__(self, shader_manager: SMProtocol):
          self._shader_manager = shader_manager
           \verb| shader_manager.create_framebuffer(ShaderType.CRT)| \\
10
      def apply(self, texture):
          self._shader_manager.render_to_fbo(ShaderType.CRT, texture)
1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
4 class Grayscale:
     def __init__(self, shader_manager: SMProtocol):
          self._shader_manager = shader_manager
           shader_manager.create_framebuffer(ShaderType.GRAYSCALE)
      def apply(self, texture):
10
           self._shader_manager.render_to_fbo(ShaderType.GRAYSCALE, texture)
1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
4 HIGHLIGHT_THRESHOLD = 0.9
6 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,
13
      texture, threshold=HIGHLIGHT_THRESHOLD, intensity=intensity)
1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
4 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):
10
          old_highlight.use(1)
11
          self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_COLOUR, texture,
12
      highlight=1, colour=colour, threshold=0.1, intensity=intensity)
1 from data.constants import ShaderType
{\tiny 2~ \textbf{from}~ \textbf{data.shaders.protocol}~ \textbf{import}~ \textbf{SMProtocol}}
3 from data.shaders.classes.shadowmap import _Shadowmap
5 LIGHT_RESOLUTION = 256
7 class _Lightmap:
      def __init__(self, shader_manager: SMProtocol):
          self._shader_manager = shader_manager
9
10
          shader_manager.load_shader(ShaderType._SHADOWMAP)
11
12
13
      def apply(self, texture, colour, softShadow, occlusion=None, falloff=0.0,
      clamp = (-180, 180)):
14
          self._shader_manager.create_framebuffer(ShaderType._LIGHTMAP, size=texture
      .size)
          self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)
15
16
           _Shadowmap(self._shader_manager).apply(texture, occlusion)
17
          shadow_map = self._shader_manager.get_fbo_texture(ShaderType._SHADOWMAP)
18
          self._shader_manager.render_to_fbo(ShaderType._LIGHTMAP, shadow_map,
20
      resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
      clamp, softShadow = softShadow)
21
           self._shader_manager._ctx.disable(self._shader_manager._ctx.BLEND)
1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
4 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 from data.shaders.classes.lightmap import _Lightmap
2 from data.shaders.classes.blend import _Blend
{\tt 3} \  \  \, \textbf{from} \  \  \, \textbf{data.shaders.protocol} \  \  \, \textbf{import} \  \  \, \textbf{SMProtocol}
4 from data.shaders.classes.crop import _Crop
5 from data.constants import ShaderType
7 class Rays:
      def __init__(self, shader_manager: SMProtocol, lights):
           self._shader_manager = shader_manager
           self._lights = lights
11
           # Load all necessary shaders
          shader_manager.load_shader(ShaderType._LIGHTMAP)
13
           shader_manager.load_shader(ShaderType._BLEND)
1.4
           shader_manager.load_shader(ShaderType._CROP)
15
           shader_manager.create_framebuffer(ShaderType.RAYS)
16
17
18
      def apply(self, texture, occlusion=None, softShadow=0.3):
19
           Applies the light rays effect to a given texture.
20
21
22
               texture (moderngl.Texture): The texture to apply the effect to.
23
               occlusion (pygame.Surface, optional): A Pygame mask surface to use as
24
      the occlusion texture. Defaults to None.
26
           final_texture = texture
27
           # Iterate through array containing light information
28
           for pos, radius, colour, *args in self._lights:
29
30
               # Topleft of light source square
               light_topleft = (pos[0] - (radius * texture.size[1] / texture.size[0])
3.1
      , pos[1] - radius)
               # Relative size of light compared to texture
32
               relative_size = (radius * 2 * texture.size[1] / texture.size[0],
33
      radius * 2)
34
               # Crop texture to light source diameter, and to position light source
3.5
      at the center
               _Crop(self._shader_manager).apply(texture, relative_pos=light_topleft,
36
       relative_size=relative_size)
               cropped_texture = self._shader_manager.get_fbo_texture(ShaderType.
      CROP)
               if occlusion:
39
                   # Calibrate Pygame mask surface and crop it
40
                   occlusion_texture = self._shader_manager.calibrate_pygame_surface(
      occlusion)
42
                   _Crop(self._shader_manager).apply(occlusion_texture, relative_pos=
      light_topleft , relative_size=relative_size)
                   occlusion_texture = self._shader_manager.get_fbo_texture(
43
      ShaderType._CROP)
               else:
44
                   occlusion texture = None
45
               # Apply lightmap shader, shadowmap and occlusion are included within
47
      the _{\mbox{Lightmap}} class
               _Lightmap(self._shader_manager).apply(cropped_texture, colour,
      softShadow, occlusion_texture, *args)
               light_map = self._shader_manager.get_fbo_texture(ShaderType._LIGHTMAP)
49
50
               # Blend the final result with the original texture
51
```

```
_Blend(self._shader_manager).apply(final_texture, light_map,
      light_topleft)
               final_texture = self._shader_manager.get_fbo_texture(ShaderType._BLEND
53
      )
54
           self._shader_manager.render_to_fbo(ShaderType.RAYS, final_texture)
5.5
1 import moderngl
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMProtocol
4 from data.shaders.classes.occlusion import _Occlusion
6 LIGHT_RESOLUTION = 256
8 class _Shadowmap:
      def __init__(self, shader_manager: SMProtocol):
9
10
           self._shader_manager = shader_manager
11
          shader_manager.load_shader(ShaderType._OCCLUSION)
12
13
     def apply(self, texture, occlusion_texture=None):
14
1.5
          self._shader_manager.create_framebuffer(ShaderType._SHADOWMAP, size=(
      texture.size[0], 1), filter=moderngl.LINEAR)
17
           if occlusion_texture is None:
               _Occlusion(self._shader_manager).apply(texture)
               occlusion_texture = self._shader_manager.get_fbo_texture(ShaderType.
19
      _OCCLUSION)
           self._shader_manager.render_to_fbo(ShaderType._SHADOWMAP,
21
      \verb|occlusion_texture|, | resolution = LIGHT_RESOLUTION||
1 from data.constants import ShaderType
2 from data.shaders.protocol import SMProtocol
3 from random import randint
5 SHAKE_INTENSITY = 3
7 class Shake:
      def __init__(self, shader_manager: SMProtocol):
           self._shader_manager = shader_manager
          self._shader_manager.create_framebuffer(ShaderType.SHAKE)
11
12
      def apply(self, texture, intensity=SHAKE_INTENSITY):
          displacement = (randint(-intensity, intensity) / 1000, randint(-intensity,
14
       intensity) / 1000)
          self._shader_manager.render_to_fbo(ShaderType.SHAKE, texture, displacement
      =displacement)
1 from data.shaders.classes.chromatic_abbreviation import ChromaticAbbreviation
2 from data.shaders.classes.highlight_brightness import _HighlightBrightness
3 from data.shaders.classes.highlight_colour import _HighlightColour
4 from data.shaders.classes.shadowmap import _Shadowmap 5 from data.shaders.classes.occlusion import _Occlusion
6 from data.shaders.classes.grayscale import Grayscale
7 from data.shaders.classes.lightmap import _Lightmap
8 from data.shaders.classes.blend import _Blend
9 from data.shaders.classes.shake import Shake
10 from data.shaders.classes.bloom import Bloom
11 from data.shaders.classes.blur import _Blur
```

```
12 from data.shaders.classes.crop import _Crop
13 from data.shaders.classes.rays import Rays
14 from data.shaders.classes.base import Base
15 from data.shaders.classes.crt import CRT
16 from data.constants import ShaderType
18 shader_pass_lookup = {
      {\tt ShaderType.CHROMATIC\_ABBREVIATION: ChromaticAbbreviation,}
19
20
      ShaderType.GRAYSCALE: Grayscale,
21
      ShaderType.SHAKE: Shake,
      ShaderType.BLOOM: Bloom,
22
23
      ShaderType.BASE: Base,
      ShaderType.RAYS: Rays,
24
      ShaderType.CRT: CRT,
2.5
26
      ShaderType._HIGHLIGHT_BRIGHTNESS: _HighlightBrightness,
27
28
      {\tt ShaderType.\_HIGHLIGHT\_COLOUR: \_HighlightColour,}
      ShaderType._CALIBRATE: lambda *args: None,
29
      ShaderType._OCCLUSION: _Occlusion,
3.0
      {\tt ShaderType.\_SHADOWMAP: \_Shadowmap,}
31
      {\tt ShaderType.\_LIGHTMAP: \_Lightmap} \ ,
32
      ShaderType._BLEND: _Blend,
3.3
      ShaderType._BLUR: _Blur,
34
      ShaderType._CROP: _Crop,
3.5
36 }
1 import pygame
2 import pyperclip
3 from data.constants import BrowserEventType, ShaderType, GAMES_PER_PAGE
4 from data.utils.database_helpers import delete_game, get_ordered_games
5 from data.states.browser.widget_dict import BROWSER_WIDGETS
6 from data.managers.logs import initialise_logger
7 from data.managers.window import window
8 from data.control import _State
9 from data.assets import MUSIC
10 from random import randint
12 logger = initialise_logger(__name__)
14 class Browser(_State):
15
      def __init__(self):
           super().__init__()
16
18
           self._selected_index = None
           self._filter_column = 'number_of_ply'
19
20
           self._filter_ascend = False
           self._games_list = []
           self._page_number = 1
22
23
24
      def cleanup(self):
           super().cleanup()
25
26
           if self._selected_index is not None:
27
               return self._games_list[self._selected_index]
28
           return None
30
      def startup(self, persist=None):
32
           self.refresh_games_list() # BEFORE RESIZE TO FILL WIDGET BEFORE RESIZING
33
34
           super().startup(BROWSER_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
3.5
```

```
self._filter_column = 'number_of_ply'
36
                      self._filter_ascend = False
37
38
                      \verb|window.set_apply_arguments| (ShaderType.BASE, background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType.background_type=ShaderType=ShaderType.background_type=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=ShaderType=Sha
             BACKGROUND_BALATRO)
40
                       BROWSER_WIDGETS['help'].kill()
41
                      BROWSER_WIDGETS['browser_strip'].kill()
42
43
                       self.draw()
44
45
46
             def refresh_games_list(self):
                      column_map = {
47
                                'moves': 'number_of_ply',
48
                                'winner': 'winner',
49
                               'time' 'created_dt'
5.0
                      }
51
52
                      ascend_map = {
53
                                'asc': True,
                                'desc': False
55
56
57
                      filter_column = BROWSER_WIDGETS['filter_column_dropdown'].
5.8
              get_selected_word()
                     filter_ascend = BROWSER_WIDGETS['filter_ascend_dropdown'].
59
              get_selected_word()
                      self._selected_index = None
61
62
63
                       start_row = (self._page_number - 1) * GAMES_PER_PAGE + 1
                      end_row = (self._page_number) * GAMES_PER_PAGE
64
65
                       self._games_list = get_ordered_games(column_map[filter_column], ascend_map
              [filter_ascend], start_row=start_row, end_row=end_row)
66
                       BROWSER_WIDGETS['browser_strip'].initialise_games_list(self._games_list)
67
                       BROWSER_WIDGETS['browser_strip'].set_surface_size(window.size)
68
                       BROWSER_WIDGETS['scroll_area'].set_image()
69
            def get_event(self, event):
71
                      widget_event = self._widget_group.process_event(event)
72
73
                      if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
7.4
75
                                BROWSER_WIDGETS['help'].kill()
76
                      if widget_event is None:
78
                               return
79
80
                      match widget_event.type:
81
                              case BrowserEventType.MENU_CLICK:
                                       self.next = 'menu'
82
                                         self.done = True
83
84
                                case BrowserEventType.BROWSER_STRIP_CLICK:
8.5
                                         self._selected_index = widget_event.selected_index
87
                                {\tt case \ BrowserEventType.COPY\_CLICK:}
88
                                        if self._selected_index is None:
89
                                                return
9.0
                                        logger.info('COPYING TO CLIPBOARD:', self._games_list[self.
91
              _selected_index]['fen_string'])
                                        pyperclip.copy(self._games_list[self._selected_index]['fen_string'
92
```

```
1)
93
                case BrowserEventType.DELETE_CLICK:
94
                    if self._selected_index is None:
                       return
96
                    delete_game(self._games_list[self._selected_index]['id'])
97
                    self.refresh_games_list()
98
99
                case BrowserEventType.REVIEW_CLICK:
                    if self _selected_index is None:
101
                        return
103
                    self.next = 'review'
104
                    self.done = True
                {\tt case \ BrowserEventType.FILTER\_COLUMN\_CLICK:}
                    selected_word = BROWSER_WIDGETS['filter_column_dropdown'].
108
       get_selected_word()
                    if selected_word is None:
                        return
                    self.refresh_games_list()
113
114
                case BrowserEventType.FILTER_ASCEND_CLICK:
                    selected_word = BROWSER_WIDGETS['filter_ascend_dropdown'].
116
       get_selected_word()
117
                    if selected_word is None:
118
                        return
                    self.refresh_games_list()
122
                case BrowserEventType.PAGE_CLICK:
123
                    self._page_number = widget_event.data
124
125
                    self.refresh_games_list()
                case BrowserEventType.HELP_CLICK:
                    self._widget_group.add(BROWSER_WIDGETS['help'])
130
                    self._widget_group.handle_resize(window.size)
131
       def draw(self):
132
133
           self._widget_group.draw()
 1 from data.components.custom_event import CustomEvent
 2 from data.constants import BrowserEventType, GAMES_PER_PAGE
 3 from data.assets import GRAPHICS
 4 from data.widgets import *
 5 from data.utils.database_helpers import get_number_of_games
 7 BROWSER_HEIGHT = 0.6
 9 browser_strip = BrowserStrip(
       relative_position = (0.0, 0.0),
10
       relative_height = BROWSER_HEIGHT,
11
       games_list=[]
12
13 )
14
15 number_of_pages = get_number_of_games() // GAMES_PER_PAGE + 1
17 carousel_widgets = {
```

```
i: Text(
           relative_position=(0, 0),
19
           relative_size=(0.3, 0.1),
20
           text=f"PAGE {i} OF {number_of_pages}",
           fill_colour=(0, 0, 0, 0),
22
23
           fit_vertical=False,
           border_width =0,
24
2.5
       for i in range(1, number_of_pages + 1)
26
27 }
28
29 sort_by_container = Rectangle(
     relative_size=(0.5, 0.1),
30
      relative_position = (0.01, 0.77),
31
       anchor_x='right',
32
       visible = True
3.3
34 )
35
36 buttons_container = Rectangle(
      relative_position=(0, 0.025),
       relative_size=(0.5, 0.1),
38
      scale_mode='height',
3.9
       anchor_x='center'
40
41 )
42
43 top_right_container = Rectangle(
       relative_position = (0, 0),
44
45
       relative_size = (0.15, 0.075),
      fixed_position=(5, 5),
46
       anchor_x = 'right'
47
48
       scale_mode='height'
49 )
50
51 BROWSER_WIDGETS = {
      'help':
52
53
       Icon(
          relative_position = (0, 0),
54
           relative_size = (1.02, 1.02),
5.5
          icon=GRAPHICS['browser_help'],
           anchor_x='center',
57
           anchor_y='center',
58
           border_width=0,
59
           fill_colour=(0, 0, 0, 0)
60
61
      ),
       'default': [
62
           buttons_container,
63
           sort_by_container,
64
           top_right_container,
65
66
           ReactiveIconButton(
67
               parent=top_right_container,
               relative_position = (0, 0),
68
69
               relative_size=(1, 1),
               anchor_x='right',
70
               scale_mode='height',
7.1
               base_icon = GRAPHICS ['home_base'],
               hover_icon = GRAPHICS['home_hover'],
73
               press_icon = GRAPHICS['home_press'],
74
               event = CustomEvent (BrowserEventType.MENU_CLICK)
           ),
7.6
77
           ReactiveIconButton(
               parent=top_right_container,
78
               relative_position = (0, 0),
79
```

```
relative_size=(1, 1),
                scale_mode='height'
81
                base_icon = GRAPHICS['help_base'],
82
                hover_icon = GRAPHICS['help_hover'],
                press_icon=GRAPHICS['help_press'],
84
8.5
                event = CustomEvent (BrowserEventType.HELP_CLICK)
            ),
86
            Reactive I con Button (
87
88
                parent=buttons_container,
                relative_position = (0, 0),
89
                relative_size=(1, 1),
90
91
                scale_mode='height'
                base_icon = GRAPHICS['copy_base'],
92
                hover_icon = GRAPHICS['copy_hover'],
93
                press_icon = GRAPHICS['copy_press'],
94
                event = CustomEvent (BrowserEventType.COPY_CLICK),
9.5
            ),
96
97
            ReactiveIconButton(
                parent=buttons_container,
98
                relative_position = (0, 0),
99
                relative_size=(1, 1),
                scale_mode='height',
101
                anchor_x='center'
                base_icon = GRAPHICS['delete_base'],
                hover_icon = GRAPHICS['delete_hover'],
104
                press_icon=GRAPHICS['delete_press'],
105
                event=CustomEvent(BrowserEventType.DELETE_CLICK),
106
107
            ),
            ReactiveIconButton(
108
                parent=buttons_container,
                relative_position = (0, 0),
                relative_size=(1, 1),
                scale_mode='height',
                anchor_x='right',
113
                base_icon = GRAPHICS['review_base'],
114
                hover_icon = GRAPHICS['review_hover'],
                press_icon = GRAPHICS['review_press'],
                event = CustomEvent(BrowserEventType.REVIEW_CLICK),
            ),
118
            Text(
119
                parent = sort_by_container,
120
                relative_position=(0, 0),
121
                relative_size=(0.3, 1),
123
                fit_vertical=False,
                text='SORT BY:',
124
                border_width =0,
                fill_colour=(0, 0, 0, 0)
128
       'browser_strip':
           browser_strip,
130
       'scroll_area':
131
       ScrollArea(
           relative_position = (0.0, 0.15),
133
            relative_size = (1, BROWSER_HEIGHT),
            vertical=False,
            widget=browser_strip
136
137
       'filter_column_dropdown':
138
       Dropdown (
            parent=sort_by_container,
140
            relative_position = (0.3, 0),
141
```

```
142
                        relative_height = 0.75,
                        anchor_x='right',
143
                        word_list=['time', 'moves', 'winner'],
144
                        fill_colour=(255, 100, 100),
                        event = CustomEvent (BrowserEventType.FILTER_COLUMN_CLICK)
146
147
               ),
               'filter_ascend_dropdown':
148
               Dropdown (
149
150
                        parent=sort_by_container,
151
                        relative_position=(0, 0),
                        relative_height = 0.75,
152
153
                        anchor_x='right',
                        word_list = ['desc', 'asc'],
154
                        fill_colour=(255, 100, 100),
155
                        event = CustomEvent (BrowserEventType.FILTER_ASCEND_CLICK)
156
157
158
               'page_carousel':
               Carousel (
                      relative_position=(0.01, 0.77),
160
                        margin=5,
161
                        widgets_dict=carousel_widgets,
                        {\tt event=CustomEvent(BrowserEventType.PAGE\_CLICK)}\ ,
163
               )
164
165 }
  1 import pygame
  2 from data.constants import ConfigEventType, Colour, ShaderType
  3 from data states config.default_config import default_config
  4 from data.states.config.widget_dict import CONFIG_WIDGETS
  5 from data.managers.logs import initialise_logger
  6 from data.managers.animation import animation
  7 from data.managers.window import window
  8 from data.managers.audio import audio
  9 from data.managers.theme import theme
 10 from data.assets import MUSIC, SFX
 11 from data.control import _State
 12 from random import randint
 14 logger = initialise_logger(__name__)
 16 class Config(_State):
 17
              def __init__(self):
                        super().__init__()
 18
 19
 20
                        self._config = None
                        self._valid_fen = True
 21
 22
                        self._selected_preset = None
             def cleanup(self):
 24
 25
                        super().cleanup()
 26
 27
                        window.clear_apply_arguments(ShaderType.BLOOM)
 28
                        return self._config
 29
 3.0
               def startup(self, persist=None):
                        super().startup(CONFIG_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
 32
                        \verb|window.set_apply_arguments| (ShaderType.BLOOM, highlight_colours=[(pygame.loop)] | (Pygame.loop) | (Pygame
 33
               Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.9)
 34
                        CONFIG_WIDGETS['invalid_fen_string'].kill()
 35
                        CONFIG_WIDGETS['help'].kill()
 36
```

```
3.7
          self._config = default_config
38
39
          if persist:
40
              self._config['FEN_STRING'] = persist
41
42
          self.set_fen_string(self._config['FEN_STRING'])
43
          self.toggle_pvc(self._config['CPU_ENABLED'])
44
45
          self.set_active_colour(self._config['COLOUR'])
46
          CONFIG_WIDGETS['cpu_depth_carousel'].set_to_key(self._config['CPU_DEPTH'])
47
          if self._config['CPU_ENABLED']:
48
              self.create_depth_picker()
49
           else:
5.0
               self.remove_depth_picker()
51
52
          self.draw()
53
54
      def create_depth_picker(self):
5.5
          # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.8))
56
          # CONFIG_WIDGETS['start_button'].set_image()
57
          CONFIG_WIDGETS['cpu_depth_carousel'].set_surface_size(window.size)
5.8
           CONFIG_WIDGETS['cpu_depth_carousel'].set_image()
59
           CONFIG_WIDGETS['cpu_depth_carousel'].set_geometry()
6.0
           self._widget_group.add(CONFIG_WIDGETS['cpu_depth_carousel'])
61
62
63
      def remove_depth_picker(self):
           # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.7))
64
           # CONFIG_WIDGETS['start_button'].set_image()
65
66
67
           CONFIG_WIDGETS['cpu_depth_carousel'].kill()
68
69
      def toggle_pvc(self, pvc_enabled):
70
           if pvc_enabled:
               CONFIG_WIDGETS['pvc_button'].set_locked(True)
7.1
               CONFIG_WIDGETS['pvp_button'].set_locked(False)
72
          else:
73
               CONFIG_WIDGETS['pvp_button'].set_locked(True)
7.4
               CONFIG_WIDGETS['pvc_button'].set_locked(False)
76
          self._config['CPU_ENABLED'] = pvc_enabled
7.7
78
          if self._config['CPU_ENABLED']:
7.9
80
               self.create_depth_picker()
           else:
81
               self.remove_depth_picker()
82
83
      def set_fen_string(self, new_fen_string):
84
          CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
85
86
          self._config['FEN_STRING'] = new_fen_string
87
           self.set_preset_overlay(new_fen_string)
88
89
90
          try:
               CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
91
               CONFIG_WIDGETS['invalid_fen_string'].kill()
92
93
               if new_fen_string[-1].lower() == 'r':
94
                  self.set_active_colour(Colour.RED)
9.5
96
               else:
                   self.set_active_colour(Colour.BLUE)
97
98
```

```
self._valid_fen = True
99
100
            except:
                CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
101
                self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])
104
                window.set_effect(ShaderType.SHAKE)
                animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE)
105
       )
                audio.play_sfx(SFX['error_1'])
107
                audio.play_sfx(SFX['error_2'])
108
                self._valid_fen = False
111
       def get_event(self, event):
112
            widget_event = self._widget_group.process_event(event)
113
114
115
            if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
                CONFIG_WIDGETS['help'].kill()
116
            if widget_event is None:
118
                return
120
           match widget_event.type:
122
                case ConfigEventType.GAME_CLICK:
                   if self._valid_fen:
123
                        self.next = 'game'
124
                         self.done = True
125
126
                {\tt case \ ConfigEventType.MENU\_CLICK:}
128
                    self.next = 'menu'
                    self.done = True
129
130
131
                case ConfigEventType.TIME_CLICK:
                    self._config['TIME_ENABLED'] = not(widget_event.data)
132
                    CONFIG_WIDGETS['timer_button'].set_next_icon()
134
                case ConfigEventType.PVP_CLICK:
135
                    self.toggle_pvc(False)
136
137
                {\tt case \ ConfigEventType.PVC\_CLICK:}
138
                    self.toggle_pvc(True)
139
140
                {\tt case \ ConfigEventType.FEN\_STRING\_TYPE:}
141
                    self.set_fen_string(widget_event.text)
142
143
                case ConfigEventType.TIME_TYPE:
                    if widget_event.text == '':
145
                        self._config['TIME'] = 5
146
147
                        self._config['TIME'] = float(widget_event.text)
148
149
                case ConfigEventType.CPU_DEPTH_CLICK:
150
                    self._config['CPU_DEPTH'] = int(widget_event.data)
151
                case ConfigEventType.PRESET_CLICK:
153
154
                    self.set_fen_string(widget_event.fen_string)
156
                case ConfigEventType.SETUP_CLICK:
157
                    self.next = 'editor'
                    self.done = True
158
```

```
case ConfigEventType.COLOUR_CLICK:
160
161
                                          self.set_active_colour(widget_event.data.get_flipped_colour())
162
                                 {\tt case \ ConfigEventType.HELP\_CLICK:}
                                          self._widget_group.add(CONFIG_WIDGETS['help'])
164
                                          self._widget_group.handle_resize(window.size)
166
               def set_preset_overlay(self, fen_string):
167
168
                        fen_string_widget_map = {
                                 'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
               PdNaFaNa3Sa b': 'preset_1'
                                 \verb|'sc3ncfcncra2|/10/3Pd2pa3|/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3|/10/2|
               RaNaFaNa3Sa b': 'preset_2',
                                 "sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3RaNaPa3Pc/4Fa5" and a constant and a co
               /3PdNaPa3Sa b': 'preset_3'
                       }
173
174
                        if fen_string in fen_string_widget_map:
                                self._selected_preset = CONFIG_WIDGETS[fen_string_widget_map[
175
               fen_string]]
176
                       else:
                                 self._selected_preset = None
177
178
              def set_active_colour(self, colour):
    if self._config['COLOUR'] != colour:
179
180
                                 CONFIG_WIDGETS['to_move_button'].set_next_icon()
181
182
183
                        self._config['COLOUR'] = colour
184
                        if colour == Colour.BLUE:
185
                                 CONFIG_WIDGETS['to_move_text'].set_text('BLUE TO MOVE')
                        elif colour == Colour.RED:
187
                                 CONFIG_WIDGETS['to_move_text'].set_text('RED TO MOVE')
188
189
                         \  \, \textbf{if} \  \  \, \textbf{self.\_valid\_fen}: \\
190
                                 self._config['FEN_STRING'] = self._config['FEN_STRING'][:-1] + colour.
191
              name[0].lower()
                                 CONFIG_WIDGETS['fen_string_input'].set_text(self._config['FEN_STRING'
192
               ])
193
               def draw(self):
194
                        self._widget_group.draw()
196
197
                        if self._selected_preset:
                                pygame.draw.rect(window.screen, theme['borderPrimary'], (*self.
198
                 _selected_preset.position, *self._selected_preset.size), width=i<mark>nt</mark>(theme['
               borderWidth']))
199
200
               def update(self, **kwargs):
201
                        self._widget_group.update()
202
                        super().update(**kwargs)
  1 from data.constants import Colour
  3 default_config = {
               'CPU_ENABLED': False,
               'CPU_DEPTH': 2,
               'FEN_STRING': 'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
               Pa2/2PdNaFaNa3Sa b',
                'TIME_ENABLED': True,
               'TIME': 5,
               'COLOUR': Colour.BLUE,
```

```
10 }
1 import pygame
2 from data.widgets import *
3 from data.states.config.default_config import default_config
4 from data.components.custom_event import CustomEvent
5 from data.constants import ConfigEventType, Colour
6 from data.assets import GRAPHICS
{\tt 7~from~data.utils.asset\_helpers~import~get\_highlighted\_icon}
8 from data.managers.theme import theme
10 def float_validator(num_string):
       try:
           float(num string)
12
13
           return True
14
      except:
           return False
1.5
17 if default_config['CPU_ENABLED']:
      pvp_icons = {False: GRAPHICS['swords'], True: GRAPHICS['swords']}
18
      pvc_icons = {True: GRAPHICS['robot'], False: GRAPHICS['robot']}
      pvc_locked = True
20
      pvp_locked = False
21
     pvp_icons = {True: GRAPHICS['swords'], False: GRAPHICS['swords']}
pvc_icons = {False: GRAPHICS['robot'], True: GRAPHICS['robot']}
23
      pvc_locked = False
25
      pvp_locked = True
26
27
28 if default_config['TIME_ENABLED']:
       time_enabled_icons = {True: GRAPHICS['timer'], False: get_highlighted_icon(
       GRAPHICS['timer'])}
30 else:
       time_enabled_icons = {False: get_highlighted_icon(GRAPHICS['timer']), True:
31
       GRAPHICS['timer']}
if default_config['COLOUR'] == Colour.BLUE:
      colour_icons = {Colour.BLUE: GRAPHICS['pharoah_0_a'], Colour.RED: GRAPHICS['
      pharoah_1_a']}
35 else:
       colour_icons = {Colour.RED: GRAPHICS['pharoah_1_a'], Colour.BLUE: GRAPHICS['
36
       pharoah_0_a']}
38 preview_container = Rectangle(
      relative_position = (-0.15, 0),
      relative_size=(0.65, 0.9),
40
41
      anchor_x = 'center',
       anchor_y = 'center',
43 )
45 config_container = Rectangle(
     relative_position = (0.325, 0),
46
47
      relative\_size=(0.3, 0.9),
      anchor_x='center',
48
       anchor_y = 'center',
49
50 )
51
52 to_move_container = Rectangle(
     parent=config_container,
53
      relative_size=(0.9, 0.15),
5.4
55
      relative_position=(0, 0.1),
      anchor_x='center'
56
```

```
57
59 board_thumbnail = BoardThumbnail(
      parent=preview_container,
       relative_position=(0, 0),
61
62
       relative_width=0.7,
       scale_mode='width',
63
       anchor_x='right',
64
65 )
66
67 top_right_container = Rectangle(
       relative_position = (0, 0),
       relative_size = (0.15, 0.075),
69
       fixed_position=(5, 5),
70
       anchor_x='right',
71
       scale_mode='height'
72
73 )
74
75 CONFIG_WIDGETS = {
       'help':
76
77
       Icon(
           relative_position = (0, 0),
7.8
            relative_size=(1.02, 1.02),
            icon=GRAPHICS['config_help'],
8.0
            anchor_x='center',
81
            anchor_y='center',
82
            border_width=0,
83
            fill_colour=(0, 0, 0, 0)
84
85
       'default': [
86
87
            preview_container,
            config_container,
88
89
            to_move_container,
            top_right_container,
90
            Reactive I con Button (
91
                parent=top_right_container,
                relative_position = (0, 0),
93
                relative_size=(1, 1),
94
                anchor_x='right',
                scale_mode='height'
96
                base_icon = GRAPHICS['home_base'],
97
                hover_icon = GRAPHICS['home_hover'],
98
                press_icon = GRAPHICS['home_press'],
99
                {\tt event=CustomEvent(ConfigEventType.MENU\_CLICK)}
100
101
            Reactive I con Button (
102
                parent=top_right_container,
                relative_position=(0, 0),
104
                relative_size=(1, 1),
                scale_mode='height',
                base_icon=GRAPHICS['help_base'],
107
108
                hover_icon = GRAPHICS['help_hover'],
                press_icon = GRAPHICS['help_press'],
109
                event=CustomEvent(ConfigEventType.HELP_CLICK)
            ),
            TextInput(
112
                parent = config_container,
113
                relative_position = (0.3, 0.3),
114
                relative_size=(0.65, 0.15),
115
                fit_vertical = True,
116
                placeholder = 'TIME CONTROL (DEFAULT 5)',
                default = str(default_config['TIME']),
118
```

```
border_width=5,
                margin=20,
                validator=float_validator,
                {\tt event=CustomEvent(ConfigEventType.TIME\_TYPE)}
            ),
123
124
            Text(
                parent=config_container,
125
                fit_vertical=False,
126
                relative_position=(0.75, 0.3),
127
                relative_size = (0.2, 0.15),
128
                text = 'MINS'
129
130
                border_width=0,
                fill_colour=(0, 0, 0, 0)
131
            ),
132
            TextButton(
                parent=preview_container,
134
135
                relative_position = (0.3, 0),
136
                relative_size = (0.15, 0.15),
                text='CUSTOM'
137
                anchor_y='bottom',
138
                fit_vertical=False,
139
                margin=10,
140
                event = CustomEvent(ConfigEventType.SETUP_CLICK)
141
            )
142
143
       ],
       'board_thumbnail':
144
145
            board_thumbnail,
       'fen_string_input':
146
       TextInput(
147
            parent=preview_container,
148
            relative_position = (0, 0),
            relative_size = (0.55, 0.15),
150
151
            fit_vertical=False,
            placeholder='ENTER FEN STRING',
152
            {\tt default='sc3ncfcncpb2/2pc7/3Pd7/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7}
       Pa2/2PdNaFaNa3Sa b',
            border_width=5,
154
            anchor_y='bottom',
155
            anchor_x='right',
156
            margin=20,
157
            event=CustomEvent(ConfigEventType.FEN_STRING_TYPE)
158
159
       'start_button':
160
161
       TextButton(
           parent=config_container,
            relative_position = (0, 0),
163
            relative_size = (0.9, 0.3),
           anchor_y='bottom',
            anchor_x='center'
166
167
            text='START NEW GAME',
168
            strength=0.1,
            text_colour = theme['textSecondary'],
169
            margin=20,
            fit_vertical=False,
            event = CustomEvent (ConfigEventType.GAME_CLICK)
       ),
173
174
       'timer_button':
       MultipleIconButton(
175
            parent=config_container,
177
            scale_mode='height'
            relative_position = (0.05, 0.3),
178
            relative_size=(0.15, 0.15),
179
```

```
margin=10,
            border_width = 5,
181
182
            border_radius=5,
            icons_dict=time_enabled_icons ,
            event=CustomEvent(ConfigEventType.TIME_CLICK)
184
185
       ),
186
       'pvp_button':
       {\tt MultipleIconButton} \, (
187
188
            parent=config_container,
            relative_position=(-0.225, 0.5),
189
            relative_size=(0.45, 0.15),
190
191
            margin=15,
            anchor_x='center',
192
            icons_dict=pvp_icons,
193
            stretch = False,
194
            event = CustomEvent (ConfigEventType.PVP_CLICK)
195
196
      ),
197
       'pvc_button':
       MultipleIconButton(
198
           parent=config_container,
            relative_position = (0.225, 0.5),
200
            relative_size = (0.45, 0.15),
201
            anchor_x='center',
202
            margin=15,
203
204
            icons_dict=pvc_icons,
205
            stretch=False,
            event = CustomEvent (ConfigEventType.PVC_CLICK)
206
207
       'invalid_fen_string':
208
       Text(
209
210
           parent=board_thumbnail,
            relative_position = (0, 0),
211
212
            relative_size = (0.9, 0.1),
213
            fit_vertical=False,
            anchor_x='center',
214
            anchor_y='center',
215
            text='INVALID FEN STRING!',
216
            margin=10,
            fill_colour=theme['fillError'],
218
            text_colour=theme['textError'],
219
220
       ),
       'preset_1':
221
       {\tt BoardThumbnailButton}\,(
222
223
            parent=preview_container,
            relative_width=0.25,
224
225
            relative_position = (0, 0),
            scale_mode='width',
            fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3
227
       /7Pa2/2PdNaFaNa3Sa b",
228
            event = CustomEvent (ConfigEventType . PRESET_CLICK)
229
230
       'preset_2':
       BoardThumbnailButton(
231
            parent=preview_container,
232
            relative_width = 0.25,
            relative_position=(0, 0.35),
234
235
            scale_mode='width',
            fen_string="sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3
236
       /10/2RaNaFaNa3Sa b",
            event = CustomEvent (ConfigEventType.PRESET_CLICK)
237
238
       'preset_3':
239
```

```
{\tt BoardThumbnailButton(}
240
            parent=preview_container,
241
            relative_width=0.25,
242
            relative_position=(0, 0.7),
243
            scale_mode='width',
244
            fen_string="sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3
245
        RaNaPa3Pc/4Fa5/3PdNaPa3Sa b",
            event = CustomEvent (ConfigEventType . PRESET_CLICK)
246
247
        'to_move_button':
248
        MultipleIconButton(
249
250
            parent=to_move_container,
            scale_mode='height',
251
            relative_position = (0, 0),
252
            relative_size=(1, 1),
253
            icons_dict=colour_icons,
254
            anchor_x='left',
255
256
            event = CustomEvent (ConfigEventType.COLOUR_CLICK)
       ),
257
258
        'to_move_text':
       Text(
259
260
            parent=to_move_container,
            relative_position = (0, 0),
261
            relative_size = (0.75, 1),
262
            fit_vertical=False,
263
            text='TO MOVE',
264
            anchor_x='right'
265
266
267
        'cpu_depth_carousel':
        Carousel (
268
269
            parent=config_container,
            relative_position=(0, 0.65),
270
            {\tt event=CustomEvent(ConfigEventType.CPU_DEPTH\_CLICK)},
271
            anchor_x='center',
272
            border_width=0,
273
274
            fill_colour=(0, 0, 0, 0),
            widgets_dict={
275
                2: Text(
276
                     parent = config_container,
277
                     relative_position=(0, 0)
278
                     relative_size=(0.8, 0.075),
279
                     text="EASY",
280
                     margin=0,
281
282
                     border_width = 0,
                     fill_colour=(0, 0, 0, 0)
283
                ),
284
                 3: Text(
                     parent=config_container,
286
287
                     relative_position = (0, 0),
288
                     relative_size=(0.8, 0.075),
                     text = "MEDIUM",
289
290
                     margin=0,
                     border_width=0,
291
                     fill_colour=(0, 0, 0, 0)
292
                ),
4: Text(
294
                     parent=config_container,
295
                     relative_position = (0, 0),
296
                     relative_size=(0.8, 0.075),
297
                     text="HARD",
298
                     margin=0,
299
                     border_width=0,
300
```

```
fill_colour=(0, 0, 0, 0)
301
                ),
302
           }
303
       )
304
305 }
 1 import pygame
 2 import pyperclip
 3 from data.constants import EditorEventType, Colour, RotationDirection, Piece,
 4 from data.states.game.components.bitboard_collection import BitboardCollection
 5 from data.states.game.components.fen_parser import encode_fen_string
 6 from data.states.game.components.overlay_draw import OverlayDraw
 7 from data.states.game.components.piece_group import PieceGroup
 {\tt 8} \  \  \, \textbf{from} \  \  \, \textbf{data.states.game.components.father} \  \  \, \textbf{import} \  \  \, \textbf{DragAndDrop}
 9 from data.utils.bitboard_helpers import coords_to_bitboard
{\tt 10 \ from \ data.states.editor.widget\_dict \ import \ EDITOR\_WIDGETS}
11 from data.utils.board_helpers import screen_pos_to_coords
12 from data.managers.logs import initialise_logger
13 from data.managers.window import window
14 from data.control import _State
16 logger = initialise_logger(__name__)
18 class Editor(_State):
      def __init__(self):
            super().__init__()
20
21
            self._bitboards = None
22
           self._piece_group = None
23
24
            self._selected_coords = None
            self._selected_tool = None
            self._selected_tool_colour = None
26
27
            self._initial_fen_string = None
28
           self._starting_colour = None
29
30
            self._drag_and_drop = None
            self._overlay_draw = None
31
32
       def cleanup(self):
33
           super() cleanup()
34
35
            self.deselect_tool()
36
37
38
            return encode_fen_string(self._bitboards)
39
40
       def startup(self, persist):
            super().startup(EDITOR_WIDGETS)
41
            EDITOR_WIDGETS['help'].kill()
42
            self._drag_and_drop = DragAndDrop(EDITOR_WIDGETS['chessboard'].position,
44
       EDITOR_WIDGETS['chessboard'].size)
            self._overlay_draw = OverlayDraw(EDITOR_WIDGETS['chessboard'].position,
45
       EDITOR_WIDGETS['chessboard'].size)
            self._bitboards = BitboardCollection(persist['FEN_STRING'])
46
            self._piece_group = PieceGroup()
48
            self._selected_coords = None
49
50
            self._selected_tool = None
5.1
            self._selected_tool_colour = None
            self._initial_fen_string = persist['FEN_STRING']
52
            self._starting_colour = Colour.BLUE
53
```

```
self.refresh_pieces()
55
           self.set_starting_colour(Colour.BLUE if persist['FEN_STRING'][-1].lower()
56
       == 'b' else Colour.RED)
           self.draw()
57
58
59
       @property
       def selected_coords(self):
6.0
61
           return self._selected_coords
62
63
       @selected_coords.setter
64
       def selected_coords(self, new_coords):
           self._overlay_draw.set_selected_coords(new_coords)
65
           self._selected_coords = new_coords
66
67
       def get_event(self, event):
68
           widget_event = self._widget_group.process_event(event)
69
70
           if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
71
72
               EDITOR_WIDGETS['help'].kill()
73
           if event.type == pygame.MOUSEBUTTONDOWN:
7.4
               clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
       chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
               if clicked_coords:
77
78
                    self.selected_coords = clicked_coords
79
                    if self._selected_tool is None:
80
8.1
                        return
82
                    if self._selected_tool == 'MOVE':
83
84
                        self.set_dragged_piece(clicked_coords)
85
                    elif self._selected_tool == 'ERASE':
86
87
                        self.remove_piece()
                    else:
88
                        self.set_piece(self._selected_tool, self._selected_tool_colour
89
       , Rotation.UP)
90
91
                    return
92
           if event.type == pygame.MOUSEBUTTONUP:
93
               clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
94
       chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
9.5
               if self._drag_and_drop.dragged_sprite:
96
                    self.remove_dragged_piece(clicked_coords)
97
98
                    return
99
100
           if widget_event is None:
               if event.type == pygame.MOUSEBUTTONDOWN and self._widget_group.
101
       on_widget(event.pos) is False:
                    self.selected_coords = None
102
104
               return
105
           match widget_event.type:
106
               case None:
107
108
109
               case EditorEventType.MENU_CLICK:
```

```
self.next = 'menu'
111
                    self.done = True
112
                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.
118
       active_colour)
119
120
                {\tt case \ EditorEventType.ROTATE\_PIECE\_CLICK:}
                    self.rotate_piece(widget_event.rotation_direction)
121
                case EditorEventType.EMPTY_CLICK:
                    self._bitboards = BitboardCollection(fen_string='sc9
124
       /10/10/10/10/10/9Sa b')
125
                    self.refresh_pieces()
126
                case EditorEventType.RESET_CLICK:
127
                    self.reset_board()
128
                case EditorEventType.COPY_CLICK:
130
                    logger.info(f'COPYING TO CLIPBOARD: {encode_fen_string(self.
131
       _bitboards)}')
                    pyperclip.copy(encode_fen_string(self._bitboards))
133
                {\tt case} \quad {\tt EditorEventType.BLUE\_START\_CLICK:}
                    self.set_starting_colour(Colour.BLUE)
135
136
137
                case EditorEventType.RED_START_CLICK:
                    self.set_starting_colour(Colour.RED)
138
139
                case EditorEventType.START_CLICK:
140
                    self.next = 'config'
141
                    self.done = True
142
143
                case EditorEventType.CONFIG_CLICK:
144
                    self.reset_board()
145
                    self.next = 'config
146
                    self.done = True
147
148
                case EditorEventType.ERASE_CLICK:
149
150
                    if self._selected_tool == 'ERASE':
                        self.deselect_tool()
151
                    else:
152
                         self.select_tool('ERASE', None)
154
                \verb|case| EditorEventType.MOVE_CLICK| :
155
                    if self._selected_tool == 'MOVE':
                        self.deselect_tool()
157
                    else:
158
                         self.select_tool('MOVE', None)
160
                case EditorEventType.HELP_CLICK:
                    self._widget_group.add(EDITOR_WIDGETS['help'])
163
                    self._widget_group.handle_resize(window.size)
164
165
       def reset_board(self):
            self._bitboards = BitboardCollection(self._initial_fen_string)
166
            self.refresh_pieces()
167
168
```

```
def refresh_pieces(self):
169
           self._piece_group.initialise_pieces(self._bitboards.convert_to_piece_list
       (), EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
171
       def set_starting_colour(self, new_colour):
172
           if new_colour == Colour.BLUE:
173
               EDITOR_WIDGETS['blue_start_button'].set_locked(True)
174
               EDITOR_WIDGETS['red_start_button'].set_locked(False)
175
176
           elif new_colour == Colour.RED:
               EDITOR_WIDGETS['blue_start_button'].set_locked(False)
               EDITOR_WIDGETS['red_start_button'].set_locked(True)
178
           if new_colour != self._starting_colour:
180
               EDITOR_WIDGETS['blue_start_button'].set_next_icon()
181
               EDITOR_WIDGETS['red_start_button'].set_next_icon()
182
183
           self._starting_colour = new_colour
184
           self._bitboards.active_colour = new_colour
185
186
       def set_dragged_piece(self, coords):
187
           bitboard_under_mouse = coords_to_bitboard(coords)
188
           dragged_piece = self._bitboards.get_piece_on(bitboard_under_mouse, Colour.
189
       BLUE) or self._bitboards.get_piece_on(bitboard_under_mouse, Colour.RED)
190
191
           if dragged_piece is None:
192
               return
193
194
           dragged_colour = self._bitboards.get_colour_on(bitboard_under_mouse)
           dragged_rotation = self._bitboards.get_rotation_on(bitboard_under_mouse)
195
196
197
           self._drag_and_drop.set_dragged_piece(dragged_piece, dragged_colour,
       dragged_rotation)
198
           self._overlay_draw.set_hover_limit(False)
199
       def remove_dragged_piece(self, coords):
200
           piece, colour, rotation = self._drag_and_drop.get_dragged_info()
201
202
           if coords and coords != self._selected_coords and piece != Piece.SPHINX:
203
               self.remove_piece()
204
               self.selected_coords = coords
205
206
               self.set_piece(piece, colour, rotation)
               self.selected_coords = None
207
208
209
           self._drag_and_drop.remove_dragged_piece()
           self._overlay_draw.set_hover_limit(True)
210
       def set_piece(self, piece, colour, rotation):
212
           if self.selected_coords is None or self.selected_coords == (0, 7) or self.
213
       selected_coords == (9, 0):
214
215
           self.remove_piece()
216
217
           selected_bitboard = coords_to_bitboard(self.selected_coords)
218
           self._bitboards.set_square(selected_bitboard, piece, colour)
219
           self._bitboards.set_rotation(selected_bitboard, rotation)
220
221
222
           self.refresh_pieces()
223
224
       def remove_piece(self):
           if self.selected_coords is None or self.selected_coords == (0, 7) or self.
       selected_coords == (9, 0):
```

```
return
226
           selected_bitboard = coords_to_bitboard(self.selected_coords)
228
           \verb|self._bitboards.clear_square(selected_bitboard, Colour.BLUE)| \\
229
           self._bitboards.clear_square(selected_bitboard, Colour.RED)
230
231
           self._bitboards.clear_rotation(selected_bitboard)
232
           self.refresh_pieces()
233
234
       def rotate_piece(self, rotation_direction):
           if self.selected_coords is None or self.selected_coords == (0, 7) or self.
236
       selected_coords == (9, 0):
               return
237
238
           selected_bitboard = coords_to_bitboard(self.selected_coords)
239
240
241
           if self._bitboards.get_piece_on(selected_bitboard, Colour.BLUE) is None
       and self._bitboards.get_piece_on(selected_bitboard, Colour.RED) is None:
242
               return
243
           current_rotation = self._bitboards.get_rotation_on(selected_bitboard)
244
245
           if rotation_direction == RotationDirection.CLOCKWISE:
246
               247
       current_rotation.get_clockwise())
248
           elif rotation_direction == RotationDirection.ANTICLOCKWISE:
               \verb|self._bitboards.update_rotation(selected_bitboard, selected_bitboard,)|\\
249
       current_rotation.get_anticlockwise())
250
251
           self.refresh_pieces()
252
       def select_tool(self, piece, colour):
           dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: '
254
       red_piece_buttons' }
           self.deselect_tool()
256
257
           if piece == 'ERASE':
258
               EDITOR_WIDGETS['erase_button'].set_locked(True)
259
               EDITOR_WIDGETS['erase_button'].set_next_icon()
260
           elif piece == 'MOVE':
261
               EDITOR_WIDGETS['move_button'].set_locked(True)
262
               EDITOR_WIDGETS['move_button'].set_next_icon()
263
264
           else:
               EDITOR_WIDGETS [dict_name_map [colour]] [piece].set_locked(True)
265
               EDITOR_WIDGETS[dict_name_map[colour]][piece].set_next_icon()
266
267
           self._selected_tool = piece
268
           self._selected_tool_colour = colour
269
270
271
       def deselect_tool(self):
           dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: '
272
       red_piece_buttons' }
273
           if self._selected_tool:
274
               if self._selected_tool == 'ERASE':
275
                   EDITOR_WIDGETS['erase_button'].set_locked(False)
276
                   EDITOR_WIDGETS['erase_button'].set_next_icon()
277
               elif self._selected_tool == 'MOVE':
278
                   EDITOR_WIDGETS['move_button'].set_locked(False)
279
                   EDITOR_WIDGETS['move_button'].set_next_icon()
280
               else:
281
```

```
EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self.
       _selected_tool].set_locked(False)
                    {\tt EDITOR\_WIDGETS} \ [ {\tt dict\_name\_map} \ [ {\tt self.\_selected\_tool\_colour} ] \ [ {\tt self.}
283
       _selected_tool].set_next_icon()
284
            self._selected_tool = None
285
            self._selected_tool_colour = None
286
287
288
       def handle_resize(self):
            super().handle_resize()
289
            \verb|self._piece_group.handle_resize| (\verb|EDITOR_WIDGETS| ['chessboard'].position|, \\
290
       EDITOR_WIDGETS['chessboard'].size)
           self._drag_and_drop.handle_resize(EDITOR_WIDGETS['chessboard'].position,
291
       EDITOR_WIDGETS['chessboard'].size)
            self._overlay_draw.handle_resize(EDITOR_WIDGETS['chessboard'].position,
       EDITOR_WIDGETS['chessboard'].size)
293
       def draw(self):
294
           self._widget_group.draw()
295
            self._overlay_draw.draw(window.screen)
296
            self._piece_group.draw(window.screen)
297
            self._drag_and_drop.draw(window.screen)
298
 1 from data.constants import Piece, Colour, RotationDirection, EditorEventType,
       BLUE_BUTTON_COLOURS
 2 from data.utils.asset_helpers import get_highlighted_icon
 3 from data.components.custom_event import CustomEvent
 _{\rm 4} from data.assets import GRAPHICS
 5 from data.widgets import *
 7 blue_pieces_container = Rectangle(
       relative_position=(0.25, 0),
       relative_size=(0.13, 0.65),
 9
       scale_mode='height',
10
       anchor_y='center',
11
       anchor_x = 'center'
12
13 )
14
15 red_pieces_container = Rectangle(
      relative_position=(-0.25, 0),
       relative_size=(0.13, 0.65),
17
18
       scale_mode='height',
       anchor_y='center',
19
       anchor_x = 'center'
20
21 )
22
23 bottom_actions_container = Rectangle(
       relative_position = (0, 0.05),
       relative_size=(0.4, 0.1),
25
       anchor_x='center',
26
       anchor_y = 'bottom'
27
28 )
30 top_actions_container = Rectangle(
       relative_position = (0, 0.05),
3.1
       relative_size=(0.3, 0.1),
       anchor_x = 'center',
33
34
       scale_mode='height'
35 )
37 top_right_container = Rectangle(
       relative_position = (0, 0),
```

```
relative_size = (0.15, 0.075),
      fixed_position=(5, 5),
40
      anchor_x='right',
41
      scale_mode='height'
43 )
44
45 EDITOR_WIDGETS = {
       'help':
46
47
      Icon(
          relative_position = (0, 0),
48
           relative_size=(1.02, 1.02),
49
50
           icon=GRAPHICS['editor_help'],
           anchor_x='center',
51
           anchor_y='center',
52
           border_width =0,
53
           fill_colour = (0, 0, 0, 0)
54
55
      ),
56
      'default': [
          red_pieces_container ,
57
           blue_pieces_container,
           bottom_actions_container,
59
          top_actions_container,
60
           top_right_container,
61
           ReactiveIconButton(
62
63
               parent=top_right_container,
               relative_position=(0, 0),
64
               relative_size=(1, 1),
6.5
66
               anchor_x='right',
               scale_mode='height'
67
               base_icon = GRAPHICS['home_base'],
68
               hover_icon = GRAPHICS['home_hover'],
69
               press_icon = GRAPHICS['home_press'],
7.0
               event=CustomEvent(EditorEventType.MENU_CLICK)
71
           ),
72
           Reactive I con Button (
7.3
74
               parent=top_right_container,
               relative_position = (0, 0),
               relative_size=(1, 1),
               scale_mode='height',
7.7
               base_icon = GRAPHICS['help_base'],
78
               hover_icon = GRAPHICS['help_hover'],
               press_icon = GRAPHICS['help_press'],
80
               event = CustomEvent(EditorEventType.HELP_CLICK)
81
82
           ),
           ReactiveIconButton(
83
               parent=bottom_actions_container ,
84
               relative_position = (0.06, 0),
               relative_size=(1, 1),
86
               anchor_x='center'
87
88
               scale_mode='height'
               base_icon = GRAPHICS['clockwise_arrow_base'],
89
               hover_icon = GRAPHICS['clockwise_arrow_hover'],
90
               press_icon = GRAPHICS['clockwise_arrow_press'],
91
               event = CustomEvent (EditorEventType.ROTATE_PIECE_CLICK,
92
      rotation_direction=RotationDirection.CLOCKWISE)
           ),
93
           Reactive I con Button (
94
               parent=bottom_actions_container,
95
               relative_position = (-0.06, 0),
96
97
               relative_size=(1, 1),
               anchor_x='center'
98
               scale_mode='height',
99
```

```
base_icon = GRAPHICS['anticlockwise_arrow_base'],
                hover_icon = GRAPHICS['anticlockwise_arrow_hover'],
101
                press_icon=GRAPHICS['anticlockwise_arrow_press'],
102
                event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
       rotation_direction=RotationDirection.ANTICLOCKWISE)
104
            ),
            ReactiveIconButton(
                parent=top_actions_container,
107
                relative_position = (0, 0),
108
                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),
114
            ),
116
            ReactiveIconButton(
                parent = top_actions_container,
                relative_position = (0, 0),
                relative_size=(1, 1),
                scale_mode='height'
                base_icon = GRAPHICS['delete_base'],
121
                hover_icon = GRAPHICS['delete_hover'],
                press_icon = GRAPHICS['delete_press'],
                event = CustomEvent(EditorEventType.EMPTY_CLICK),
            ),
126
            ReactiveIconButton(
                parent=top_actions_container,
                relative_position = (0, 0),
128
                relative_size=(1, 1),
                scale_mode='height',
130
                anchor_x='center'
131
                base_icon = GRAPHICS['discard_arrow_base'],
                hover_icon = GRAPHICS['discard_arrow_hover'],
                press_icon = GRAPHICS['discard_arrow_press'],
                event = CustomEvent (EditorEventType . RESET_CLICK),
            ).
136
            ReactiveIconButton(
                relative_position = (0, 0),
138
                fixed_position = (10, 0),
                relative_size=(0.1, 0.1),
140
                anchor_x='right',
141
                anchor_y='center',
142
                scale_mode='height'
143
                base_icon=GRAPHICS['play_arrow_base'],
144
                hover_icon=GRAPHICS['play_arrow_hover'],
press_icon=GRAPHICS['play_arrow_press'],
145
146
                event = CustomEvent(EditorEventType.START_CLICK),
147
148
            ),
            Reactive I con Button (
149
                relative_position = (0, 0),
150
                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'],
158
                 event = CustomEvent (EditorEventType.CONFIG_CLICK),
            )
       ٦.
160
```

```
161
       'blue_piece_buttons': {},
162
       'red_piece_buttons': {},
       'erase_button':
       MultipleIconButton(
           parent=red_pieces_container,
            relative_position=(0, 0),
166
           relative_size=(0.2, 0.2),
167
           scale_mode='height',
168
169
           margin=10,
           icons_dict={True: GRAPHICS['eraser'], False: get_highlighted_icon(GRAPHICS
170
       ['eraser'])},
171
           event = CustomEvent (EditorEventType . ERASE_CLICK),
173
       'move_button':
       MultipleIconButton(
174
           parent=blue_pieces_container,
175
176
            relative_position = (0, 0),
            relative_size = (0.2, 0.2),
           scale_mode='height'
178
           box_colours=BLUE_BUTTON_COLOURS,
           icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
180
       ['finger'])},
            event = CustomEvent (EditorEventType.MOVE_CLICK),
182
183
       'chessboard':
184
       Chessboard(
185
           relative_position=(0, 0),
186
            relative_width=0.4,
           scale_mode='width',
187
           anchor_x='center',
188
            anchor_y='center'
190
191
       'blue_start_button':
       MultipleIconButton(
192
           parent=bottom_actions_container,
193
           relative_position = (0, 0),
194
           relative_size=(1, 1),
           scale_mode='height',
196
            anchor_x='right',
197
           box_colours = BLUE_BUTTON_COLOURS,
198
           icons_dict={False: get_highlighted_icon(GRAPHICS['pharoah_0_a']), True:
       GRAPHICS['pharoah_0_a']},
           {\tt event=CustomEvent(EditorEventType.BLUE\_START\_CLICK)}
200
201
       'red_start_button':
202
       MultipleIconButton(
           parent=bottom_actions_container,
           relative_position=(0, 0),
205
206
           relative_size=(1, 1),
207
           scale_mode='height',
            icons_dict={True: GRAPHICS['pharoah_1_a'], False: get_highlighted_icon(
208
       GRAPHICS['pharoah_1_a'])},
           event = CustomEvent (EditorEventType.RED_START_CLICK)
209
211 }
212
213 for index, piece in enumerate([piece for piece in Piece if piece != Piece.SPHINX])
       blue_icon = GRAPHICS[f'{piece.name.lower()}_0_a']
214
215
       dimmed_blue_icon = get_highlighted_icon(blue_icon)
216
       EDITOR_WIDGETS['blue_piece_buttons'][piece] = MultipleIconButton(
217
```

```
218
           parent=blue_pieces_container,
           relative_position=(0, (index + 1) / 5),
219
           relative_size = (0.2, 0.2),
           scale_mode='height',
221
           box_colours = BLUE_BUTTON_COLOURS,
222
           icons_dict={True: blue_icon, False: dimmed_blue_icon},
223
           event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
224
       active_colour=Colour.BLUE)
225
226
       red_icon = GRAPHICS[f'{piece.name.lower()}_1_a']
227
228
       dimmed_red_icon = get_highlighted_icon(red_icon)
229
230
       EDITOR_WIDGETS['red_piece_buttons'][piece] = MultipleIconButton(
231
           parent=red_pieces_container,
232
           relative_position=(0, (index + 1) / 5),
233
234
           relative_size = (0.2, 0.2),
           scale_mode='height',
235
           icons_dict={True: red_icon, False: dimmed_red_icon},
236
           event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
237
       active_colour=Colour.RED)
 1 import pygame
 2 from functools import partial
 3 from data.states.game.mvc.game_controller import GameController
 4 from data.utils.database_helpers import insert_into_games
 5 from data.states.game.mvc.game_model import GameModel
 6 from data.states.game.mvc.pause_view import PauseView
 7 from data.states.game.mvc.game_view import GameView
 8 from data.states.game.mvc.win_view import WinView
 9 from data.components.game_entry import GameEntry
10 from data.managers.logs import initialise_logger
11 from data.managers.window import window
12 from data.managers.audio import audio
13 from data.constants import ShaderType
14 from data.assets import MUSIC, SFX
15 from data.control import _State
17 logger = initialise_logger(__name__)
19 class Game(_State):
      def __init__(self):
2.0
21
           super().__init__()
22
23
       def cleanup(self):
           super().cleanup()
25
           window.clear_apply_arguments(ShaderType.BLOOM)
26
           window.clear_effect(ShaderType.RAYS)
27
28
           game_entry = GameEntry(self.model.states, final_fen_string=self.model.
29
       get_fen_string())
3.0
           inserted_game = insert_into_games(game_entry.convert_to_row())
           return inserted_game
32
33
34
       def switch_to_menu(self):
3.5
           self.next = 'menu'
           self.done = True
36
37
```

```
def switch_to_review(self):
           self.next = 'review'
39
           self.done = True
40
      def startup(self, persist):
42
      music = MUSIC[['cpu_easy', 'cpu_medium', 'cpu_hard'][persist['CPU_DEPTH']
- 2]] if persist['CPU_ENABLED'] else MUSIC['pvp']
43
           super().startup(music=music)
44
45
           window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
46
      BACKGROUND_LASERS)
          window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
47
      Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
           binded_startup = partial(self.startup, persist)
48
49
           self.model = GameModel(persist)
5.0
           self.view = GameView(self.model)
51
52
           self.pause_view = PauseView(self.model)
           self.win_view = WinView(self.model)
53
           self.controller = GameController(self.model, self.view, self.win_view,
      self.pause_view, self.switch_to_menu, self.switch_to_review, binded_startup)
5.5
           self.view.draw()
56
5.7
           audio.play_sfx(SFX['game_start_1'])
58
           audio.play_sfx(SFX['game_start_2'])
59
6.0
61
      def get_event(self, event):
           self.controller.handle_event(event)
62
63
64
      def handle_resize(self):
          self.view.handle_resize()
65
66
           self.win_view.handle_resize()
           self.pause_view.handle_resize()
67
68
     def draw(self):
69
           self.view.draw()
70
           self.win_view.draw()
7.1
           self.pause_view.draw()
73
      def update(self):
74
           self.controller.check_cpu()
           super().update()
7.6
1 from data.widgets import *
2 from data.components.custom_event import CustomEvent
3 from data.constants import GameEventType, RotationDirection, Colour
4 from data.assets import GRAPHICS
6 right_container = Rectangle(
      relative_position=(0.05, 0),
      relative_size = (0.2, 0.5),
      anchor_y = 'center',
      anchor_x='right',
10
11 )
13 rotate_container = Rectangle(
     relative_position=(0, 0.05),
      relative_size=(0.2, 0.1),
15
      anchor_x='center',
16
      anchor_y = 'bottom',
17
18 )
```

```
20 move_list = MoveList(
       parent=right_container,
21
      relative_position=(0, 0),
      relative_width=1,
23
24
      minimum_height = 300,
      move_list=[]
25
26
27
28 resign_button = TextButton(
     parent=right_container,
29
30
       relative_position = (0, 0),
      relative_size=(0.5, 0.2),
31
      fit_vertical=False,
32
      anchor_y = 'bottom',
33
      text=" Resign",
3.4
      margin=5,
35
36
       event = CustomEvent (GameEventType . RESIGN_CLICK)
37 )
39 draw_button = TextButton(
       parent=right_container,
40
      relative_position=(0, 0),
41
      relative_size=(0.5, 0.2),
42
      fit_vertical=False,
43
      anchor_x='right',
44
      anchor_y='bottom',
45
46
      text=" Draw",
47
      margin=5,
       event = CustomEvent (GameEventType.DRAW_CLICK)
48
49 )
5.0
51 top_right_container = Rectangle(
       relative_position=(0, 0),
52
       relative_size=(0.225, 0.075),
5.3
54
       fixed_position=(5, 5),
       anchor_x = 'right',
55
       scale_mode='height'
56
57 )
58
59 GAME_WIDGETS = {
      'help':
60
61
       Icon(
           relative_position = (0, 0),
62
           relative_size = (1.02, 1.02),
63
          icon=GRAPHICS['game_help'],
64
65
           anchor_x='center',
           anchor_y='center',
66
           border_width=0,
67
68
           fill_colour=(0, 0, 0, 0)
     ),
6.9
      'tutorial':
70
71
      Icon(
          relative_position=(0, 0),
73
           relative_size=(0.9, 0.9),
           icon=GRAPHICS['game_tutorial'],
74
           anchor_x='center',
7.5
           anchor_y='center',
76
7.7
       'default': [
78
          right_container,
79
           rotate_container,
80
```

```
top_right_container,
            ReactiveIconButton(
82
                parent=top_right_container,
83
                relative_position = (0, 0),
                relative_size=(1, 1),
85
                anchor_x='right'
86
                scale_mode='height'
87
                base_icon=GRAPHICS['home_base'],
hover_icon=GRAPHICS['home_hover'],
88
89
                press_icon = GRAPHICS['home_press'],
90
                event = CustomEvent (GameEventType.MENU_CLICK)
91
            ),
            Reactive I con Button (
93
94
                parent=top_right_container,
                relative_position = (0, 0),
95
                relative_size=(1, 1),
96
97
                scale_mode='height',
98
                base_icon = GRAPHICS['tutorial_base'],
                hover_icon = GRAPHICS['tutorial_hover'],
99
                press_icon = GRAPHICS['tutorial_press'],
100
                event = CustomEvent (GameEventType.TUTORIAL_CLICK)
            ),
            ReactiveIconButton(
                parent=top_right_container,
104
                relative_position = (0.33, 0),
                relative_size=(1, 1),
                scale_mode='height'
107
                base_icon = GRAPHICS['help_base'],
108
                hover_icon = GRAPHICS['help_hover'],
                press_icon=GRAPHICS['help_press'],
                event = CustomEvent (GameEventType . HELP_CLICK)
            ),
113
            Reactive I con Button (
114
                parent=rotate_container,
                relative_position = (0, 0),
                relative_size=(1, 1),
                scale_mode='height',
                anchor_x='right',
118
                base_icon = GRAPHICS['clockwise_arrow_base'],
                hover_icon = GRAPHICS['clockwise_arrow_hover'],
                press_icon=GRAPHICS['clockwise_arrow_press'],
121
                event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
       RotationDirection.CLOCKWISE)
123
            ),
            ReactiveIconButton(
                parent=rotate_container,
                relative_position = (0, 0),
                relative_size=(1, 1),
                scale_mode='height'
128
                base_icon = GRAPHICS['anticlockwise_arrow_base'],
                hover_icon = GRAPHICS['anticlockwise_arrow_hover'],
130
                press_icon = GRAPHICS['anticlockwise_arrow_press'],
131
                event = CustomEvent (GameEventType . ROTATE_PIECE, rotation_direction =
       RotationDirection.ANTICLOCKWISE)
           ),
            resign_button,
134
            draw_button,
136
            Icon(
137
                parent=resign_button,
138
                relative_position = (0, 0),
                relative_size = (0.75, 0.75),
139
                fill_colour=(0, 0, 0, 0),
140
```

```
scale_mode='height',
141
                anchor_y='center',
142
                border_radius=0,
143
                border_width =0,
                margin=5,
145
                icon = GRAPHICS['resign']
146
            ),
147
            Icon(
148
                parent=draw_button,
149
                relative_position = (0, 0),
150
                relative_size=(0.75, 0.75),
151
152
                fill_colour=(0, 0, 0, 0),
                scale_mode='height',
153
                anchor_y='center',
154
                border_radius=0,
155
                border_width=0,
156
157
                margin=5,
158
                icon = GRAPHICS ['draw']
            ),
160
       'scroll_area': # REMEMBER SCROLL AREA AFTER CONTAINER FOR RESIZING
161
       ScrollArea(
162
           parent=right_container,
            relative_position = (0, 0),
164
            relative_size=(1, 0.8),
165
            vertical=True,
166
            widget=move_list
167
168
       ),
169
       'move_list':
            move_list,
       'blue_timer':
171
       Timer(
            relative_position = (0.05, 0.05),
173
            anchor_y= center,
174
            relative_size=(0.1, 0.1),
175
176
            active_colour=Colour.BLUE,
            event = CustomEvent (GameEventType.TIMER_END),
       ),
178
179
       'red_timer':
       Timer(
180
           relative_position = (0.05, -0.05),
181
            anchor_y='center',
182
            relative_size=(0.1, 0.1),
183
            active_colour=Colour.RED,
184
            event = CustomEvent (GameEventType.TIMER_END),
185
       ),
186
187
       'status_text':
       Text(
188
           relative_position = (0, 0.05),
189
190
            relative_size=(0.4, 0.1),
            anchor_x='center'
191
192
            fit_vertical=False,
            margin=10,
193
            text="g",
194
            minimum_width = 400
       ),
196
        'chessboard':
197
       Chessboard(
198
           relative_position = (0, 0),
199
200
            anchor_x='center',
            anchor_y='center'
201
            scale_mode='width',
202
```

```
relative_width=0.4
203
204
        'blue_piece_display':
205
206
        PieceDisplay(
           relative_position=(0.05, 0.05),
207
            relative_size=(0.2, 0.1),
208
            anchor_y='bottom',
209
            active_colour=Colour.BLUE
211
        'red_piece_display':
212
        PieceDisplay(
213
214
            relative_position = (0.05, 0.05),
215
            relative_size=(0.2, 0.1),
            active_colour = Colour . RED
216
217
218 }
219
220 PAUSE_WIDGETS = {
      'default': [
221
222
            TextButton(
                relative_position = (0, -0.125),
223
                relative_size = (0.3, 0.2),
224
                anchor_x='center',
225
                anchor_y='center',
226
                 text='GO TO MENU',
227
                fit_vertical=False,
228
                event = CustomEvent (GameEventType.MENU_CLICK)
229
230
            ),
231
            TextButton(
                relative_position=(0, 0.125),
232
233
                 relative_size=(0.3, 0.2),
                anchor_x='center',
234
                anchor_y='center'
235
                text='RESUME GAME',
236
                fit_vertical=False,
237
                event = CustomEvent (GameEventType.PAUSE_CLICK)
238
239
            )
        1
240
241 }
242
243 win_container = Rectangle(
     relative_position=(0, 0),
244
       relative_size=(0.4, 0.8),
245
246
        scale_mode='height',
       anchor_x='center',
247
       anchor_y = 'center',
fill_colour=(128, 128, 128, 200),
248
249
        visible=True
250
251 )
252
253 WIN_WIDGETS = {
        'default': [
254
255
            win_container,
            TextButton(
256
257
                parent=win_container,
                relative_position=(0, 0.5),
258
                relative_size=(0.8, 0.15),
259
                text='GO TO MENU',
260
                anchor_x='center',
261
262
                fit_vertical=False,
                event = CustomEvent (GameEventType.MENU_CLICK)
263
            ),
264
```

```
TextButton(
265
                parent=win_container,
266
                 relative_position = (0, 0.65),
267
268
                 relative_size=(0.8, 0.15),
                 text='REVIEW GAME',
269
                 anchor_x='center',
270
                 fit_vertical=False,
271
                 event = CustomEvent (GameEventType.REVIEW_CLICK)
272
            ),
273
            TextButton(
274
                parent=win_container,
275
276
                 relative_position = (0, 0.8),
277
                relative_size=(0.8, 0.15),
                 text='NEW GAME',
278
279
                 anchor_x='center',
                fit_vertical=False,
280
                 {\tt event=CustomEvent(GameEventType.GAME\_CLICK)}
281
282
            ),
       ],
283
284
        'blue_won':
        Icon(
285
            parent=win_container,
286
            relative_position = (0, 0.05),
            relative_size=(0.8, 0.3), anchor_x='center',
288
289
            border_width=0,
290
291
            margin=0,
            icon=GRAPHICS['blue_won'],
292
            fill_colour=(0, 0, 0, 0),
293
       ),
294
295
        'red_won':
        Icon(
296
297
            parent=win_container,
            relative_position=(0, 0.05),
298
            relative_size=(0.8, 0.3),
299
300
            anchor_x='center',
301
            border_width =0,
            margin=0,
302
            icon = GRAPHICS ['red_won'],
            fill_colour=(0, 0, 0, 0),
304
305
            fit_icon=True,
306
        'draw_won':
307
308
        Icon(
            parent=win_container,
309
            relative_position = (0, 0.05),
310
311
            relative_size=(0.8, 0.3),
            anchor_x='center',
312
            border_width=0,
313
314
            margin=0,
            icon=GRAPHICS['draw_won'],
315
316
            fill_colour=(0, 0, 0, 0),
317
        'by_checkmate':
318
319
        Icon(
           parent=win_container,
320
            relative_position = (0, 0.375),
321
            relative_size=(0.8, 0.1),
322
            anchor_x='center',
323
            border_width =0,
324
            margin=0,
325
            icon = GRAPHICS ['by_checkmate'],
326
```

```
fill_colour=(0, 0, 0, 0),
327
328
       ),
       'by_resignation':
329
       Icon(
330
331
           parent=win_container,
332
           relative_position=(0, 0.375),
           relative_size=(0.8, 0.1),
333
           anchor_x='center',
334
           border_width=0,
335
           margin=0,
336
           icon=GRAPHICS['by_resignation'],
337
           fill_colour=(0, 0, 0, 0),
338
339
       'by_draw':
340
       Icon(
341
           parent=win_container,
342
           relative_position = (0, 0.375),
343
344
           relative_size = (0.8, 0.1),
           anchor_x='center',
345
346
           border_width=0,
           margin=0,
347
           icon=GRAPHICS['by_draw'],
348
           fill_colour=(0, 0, 0, 0),
       ),
350
       'by_timeout':
351
       Icon(
352
353
           parent=win_container,
354
           relative_position = (0, 0.375),
355
           relative_size = (0.8, 0.1),
           anchor_x='center',
356
357
           border_width =0,
           margin=0,
358
           icon = GRAPHICS ['by_timeout'],
359
            fill_colour=(0, 0, 0, 0),
360
       )
361
362 }
 1 from data.constants import Rank, File, Piece, Colour, Rotation, RotationIndex,
       EMPTY_BB
 2 from data.states.game.components.fen_parser import parse_fen_string
 3 from data.states.game.cpu.zobrist_hasher import ZobristHasher
 4 from data.utils import bitboard_helpers as bb_helpers
 5 from data.managers.logs import initialise_logger
 7 logger = initialise_logger(__name__)
 9 class BitboardCollection:
       def __init__(self, fen_string):
           self.piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char:
       EMPTY_BB for char in Piece}]
           self.combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
12
           self.combined_all_bitboard = EMPTY_BB
13
            self.rotation_bitboards = [EMPTY_BB, EMPTY_BB]
           self.active_colour = Colour.BLUE
15
           self._hasher = ZobristHasher()
16
           try:
18
19
                if fen_string:
                    self.piece_bitboards, self.combined_colour_bitboards, self.
20
       combined_all_bitboard, self.rotation_bitboards, self.active_colour =
       parse_fen_string(fen_string)
                    self.initialise_hash()
```

```
except ValueError as error:
               logger.error('Please input a valid FEN string:', error)
23
               raise error
24
      def __str__(self):
26
27
           Returns a string representation of the bitboards.
28
29
30
           Returns:
           str: Bitboards formatted with piece type and colour shown.
31
32
           characters = ''
33
           for rank in reversed(Rank):
34
               for file in File:
3.5
                   bitboard = 1 << (rank * 10 + file)
36
37
38
                   colour = self.get_colour_on(bitboard)
39
                   piece = self.get_piece_on(bitboard, Colour.BLUE) or self.
      get_piece_on(bitboard, Colour.RED)
40
                   if piece is not None:
41
                            characters += f'{piece.upper() if colour == Colour.BLUE
42
      else piece}
                   else:
43
                       characters += '. '
44
45
               characters += | \n \n |
46
47
           return characters
48
49
50
      def get_rotation_string(self):
5.1
52
           Returns a string representation of the board rotations.
53
           Returns:
54
           str: Board formatted with only rotations shown.
55
56
           characters = ''
5.7
          for rank in reversed(Rank):
59
               for file in File:
60
                   mask = 1 << (rank * 10 + file)
61
                   rotation = self.get_rotation_on(mask)
62
63
                   has_piece = bb_helpers.is_occupied(self.combined_all_bitboard,
      mask)
64
                   if has_piece:
65
                       characters += f'{rotation.upper()} '
66
                   else:
67
68
                       characters += '. '
6.9
70
               characters += | \n \n |
71
           return characters
      def initialise_hash(self):
74
7.5
          Initialises the Zobrist hash for the current board state.
76
7.7
78
           for piece in Piece:
               for colour in Colour:
79
                   piece_bitboard = self.get_piece_bitboard(piece, colour)
80
```

```
81
                    for occupied_bitboard in bb_helpers.occupied_squares(
82
       piece_bitboard):
                        self._hasher.apply_piece_hash(occupied_bitboard, piece, colour
84
           for bitboard in bb_helpers.loop_all_squares():
85
                rotation = self.get_rotation_on(bitboard)
86
                \verb|self._hasher.apply_rotation_hash(bitboard, rotation)|\\
87
88
            if self.active_colour == Colour.RED:
89
90
                self._hasher.apply_red_move_hash()
91
       def flip_colour(self):
92
93
           Flips the active colour and updates the Zobrist hash.
94
95
96
           self.active_colour = self.active_colour.get_flipped_colour()
97
            if self.active_colour == Colour.RED:
98
               self._hasher.apply_red_move_hash()
99
100
101
       def update_move(self, src, dest):
102
           Updates the bitboards for a move.
103
104
105
106
                src (int): The bitboard representation of the source square.
               dest (int): The bitboard representation of the destination square.
108
           piece = self.get_piece_on(src, self.active_colour)
111
           self.clear_square(src, Colour.BLUE)
           self.clear_square(dest, Colour.BLUE)
112
           self.clear_square(src, Colour.RED)
113
            self.clear_square(dest, Colour.RED)
114
115
            self.set_square(dest, piece, self.active_colour)
116
117
       def update_rotation(self, src, dest, new_rotation):
118
119
           Updates the rotation bitboards for a move.
120
               src (int): The bitboard representation of the source square.
123
                {\tt dest\ (int):\ The\ bitboard\ representation\ of\ the\ destination\ square}\,.
124
               new_rotation (Rotation): The new rotation.
126
            self.clear_rotation(src)
127
128
            self.set_rotation(dest, new_rotation)
129
       def clear_rotation(self, bitboard):
130
131
           Clears the rotation for a given square.
132
134
           Args:
           bitboard (int): The bitboard representation of the square. \hfill\Box
135
136
137
            old_rotation = self.get_rotation_on(bitboard)
138
           rotation_1, rotation_2 = self.rotation_bitboards
            self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.clear_square(
139
       rotation_1, bitboard)
```

```
self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.clear_square
140
       (rotation_2, bitboard)
141
           self._hasher.apply_rotation_hash(bitboard, old_rotation)
143
       def clear_square(self, bitboard, colour):
144
145
           Clears a square piece and rotation for a given colour.
146
147
148
           Args:
               bitboard (int): The bitboard representation of the square.
149
150
               colour (Colour): The colour to clear.
151
           piece = self.get_piece_on(bitboard, colour)
152
           if piece is None:
154
155
               return
           piece_bitboard = self.get_piece_bitboard(piece, colour)
157
           colour_bitboard = self.combined_colour_bitboards[colour]
158
           all_bitboard = self.combined_all_bitboard
160
           self.piece_bitboards[colour][piece] = bb_helpers.clear_square(
161
       piece_bitboard, bitboard)
           self.combined_colour_bitboards[colour] = bb_helpers.clear_square(
162
       colour_bitboard, bitboard)
163
           self.combined_all_bitboard = bb_helpers.clear_square(all_bitboard,
       bitboard)
164
           self._hasher.apply_piece_hash(bitboard, piece, colour)
165
166
       def set_rotation(self, bitboard, rotation):
167
168
           Sets the rotation for a given square.
171
           Args:
               bitboard (int): The bitboard representation of the square.
172
               rotation (Rotation): The rotation to set.
173
174
           rotation_1, rotation_2 = self.rotation_bitboards
175
176
           self._hasher.apply_rotation_hash(bitboard, rotation)
           match rotation:
178
179
               case Rotation.UP:
                   return
180
               case Rotation.RIGHT:
181
                    self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.
       set_square(rotation_1, bitboard)
                    return
183
184
               case Rotation.DOWN:
                   self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.
185
       set_square(rotation_2, bitboard)
186
                   return
                case Rotation.LEFT:
187
                   self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.
188
       set_square(rotation_1, bitboard)
                    self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.
189
       set_square(rotation_2, bitboard)
190
                   return
191
               case _:
                   raise ValueError('Invalid rotation input (bitboard.py):', rotation
192
       )
```

```
193
       def set_square(self, bitboard, piece, colour):
194
           Sets a piece on a given square.
197
198
           Args:
               bitboard (int): The bitboard representation of the square.
199
               piece (Piece): The piece to set.
200
201
               colour (Colour): The colour of the piece.
202
           piece_bitboard = self.get_piece_bitboard(piece, colour)
203
204
            colour_bitboard = self.combined_colour_bitboards[colour]
           all_bitboard = self.combined_all_bitboard
205
206
           self.piece_bitboards[colour][piece] = bb_helpers.set_square(piece_bitboard
207
       , bitboard)
208
           self.combined_colour_bitboards[colour] = bb_helpers.set_square(
       colour_bitboard, bitboard)
           self.combined_all_bitboard = bb_helpers.set_square(all_bitboard, bitboard)
210
           self._hasher.apply_piece_hash(bitboard, piece, colour)
211
       def get_piece_bitboard(self, piece, colour):
213
214
           Gets the bitboard for a piece type for a given colour.
215
216
217
           Args:
218
               piece (Piece): The piece bitboard to get.
               colour (Colour): The colour of the piece.
219
220
221
           Returns:
              int: The bitboard representation for all squares occupied by that
       piece type.
           return self.piece_bitboards[colour][piece]
224
225
       def get_piece_on(self, target_bitboard, colour):
226
           Gets the piece on a given square for a given colour.
228
230
           Args:
                target_bitboard (int): The bitboard representation of the square.
231
                colour (Colour): The colour of the piece.
232
233
           Returns:
234
               Piece: The piece on the square, or None if square is empty.
235
236
           if not (bb_helpers.is_occupied(self.combined_colour_bitboards[colour],
237
       target_bitboard)):
238
               return None
239
240
           return next(
               (piece for piece in Piece if
241
                   bb_helpers.is_occupied(self.get_piece_bitboard(piece, colour),
242
       target_bitboard)),
               None)
243
244
       def get_rotation_on(self, target_bitboard):
245
246
247
           Gets the rotation on a given square.
248
           Args:
249
```

```
target_bitboard (int): The bitboard representation of the square.
250
251
            Returns:
               Rotation: The rotation on the square.
253
254
            rotationBits = [bb_helpers.is_occupied(self.rotation_bitboards[
255
       {\tt RotationIndex.SECONDBIT]}\,,\,\,\,{\tt target\_bitboard)}\,,\,\,\,{\tt bb\_helpers.is\_occupied(self.)}
       \verb|rotation_bitboards[RotationIndex.FIRSTBIT]|, | target_bitboard)||\\
            match rotationBits:
257
                case [False, False]:
258
                    return Rotation.UP
                case [False, True]:
260
                    return Rotation.RIGHT
261
                case [True, False]:
262
                   return Rotation.DOWN
263
                case [True, True]:
264
265
                    return Rotation.LEFT
266
       def get_colour_on(self, target_bitboard):
267
268
            Gets the colour of the piece on a given square.
269
270
271
            Args:
                target_bitboard (int): The bitboard representation of the square.
272
273
274
            Returns:
            Colour: The colour of the piece on the square.
275
276
            for piece in Piece:
                if self.get_piece_bitboard(piece, Colour.BLUE) & target_bitboard !=
                     return Colour.BLUE
                elif self.get_piece_bitboard(piece, Colour.RED) & target_bitboard !=
280
       {\tt EMPTY\_BB}:
281
                     return Colour.RED
282
       def get_piece_count(self, piece, colour):
283
284
            Gets the count of a given piece type and colour.
285
286
287
            Args:
                piece (Piece): The piece to count.
288
289
                colour (Colour): The colour of the piece.
290
291
            Returns:
            int: The number of that piece of that colour on the board.
293
            return bb_helpers.pop_count(self.get_piece_bitboard(piece, colour))
294
296
       def get_hash(self):
297
            Gets the Zobrist hash of the current board state.
298
299
            int: The Zobrist hash.
301
302
            return self._hasher.hash
303
304
305
       def convert_to_piece_list(self):
306
            Converts all bitboards to a list of pieces.
307
```

```
308
309
           list: Board represented as a 2D list of Piece and Rotation objects.
310
311
           piece_list = []
312
313
           for i in range(80):
314
               if x := self.get_piece_on(1 << i, Colour.BLUE):</pre>
315
316
                   rotation = self.get_rotation_on(1 << i)</pre>
                   piece_list.append((x.upper(), rotation))
317
               elif y := self.get_piece_on(1 << i, Colour.RED):</pre>
318
319
                   rotation = self.get_rotation_on(1 << i)
                   piece_list.append((y, rotation))
320
               else:
321
                   piece_list.append(None)
323
           return piece_list
324
 1 from data.states.game.components.move import Move
 2 from data.states.game.components.laser import Laser
 4 from data.constants import Colour, Piece, Rank, File, MoveType, RotationDirection,
        Miscellaneous, A_FILE_MASK, J_FILE_MASK, ONE_RANK_MASK, EIGHT_RANK_MASK,
       EMPTY_BB
 5 from data.states.game.components.bitboard_collection import BitboardCollection
 6 from data.utils import bitboard_helpers as bb_helpers
 7 from collections import defaultdict
 9 class Board:
      def __init__(self, fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/
10
       pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa b"):
11
           self.bitboards = BitboardCollection(fen_string)
           self.hash_list = [self.bitboards.get_hash()]
13
       def __str__(self):
14
1.5
           Returns a string representation of the board.
17
18
           Returns:
           str: Board formatted as string.
20
           characters = '8 '
21
           pieces = defaultdict(int)
22
23
24
           for rank_idx, rank in enumerate(reversed(Rank)):
               for file_idx, file in enumerate(File):
25
26
                   mask = 1 << (rank * 10 + file)
                   blue_piece = self.bitboards.get_piece_on(mask, Colour.BLUE)
27
                   red_piece = self.bitboards.get_piece_on(mask, Colour.RED)
28
29
30
                   if blue_piece:
3.1
                       pieces[blue_piece.value.upper()] += 1
                       characters += f'{blue_piece.upper()}
32
                   elif red_piece:
33
                       pieces[red_piece.value] += 1
34
                       characters += f'{red_piece}
                   else:
36
                       characters += '.
37
38
           39
40
           characters += str(dict(pieces))
41
```

```
characters += f'\nCURRENT PLAYER TO MOVE: {self.bitboards.active_colour.
42
       name } \n '
           return characters
43
      def get_piece_list(self):
45
46
           Converts the board bitboards to a list of pieces.
47
48
49
           Returns:
           list: List of Pieces.
50
51
52
           return self.bitboards.convert_to_piece_list()
53
      def get_active_colour(self):
54
55
           Gets the active colour.
56
57
58
           Colour: The active colour.
59
60
           return self.bitboards.active_colour
61
62
      def to_hash(self):
63
64
           Gets the hash of the current board state.
65
66
67
           Returns:
           int: A Zobrist hash.
68
69
           return self.bitboards.get_hash()
71
      def check_win(self):
72
73
           Checks for a Pharoah capture or threefold-repetition.
74
7.5
76
           Colour \mid Miscellaneous: The winning colour, or Miscellaneous.DRAW.
77
7.8
           for colour in Colour:
               if self.bitboards.get_piece_bitboard(Piece.PHAROAH, colour) ==
80
      EMPTY_BB:
                   return colour.get_flipped_colour()
81
82
           if self.hash_list.count(self.hash_list[-1]) >= 3:
83
               return Miscellaneous.DRAW
84
8.5
           return None
86
87
       def apply_move(self, move, fire_laser=True, add_hash=False):
88
89
           Applies a move to the board.
9.0
91
92
           Args:
               move (Move): The move to apply.
93
               fire_laser (bool): Whether to fire the laser after the move.
               add_hash (bool): Whether to add the board state hash to the hash list.
95
96
              Laser: The laser trajectory result.
98
99
           piece_symbol = self.bitboards.get_piece_on(move.src, self.bitboards.
100
       active_colour)
```

```
101
102
           if piece_symbol is None:
               raise ValueError('Invalid move - no piece found on source square')
           elif piece_symbol == Piece.SPHINX:
               raise ValueError('Invalid move - sphinx piece is immovable')
105
106
           if move.move_type == MoveType.MOVE:
107
               possible_moves = self.get_valid_squares(move.src)
108
                if bb_helpers.is_occupied(move.dest, possible_moves) is False:
                    raise ValueError('Invalid move - destination square is occupied')
               piece_rotation = self.bitboards.get_rotation_on(move.src)
113
               self.bitboards.update_move(move.src, move.dest)
114
               self.bitboards.update_rotation(move.src, move.dest, piece_rotation)
116
           elif move.move_type == MoveType.ROTATE:
               piece_symbol = self.bitboards.get_piece_on(move.src, self.bitboards.
118
       active_colour)
               piece_rotation = self.bitboards.get_rotation_on(move.src)
119
120
               if move.rotation_direction == RotationDirection.CLOCKWISE:
                   new_rotation = piece_rotation.get_clockwise()
122
               elif move.rotation_direction == RotationDirection.ANTICLOCKWISE:
123
124
                   new_rotation = piece_rotation.get_anticlockwise()
126
               self.bitboards.update_rotation(move.src, move.src, new_rotation)
127
           laser = None
128
           if fire_laser:
130
               laser = self.fire_laser(add_hash)
131
132
           if add hash:
               self.hash_list.append(self.bitboards.get_hash())
134
           self.bitboards.flip_colour()
135
136
           return laser
137
138
       def undo_move(self, move, laser_result):
139
140
           Undoes a move on the board.
141
142
143
               move (Move): The move to undo.
144
145
               laser_result (Laser): The laser trajectory result.
146
           self.bitboards.flip_colour()
147
148
149
           if laser_result.hit_square_bitboard:
               # Get info of destroyed piece, and add it to the board again
150
               src = laser_result.hit_square_bitboard
151
               piece = laser_result.piece_hit
               colour = laser_result.piece_colour
153
               rotation = laser_result.piece_rotation
               self.bitboards.set_square(src, piece, colour)
156
               self.bitboards.clear_rotation(src)
157
               self.bitboards.set_rotation(src, rotation)
158
159
           # Create new Move object that is the inverse of the passed move
160
           if move.move_type == MoveType.MOVE:
161
```

```
reversed_move = Move.instance_from_bitboards(MoveType.MOVE, move.dest,
162
        move.src)
           elif move.move_type == MoveType.ROTATE:
               reversed_move = Move.instance_from_bitboards(MoveType.ROTATE, move.src
       , move.src, move.rotation_direction.get_opposite())
166
            self.apply_move(reversed_move, fire_laser=False)
           self.bitboards.flip_colour()
167
168
169
       def remove_piece(self, square_bitboard):
171
           Removes a piece from a given square.
172
173
           Args:
               square_bitboard (int): The bitboard representation of the square.
174
175
           \verb|self.bitboards.clear_square(square_bitboard, Colour.BLUE)| \\
176
           self.bitboards.clear_square(square_bitboard, Colour.RED)
           self.bitboards.clear_rotation(square_bitboard)
178
179
       def get_valid_squares(self, src_bitboard, colour=None):
180
181
           Gets valid squares for a piece to move to.
182
183
184
               src_bitboard (int): The bitboard representation of the source square.
185
186
               colour (Colour, optional): The active colour of the piece.
187
188
           Returns:
               int: The bitboard representation of valid squares.
189
           target_top_left = (src_bitboard & A_FILE_MASK & EIGHT_RANK_MASK) << 9</pre>
191
192
           target_top_middle = (src_bitboard & EIGHT_RANK_MASK) << 10</pre>
           target_top_right = (src_bitboard & J_FILE_MASK & EIGHT_RANK_MASK) << 11</pre>
           target_middle_right = (src_bitboard & J_FILE_MASK) << 1</pre>
194
195
           target_bottom_right = (src_bitboard & J_FILE_MASK & ONE_RANK_MASK) >> 9
196
           target_bottom_middle = (src_bitboard & ONE_RANK_MASK) >> 10
197
           target_bottom_left = (src_bitboard & A_FILE_MASK & ONE_RANK_MASK)>> 11
           target_middle_left = (src_bitboard & A_FILE_MASK) >> 1
199
200
           possible_moves = target_top_left | target_top_middle | target_top_right |
201
       target_middle_right | target_bottom_right | target_bottom_middle |
       target_bottom_left | target_middle_left
202
           if colour is not None:
                valid_possible_moves = possible_moves & ~self.bitboards.
       combined_colour_bitboards[colour]
205
           else:
206
                valid_possible_moves = possible_moves & ~self.bitboards.
       {\tt combined\_all\_bitboard}
207
           return valid_possible_moves
208
       def get_mobility(self, colour):
211
           Gets all valid squares for a given colour.
212
213
214
           Args:
                colour (Colour): The colour of the pieces.
215
216
           Returns:
217
```

```
int: The bitboard representation of all valid squares.
218
219
           active_pieces = self.get_all_active_pieces(colour)
           possible_moves = 0
221
222
223
           for square in bb_helpers.occupied_squares(active_pieces):
               possible_moves += bb_helpers.pop_count(self.get_valid_squares(square))
224
           return possible_moves
226
227
       def get_all_active_pieces(self, colour=None):
228
229
           Gets all active pieces for the current player.
230
231
           Args:
232
               colour (Colour): Active colour of pieces to retrieve. Defaults to None
234
           Returns:
235
              int: The bitboard representation of all active pieces.
236
237
           if colour is None:
238
               colour = self.bitboards.active_colour
239
240
           active_pieces = self.bitboards.combined_colour_bitboards[colour]
241
           sphinx_bitboard = self.bitboards.get_piece_bitboard(Piece.SPHINX, colour)
242
           return active_pieces ^ sphinx_bitboard
243
244
       def fire_laser(self, remove_hash):
245
246
           Fires the laser and removes hit pieces.
248
249
           Args:
               remove_hash (bool): Whether to clear the hash list if a piece is hit.
250
251
           Returns:
252
           Laser: The result of firing the laser.
253
254
           laser = Laser(self.bitboards)
255
257
           if laser.hit_square_bitboard:
               self.remove_piece(laser.hit_square_bitboard)
258
259
260
                   self.hash_list = [] # Remove all hashes for threefold repetition,
261
       as the position is impossible to be repeated after a piece is removed
           return laser
262
263
264
       def generate_square_moves(self, src):
265
           Generates all valid moves for a piece on a given square.
266
267
268
           Args:
               src (int): The bitboard representation of the source square.
269
270
           Yields:
271
           Move: A valid move for the piece.
272
273
           for dest in bb_helpers.occupied_squares(self.get_valid_squares(src)):
274
275
                yield Move(MoveType.MOVE, src, dest)
276
       def generate_all_moves(self, colour):
277
```

```
278
            Generates all valid moves for a given colour.
280
281
               colour (Colour): The colour of the pieces.
282
283
           Yields:
284
           Move: A valid move for the active colour.
285
286
            sphinx_bitboard = self.bitboards.get_piece_bitboard(Piece.SPHINX, colour)
287
            # Remove source squares for Sphinx pieces, as they cannot be moved
288
            sphinx_masked_bitboard = self.bitboards.combined_colour_bitboards[colour]
       ^ sphinx_bitboard
290
            for square in bb_helpers.occupied_squares(sphinx_masked_bitboard):
291
                # Generate movement moves
292
293
                yield from self.generate_square_moves(square)
294
                # Generate rotational moves
295
                for rotation_direction in RotationDirection:
296
                    yield Move(MoveType.ROTATE, square, rotation_direction=
297
       rotation_direction)
 1 from data.states.game.components.particles_draw import ParticlesDraw
 2 from data.utils.board_helpers import coords_to_screen_pos
 3 from data.constants import Colour, ShaderType
 4 from data.managers.window import window
 {\tt 5} \quad \textbf{from} \quad \textbf{data.managers.animation} \quad \textbf{import} \quad \textbf{animation}
 7 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()
 12
       def add_capture(self, piece, colour, rotation, piece_coords, sphinx_coords,
13
       active_colour, particles=True, shake=True):
           if particles:
14
15
                self._particles_draw.add_captured_piece(
                    piece,
                    colour.
17
                    rotation,
 18
                    coords_to_screen_pos(piece_coords, self._board_position, self.
       _square_size),
20
                    self._square_size
21
                self._particles_draw.add_sparks(
22
23
                    (255, 0, 0) if active_colour == Colour.RED else (0, 0, 255),
24
                    coords_to_screen_pos(sphinx_coords, self._board_position, self.
25
       _square_size)
26
27
            if shake:
28
29
                window.set_effect(ShaderType.SHAKE)
                animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE)
       def draw(self, screen):
32
            self._particles_draw.draw(screen)
33
34
       def update(self):
35
```

```
self._particles_draw.update()
36
      def handle_resize(self, board_position, board_size):
38
          self._board_position = board_position
          self._square_size = board_size[0] / 10
40
1 import pygame
2 from data.constants import CursorMode
3 from data.states.game.components.piece_sprite import PieceSprite
4 from data.managers.cursor import cursor
5 from data.managers.audio import audio
6 from data.assets import SFX
8 DRAG_THRESHOLD = 500
10 class DragAndDrop:
      def __init__(self, board_position, board_size, change_cursor=True):
          self._board_position = board_position
          self._board_size = board_size
13
          self._change_cursor = change_cursor
14
          self._ticks_since_drag = 0
16
17
          self.dragged_sprite = None
     def set_dragged_piece(self, piece, colour, rotation):
19
20
          sprite = PieceSprite(piece=piece, colour=colour, rotation=rotation)
          sprite.set_geometry((0, 0), self._board_size[0] / 10)
21
          sprite.set_image()
22
23
          self.dragged_sprite = sprite
24
25
          self._ticks_since_drag = pygame.time.get_ticks()
26
          if self._change_cursor:
27
              cursor.set_mode(CursorMode.CLOSEDHAND)
28
29
     def remove_dragged_piece(self):
3.0
31
          self.dragged_sprite = None
          time_dragged = pygame.time.get_ticks() - self._ticks_since_drag
32
33
          self._ticks_since_drag = 0
          if self._change_cursor:
35
36
               cursor.set_mode(CursorMode.OPENHAND)
37
          return time_dragged > DRAG_THRESHOLD
38
39
      def get_dragged_info(self):
40
41
          return self.dragged_sprite.type, self.dragged_sprite.colour, self.
      dragged_sprite.rotation
42
      def draw(self, screen):
43
          if self.dragged_sprite is None:
44
45
              return
46
           self.dragged_sprite.rect.center = pygame.mouse.get_pos()
47
48
          screen.blit(self.dragged_sprite.image, self.dragged_sprite.rect.topleft)
     def handle_resize(self, board_position, board_size):
50
51
          if self.dragged_sprite:
              self.dragged_sprite.set_geometry(board_position, board_size[0] / 10)
52
53
          self._board_position = board_position
54
          self._board_size = board_size
55
```

```
1 from data.constants import Colour, RotationIndex, Rotation, Piece, EMPTY_BB
2 from data.utils.bitboard_helpers import occupied_squares, print_bitboard,
      bitboard_to_index
4 def parse_fen_string(fen_string):
      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(' ')
10
11
      rank = 7
12
      file = 0
1.3
14
15
      piece_count = {char.lower(): 0 for char in Piece} | {char.upper(): 0 for char
      in Piece}
     for index, character in enumerate(part_1):
17
          square = rank * 10 + file
1.8
19
          if character.lower() in Piece:
2.0
21
              piece_count[character] += 1
22
              if character.isupper():
                  piece_bitboards[Colour.BLUE][character.lower()] |= 1 << square</pre>
23
24
25
                  piece_bitboards[Colour.RED][character.lower()] |= 1 << square</pre>
26
              rotation = part_1[index + 1]
28
29
              match rotation:
                  case Rotation.UP:
30
                      pass
3.1
                  case Rotation.RIGHT:
                      rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
33
                  case Rotation.DOWN:
3.4
                      rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
                  case Rotation.LEFT:
36
                      rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
37
                      rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
38
3.9
                  case _:
40
                      raise ValueError('Invalid FEN String - piece character not
      followed by rotational character')
41
              file += 1
          elif character in '0123456789':
43
              if character == '1' and fen_string[index + 1] == '0':
44
                  file += 10
45
                  continue
46
47
              file += int(character)
48
          elif character == '/':
49
              rank = rank - 1
              file = 0
51
          elif character in Rotation:
52
53
54
          else:
             raise ValueError ('Invalid FEN String - invalid character found:',
      character)
56
```

```
if piece_count['s'] != 1 or piece_count['S'] != 1:
57
           raise ValueError('Invalid FEN string - invalid number of Sphinx pieces')
58
       # COMMENTED OUT AS NO PHAROAH PIECES IS OKAY IF PARSING FEN STRING FOR
59
       FINISHED GAME BOARD THUMBNAIL
       \begin{tabular}{ll} elif & piece\_count['f'] > 1 & or & piece\_count['F'] > 1: \\ \end{tabular}
60
           raise ValueError('Invalid FEN string - invalid number of Pharoah pieces')
6.1
62
       if part_2 == 'b':
63
           colour = Colour.BLUE
64
       elif part_2 == 'r':
65
           colour = Colour.RED
66
67
       else:
           raise ValueError('Invalid FEN string - invalid active colour')
68
69
       for piece in Piece:
           combined_colour_bitboards[Colour.BLUE] |= piece_bitboards[Colour.BLUE][
7.1
       piece]
72
           combined_colour_bitboards[Colour.RED] |= piece_bitboards[Colour.RED][piece
73
       combined_all_bitboard = combined_colour_bitboards[Colour.BLUE] |
74
       combined_colour_bitboards[Colour.RED]
       return (piece_bitboards, combined_colour_bitboards, combined_all_bitboard,
       rotation_bitboards, colour)
77 def encode_fen_string(bitboard_collection):
       blue_bitboards = bitboard_collection.piece_bitboards[Colour.BLUE]
7.8
79
       red_bitboards = bitboard_collection.piece_bitboards[Colour.RED]
80
       fen_string_list = [''] * 80
8.1
82
       for piece, bitboard in blue_bitboards.items():
83
84
           for individual_bitboard in occupied_squares(bitboard):
                index = bitboard_to_index(individual_bitboard)
85
                rotation = bitboard_collection.get_rotation_on(individual_bitboard)
86
                fen_string_list[index] = piece.upper() + rotation
87
88
      for piece, bitboard in red_bitboards.items():
89
           for individual_bitboard in occupied_squares(bitboard):
90
                index = bitboard_to_index(individual_bitboard)
91
                rotation = bitboard_collection.get_rotation_on(individual_bitboard)
92
               fen_string_list[index] = piece.lower() + rotation
93
94
95
       fen_string = ''
       row_string = ''
96
       empty_count = 0
97
       for index, square in enumerate(fen_string_list):
98
           if square == '':
99
100
                empty_count += 1
101
           else:
               if empty_count > 0:
102
                   row_string += str(empty_count)
103
                    empty_count = 0
104
105
               row_string += square
107
           if index % 10 == 9:
108
109
               if empty_count > 0:
                   fen_string = '/' + row_string + str(empty_count) + fen_string
                else:
111
                   fen_string = '/' + row_string + fen_string
112
113
```

```
row_string = ''
114
               empty_count = 0
116
       fen_string = fen_string[1:]
118
119
       if bitboard_collection.active_colour == Colour.BLUE:
120
           colour = 'b'
       else:
           colour = 'r'
123
      return fen_string + ' ' + colour
124
 1 from data.utils import bitboard_helpers as bb_helpers
 2 from data.constants import Piece, Colour, Rotation, A_FILE_MASK, J_FILE_MASK,
       ONE_RANK_MASK, EIGHT_RANK_MASK, EMPTY_BB
 3 from data.utils.bitboard_helpers import print_bitboard
 5 class Laser:
       def __init__(self, bitboards):
           self._bitboards = bitboards
           \tt 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.
12
       hit_square_bitboard)
       def calculate_trajectory(self):
14
           current_square = self._bitboards.get_piece_bitboard(Piece.SPHINX, self.
15
       _bitboards.active_colour)
           previous_direction = self._bitboards.get_rotation_on(current_square)
16
           trajectory_bitboard = 0b0
17
           trajectory_list = []
18
           square_animation_states = []
19
20
           pieces_on_trajectory = []
21
22
           while current_square:
               current_piece = self._bitboards.get_piece_on(current_square, Colour.
23
       BLUE) or self._bitboards.get_piece_on(current_square, Colour.RED)
               current_rotation = self._bitboards.get_rotation_on(current_square)
24
25
               next_square, direction, piece_hit = self.calculate_next_square(
26
       current_square, current_piece, current_rotation, previous_direction)
27
28
               trajectory_bitboard |= current_square
               trajectory_list.append(bb_helpers.bitboard_to_coords(current_square))
               square_animation_states.append(direction)
30
31
32
               if previous_direction != direction:
33
                    pieces_on_trajectory.append(current_square)
34
               if next_square == EMPTY_BB:
35
36
                   hit_square_bitboard = 0b0
                    if piece_hit:
38
39
                        hit_square_bitboard = current_square
40
41
                    return hit_square_bitboard, piece_hit, list(zip(trajectory_list,
       square_animation_states)), trajectory_bitboard, pieces_on_trajectory
```

```
43
                current_square = next_square
               previous_direction = direction
44
45
       def calculate_next_square(self, square, piece, rotation, previous_direction):
46
           match piece:
47
               case Piece.SPHINX:
48
                   if previous_direction != rotation:
49
                        return EMPTY_BB, previous_direction, None
5.0
51
52
                   next_square = self.next_square_bitboard(square, rotation)
                    return next_square, previous_direction, Piece.SPHINX
53
54
               case Piece.PYRAMID:
55
                   if previous_direction in [rotation, rotation.get_clockwise()]:
5.6
                        return EMPTY_BB, previous_direction, Piece.PYRAMID
57
5.8
                    if previous_direction == rotation.get_anticlockwise():
59
60
                        new_direction = previous_direction.get_clockwise()
                    else:
61
                        new_direction = previous_direction.get_anticlockwise()
63
                   next_square = self.next_square_bitboard(square, new_direction)
64
65
                    return next_square, new_direction, None
66
67
               case Piece.ANUBIS:
68
6.9
                    if previous_direction == rotation.get_clockwise().get_clockwise():
70
                        return EMPTY_BB, previous_direction, None
71
                    return EMPTY_BB, previous_direction, Piece.ANUBIS
               case Piece. SCARAB:
7.4
75
                    if previous_direction in [rotation.get_clockwise(), rotation.
       get_anticlockwise()]:
                        new_direction = previous_direction.get_anticlockwise()
7.6
                    else:
77
                        new_direction = previous_direction.get_clockwise()
78
7.9
                    next_square = self.next_square_bitboard(square, new_direction)
80
81
                    return next_square, new_direction, None
82
83
               case Piece.PHAROAH:
84
                    return EMPTY_BB, previous_direction, Piece.PHAROAH
85
86
87
               case None:
                   next_square = self.next_square_bitboard(square, previous_direction
89
90
                    return next_square, previous_direction, None
91
       def next_square_bitboard(self, src_bitboard, previous_direction):
92
           match previous_direction:
93
               case Rotation.UP:
94
                   masked_src_bitboard = src_bitboard & EIGHT_RANK_MASK
                   return masked_src_bitboard << 10</pre>
96
97
               case Rotation.RIGHT:
                   masked_src_bitboard = src_bitboard & J_FILE_MASK
98
99
                    return masked_src_bitboard << 1</pre>
100
               case Rotation.DOWN:
                   masked_src_bitboard = src_bitboard & ONE_RANK_MASK
101
                   return masked_src_bitboard >> 10
102
```

```
case Rotation.LEFT:
                                      masked_src_bitboard = src_bitboard & A_FILE_MASK
104
                                      return masked_src_bitboard >> 1
105
 1 import pygame
 2 from data.utils.board_helpers import coords_to_screen_pos
  3 from data.constants import EMPTY_BB, ShaderType, Colour
  4 from data.managers.animation import animation
  5 from data.managers.window import window
  6 from data.managers.audio import audio
  7 from data.assets import GRAPHICS, SFX
  8 from data.constants import LaserType
 10 type_to_image = {
              LaserType.END: ['laser_end_1', 'laser_end_2'],
              LaserType.STRAIGHT: ['laser_straight_1', 'laser_straight_2'],
 12
              LaserType.CORNER: ['laser_corner_1', 'laser_corner_2']
 1.3
 14 }
 15
 16 GLOW_SCALE_FACTOR = 1.5
 18 class LaserDraw:
              \begin{tabular}{ll} \beg
 19
                      self._board_position = board_position
 20
                      self._square_size = board_size[0] / 10
 2.1
 22
                      self._laser_lists = []
 23
 24
             @property
              def firing(self):
 25
                      return len(self._laser_lists) > 0
 26
 27
 28
             def add_laser(self, laser_result, laser_colour):
 29
                      Adds a laser to the board.
 30
 31
 32
                     Args:
                             laser_result (Laser): Laser class instance containing laser trajectory
               info.
                      laser_colour (Colour.RED | Colour.BLUE): Active colour of laser.
 34
                      laser_path = laser_result.laser_path.copy()
 36
 37
                     laser_types = [LaserType.END]
                      # List of angles in degree to rotate the laser image surface when drawn
 38
                      laser_rotation = [laser_path[0][1]]
 3.9
 40
                      laser_lights = []
 41
 42
                      # Iterates through every square laser passes through
                      for i in range(1, len(laser_path)):
 43
                             previous_direction = laser_path[i-1][1]
 44
                              current_coords , current_direction = laser_path[i]
 45
 46
                              if current_direction == previous_direction:
 47
                                      laser_types.append(LaserType.STRAIGHT)
 48
                                      laser_rotation.append(current_direction)
 49
                              elif current_direction == previous_direction.get_clockwise():
 5.0
                                      laser_types.append(LaserType.CORNER)
 51
                                      laser_rotation.append(current_direction)
 52
 53
                              elif current_direction == previous_direction.get_anticlockwise():
                                      laser_types.append(LaserType.CORNER)
 54
 5.5
                                      laser_rotation.append(current_direction.get_anticlockwise())
 56
                              # Adds a shader ray effect on the first and last square of the laser
 57
```

```
trajectory
                if i in [1, len(laser_path) - 1]:
58
                     abs_position = coords_to_screen_pos(current_coords, self.
59
       _board_position, self._square_size)
                    laser_lights.append([
60
                         (abs_position[0] / window.size[0], abs_position[1] / window.
61
       size[1]),
                         0.35.
62
                          (0, 0, 255) if laser_colour == Colour.BLUE else (255, 0, 0),
63
                     ])
64
6.5
            # Sets end laser draw type if laser hits a piece
66
            if laser_result.hit_square_bitboard != EMPTY_BB:
67
                laser_types[-1] = LaserType.END
68
                laser_path[-1] = (laser_path[-1][0], laser_path[-2][1].get_opposite())
69
                laser_rotation[-1] = laser_path[-2][1].get_opposite()
7.1
72
                audio.play_sfx(SFX['piece_destroy'])
7.3
            laser_path = [(coords, rotation, type) for (coords, dir), rotation, type
       in zip(laser_path, laser_rotation, laser_types)]
            self._laser_lists.append((laser_path, laser_colour))
7.5
76
            window.clear_effect(ShaderType.RAYS)
7.7
            window.set_effect(ShaderType.RAYS, lights=laser_lights)
78
            animation.set_timer(1000, self.remove_laser)
79
80
81
            audio.play_sfx(SFX['laser_1'])
            audio.play_sfx(SFX['laser_2'])
82
83
84
       def remove_laser(self):
8.5
86
            Removes a laser from the board.
87
            self._laser_lists.pop(0)
88
89
            if len(self._laser_lists) == 0:
90
                window.clear_effect(ShaderType.RAYS)
9.1
92
       def draw_laser(self, screen, laser_list, glow=True):
93
94
            Draws every laser on the screen.
95
96
97
                screen (pygame.Surface): The screen to draw on.
98
                {\tt laser\_list}\ ({\tt list})\colon \ {\tt The}\ {\tt list}\ {\tt of}\ {\tt laser}\ {\tt segments}\ {\tt to}\ {\tt draw}\,.
99
                glow (bool, optional): Whether to draw a glow effect. Defaults to True
101
            laser_path , laser_colour = laser_list
            laser_list = []
103
            glow_list = []
104
            for coords, rotation, type in laser_path:
                square_x , square_y = coords_to_screen_pos(coords , self ._board_position
       , self._square_size)
108
                 image = GRAPHICS[type_to_image[type][laser_colour]]
                rotated_image = pygame.transform.rotate(image, rotation.to_angle())
scaled_image = pygame.transform.scale(rotated_image, (self.
        _square_size + 1, self._square_size + 1)) # +1 to prevent rounding creating
       black lines
```

```
laser_list.append((scaled_image, (square_x, square_y)))
                # Scales up the laser image surface as a glow surface
114
                scaled_glow = pygame.transform.scale(rotated_image, (self._square_size
        * GLOW_SCALE_FACTOR, self._square_size * GLOW_SCALE_FACTOR))
                offset = self._square_size * ((GLOW_SCALE_FACTOR - 1) / 2)
                glow_list.append((scaled_glow, (square_x - offset, square_y - offset))
       )
118
           # Scaled glow surfaces drawn on top with the RGB_ADD blend mode
119
           if glow:
120
121
                {\tt screen.fblits(glow\_list, pygame.BLEND\_RGB\_ADD)}
           screen.blits(laser list)
123
124
       def draw(self, screen):
126
           Draws all lasers on the screen.
128
129
           screen (pygame.Surface): The screen to draw on.
130
131
           for laser_list in self._laser_lists:
132
                self.draw_laser(screen, laser_list)
133
134
135
       def handle_resize(self, board_position, board_size):
136
137
           Handles resizing of the board.
138
139
           Args:
                board_position (tuple[int, int]): The new position of the board.
               board_size (tuple[int, int]): The new size of the board.
141
142
           self._board_position = board_position
143
           self._square_size = board_size[0] / 10
144
 1 from data.constants import MoveType, Colour, RotationDirection
 {\tt 2~from~data.utils.bitboard\_helpers~import~notation\_to\_bitboard\,,~coords\_to\_bitboard\,,}\\
       bitboard_to_coords, bitboard_to_notation, print_bitboard
 3 import re
 4 from data.managers.logs import initialise_logger
 6 logger = initialise_logger(__name__)
 8 class Move():
      def __init__(self, move_type, src, dest=None, rotation_direction=None):
1.0
           self.move_type = move_type
           self.src = src
11
           self.dest = dest
12
           self.rotation_direction = rotation_direction
14
1.5
       def to_notation(self, colour, piece, hit_square_bitboard):
           hit_square = ''
16
           if colour == Colour.BLUE:
17
               piece = piece.upper()
18
           if hit_square_bitboard:
20
21
               hit_square = 'x' + bitboard_to_notation(hit_square_bitboard)
22
           if self.move_type == MoveType.MOVE:
23
                return 'M' + piece + bitboard_to_notation(self.src) +
       bitboard_to_notation(self.dest) + hit_square
```

```
2.5
          else:
              return 'R' + piece + bitboard_to_notation(self.src) + self.
      rotation_direction + hit_square
      def __str__(self):
28
          rotate_text = ''
29
           coords_1 = '(' + chr(bitboard_to_coords(self.src)[0] + 65) + ',' + str(
30
      bitboard_to_coords(self.src)[1] + 1) + ')'
3.1
32
           if self.move_type == MoveType.ROTATE:
              rotate_text = ' ' + self.rotation_direction.name
33
34
               return f'{self.move_type.name}{rotate_text}: ON {coords_1}'
35
          elif self.move_type == MoveType.MOVE:
36
               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 {
3.8
      coords_2}'
39
           # (Rotation: {self.rotation_direction})
40
41
      0 classmethod
42
      def instance_from_notation(move_cls, notation):
43
44
              notation = notation.split('x')[0]
45
              move_type = notation[0].lower()
46
47
48
              moves = notation[2:]
              letters = re.findall(r'[A-Za-z]+', moves)
49
              numbers = re.findall(r'\d+', moves)
5.0
51
               if move_type == MoveType.MOVE:
52
                   src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
53
                   dest_bitboard = notation_to_bitboard(letters[1] + numbers[1])
54
5.5
                   return move_cls(move_type, src_bitboard, dest_bitboard)
56
57
               elif move_type == MoveType.ROTATE:
5.8
                   src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
59
                   rotation_direction = RotationDirection(letters[1])
60
61
                  return move_cls(move_type, src_bitboard, src_bitboard,
62
      rotation_direction)
63
               else:
                  raise ValueError('(Move.instance_from_notation) Invalid move type:
64
      ', move_type)
           except Exception as error:
66
              logger.info('(Move.instance_from_notation) Error occured while parsing
67
      :', error)
68
              raise error
69
      @classmethod
70
      def instance_from_input(move_cls, move_type, src, dest=None, rotation=None):
7.1
72
              if move_type == MoveType.MOVE:
73
                   src_bitboard = notation_to_bitboard(src)
74
                   dest_bitboard = notation_to_bitboard(dest)
75
7.6
7.7
               elif move_type == MoveType.ROTATE:
                   src_bitboard = notation_to_bitboard(src)
78
                   dest_bitboard = src_bitboard
79
```

```
return move_cls(move_type, src_bitboard, dest_bitboard, rotation)
81
82
           except Exception as error:
               logger.info('Error (Move.instance_from):', error)
               raise error
84
8.5
86
       @classmethod
       def instance_from_coords(move_cls, move_type, src_coords, dest_coords=None,
87
       rotation direction=None):
           try:
               src_bitboard = coords_to_bitboard(src_coords)
89
               dest_bitboard = coords_to_bitboard(dest_coords)
90
91
               return move_cls(move_type, src_bitboard, dest_bitboard,
92
       rotation direction)
           except Exception as error:
93
               logger.info('Error (Move.instance_from_coords):', error)
94
95
               raise error
96
       @classmethod
97
       def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
98
       None, rotation_direction=None):
           try:
               return move_cls(move_type, src_bitboard, dest_bitboard,
100
       rotation_direction)
101
           except Exception as error:
               logger.info('Error (Move.instance_from_bitboards):', error)
102
               raise error
1 import pygame
 2 from data.constants import OVERLAY_COLOUR_LIGHT, OVERLAY_COLOUR_DARK
 3 from data.utils.board_helpers import coords_to_screen_pos, screen_pos_to_coords,
       create_square_overlay , create_circle_overlay
 5 class OverlayDraw:
       def __init__(self, board_position, board_size, limit_hover=True):
           self._board_position = board_position
           self._board_size = board_size
 8
           self._hovered_coords = None
10
           self._selected_coords = None
11
           self._available_coords = None
12
13
           self._limit_hover = limit_hover
14
15
           self._selected_overlay = None
16
           self._hovered_overlay = None
17
           self._available_overlay = None
19
           self.initialise_overlay_surfaces()
20
21
22
       @property
       def square_size(self):
23
           return self._board_size[0] / 10
24
25
       def initialise_overlay_surfaces(self):
           self._selected_overlay = create_square_overlay(self.square_size,
27
       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)
```

```
31
      def set_hovered_coords(self, mouse_pos):
          self._hovered_coords = screen_pos_to_coords(mouse_pos, self.
32
      _board_position, self._board_size)
33
      def set_selected_coords(self, coords):
34
          self._selected_coords = coords
35
36
      def set_available_coords(self, coords_list):
3.7
          self._available_coords = coords_list
38
3.9
40
      def set_hover_limit(self, new_limit):
          self._limit_hover = new_limit
41
42
      def draw(self, screen):
43
          self.set_hovered_coords(pygame.mouse.get_pos())
44
45
46
          if self._selected_coords:
              screen.blit(self._selected_overlay, coords_to_screen_pos(self.
47
      _selected_coords, self._board_position, self.square_size))
48
          if self._available_coords:
49
               for coords in self._available_coords:
50
                   screen.blit(self._available_overlay, coords_to_screen_pos(coords,
5.1
      self._board_position, self.square_size))
52
53
          if self._hovered_coords:
54
               if self._hovered_coords is None:
                   return
55
5.6
              if self._limit_hover and ((self._available_coords is None) or (self.
      _hovered_coords not in self._available_coords)):
5.8
                  return
59
               \verb|screen.blit(self._hovered_overlay, coords_to\_screen_pos(self.|
60
      _hovered_coords, self._board_position, self.square_size))
61
      def handle_resize(self, board_position, board_size):
62
          self._board_position = board_position
          self _board_size = board_size
64
6.5
          self.initialise_overlay_surfaces()
1 import pygame
2 from random import randint
3 from data.utils.asset_helpers import get_perimeter_sample, get_vector,
      get_angle_between_vectors, get_next_corner
4 from data.states.game.components.piece_sprite import PieceSprite
6 class ParticlesDraw:
      def __init__(self, gravity=0.2, rotation=180, shrink=0.5, opacity=150):
          self._particles = []
          self._glow_particles = []
10
11
          self._gravity = gravity
          self._rotation = rotation
          self._shrink = shrink
13
          self._opacity = opacity
14
15
16
      def fragment_image(self, image, number):
17
          image_size = image.get_rect().size
1.8
```

```
1. Takes an image surface and samples random points on the perimeter.
           2. Iterates through points, and depending on the nature of two consecutive
20
       points, finds a corner between them.
          3. Draws a polygon with the points as the vertices to mask out the area
      not in the fragment.
22
23
           Args:
               {\tt image \ (pygame.Surface): \ Image \ to \ fragment.}
24
               number (int): The number of fragments to create.
2.5
26
27
          Returns:
              list[pygame.Surface]: List of image surfaces with fragment of original
       surface drawn on top.
29
30
           center = image.get_rect().center
          points_list = get_perimeter_sample(image_size, number)
fragment_list = []
3.1
32
33
           points_list.append(points_list[0])
34
35
           # Iterate through points_list, using the current point and the next one
36
           for i in range(len(points_list) - 1):
3.7
               vertex_1 = points_list[i]
38
               vertex_2 = points_list[i + 1]
39
               vector_1 = get_vector(center, vertex_1)
40
               vector_2 = get_vector(center, vertex_2)
41
42
               angle = get_angle_between_vectors(vector_1, vector_2)
43
               cropped_image = pygame.Surface(image_size, pygame.SRCALPHA)
44
               cropped_image.fill((0, 0, 0, 0))
45
46
               cropped_image.blit(image, (0, 0))
47
48
               corners_to_draw = None
49
               if vertex_1[0] == vertex_2[0] or vertex_1[1] == vertex_2[1]: # Points
5.0
      on the same side
                   corners_to_draw = 4
51
5.2
               elif abs(vertex_1[0] - vertex_2[0]) == image_size[0] or abs(vertex_1
53
      [1] - vertex_2[1]) == image_size[1]: # Points on opposite sides
5.4
                   corners_to_draw = 2
55
               elif angle < 180: # Points on adjacent sides
56
57
                   corners_to_draw = 3
58
5.9
               else:
                   corners_to_draw = 1
60
61
62
               corners_list = []
63
               for j in range(corners_to_draw):
                   if len(corners_list) == 0:
64
65
                       corners_list.append(get_next_corner(vertex_2, image_size))
66
                       corners_list.append(get_next_corner(corners_list[-1],
67
      image_size))
68
               pygame.draw.polygon(cropped_image, (0, 0, 0, 0), (center, vertex_2, *
69
      corners_list, vertex_1))
7.0
71
               fragment_list.append(cropped_image)
72
          return fragment_list
73
```

```
74
       def add_captured_piece(self, piece, colour, rotation, position, size):
75
7.6
           Adds a captured piece to fragment into particles.
77
78
79
            Args:
               piece (Piece): The piece type.
80
                colour (Colour): The active colour of the piece. rotation (int): The rotation of the piece.
8.1
82
                position (tuple[int, int]): The position where particles originate
83
       from.
               size (tuple[int, int]): The size of the piece.
85
           piece_sprite = PieceSprite(piece, colour, rotation)
86
           piece_sprite.set_geometry((0, 0), size)
87
           piece_sprite.set_image()
88
89
90
           particles = self.fragment_image(piece_sprite.image, 5)
91
           for particle in particles:
                self.add_particle(particle, position)
93
94
       def add_sparks(self, radius, colour, position):
95
96
           Adds laser spark particles.
97
98
99
            Args:
100
                radius (int): The radius of the sparks.
                colour (Colour): The active colour of the sparks.
101
                position \ (tuple[int,\ int]): \ The \ position \ where \ particles \ originate
102
       from.
104
           for i in range(randint(10, 15)):
                velocity = [randint(-15, 15) / 10, randint(-20, 0) / 10]
                random\_colour = [min(max(val + randint(-20, 20), 0), 255) for val in
       colourl
               self._particles.append([None, [radius, random_colour], [*position],
       velocity, 0])
       def add_particle(self, image, position):
           Adds a particle.
111
112
113
            Args:
               image (pygame.Surface): The image of the particle.
114
115
                position (tuple): The position of the particle.
           velocity = [randint(-15, 15) / 10, randint(-20, 0) / 10]
118
           # Each particle is stored with its attributes: [surface, copy of surface,
       position, velocity, lifespan]
120
           self._particles.append([image, image.copy(), [*position], velocity, 0])
121
       def update(self):
           Updates each particle and its attributes.
124
125
           for i in range(len(self._particles) - 1, -1, -1):
126
               particle = self._particles[i]
128
                #update position
                particle[2][0] += particle[3][0]
130
```

```
particle[2][1] += particle[3][1]
131
               #update lifespan
               self._particles[i][4] += 0.01
136
               if self._particles[i][4] >= 1:
                    self._particles.pop(i)
137
                    continue
138
               if isinstance(particle[1], pygame.Surface): # Particle is a piece
140
                    # Update velocity
141
142
                    particle[3][1] += self._gravity
143
                    # Update size
144
                    image_size = particle[1].get_rect().size
145
                    end_size = ((1 - self._shrink) * image_size[0], (1 - self._shrink)
146
        * image_size[1])
                    target_size = (image_size[0] - particle[4] * (image_size[0]
147
       end_size[0]), image_size[1] - particle[4] * (image_size[1] - end_size[1]))
148
149
                    # Update rotation
                    rotation = (self._rotation if particle[3][0] \leftarrow 0 else -self.
       _rotation) * particle[4]
151
                   updated_image = pygame.transform.scale(pygame.transform.rotate(
       particle[1], rotation), target_size)
                elif isinstance(particle[1], list): # Particle is a spark
154
                    # Update size
                    end_radius = (1 - self._shrink) * particle[1][0]
157
                    target_radius = particle[1][0] - particle[4] * (particle[1][0] -
       end radius)
158
                    updated_image = pygame.Surface((target_radius * 2, target_radius *
159
        2), pygame.SRCALPHA)
                   pygame.draw.circle(updated_image, particle[1][1], (target_radius,
160
       target_radius), target_radius)
161
               # Update opacity
162
               alpha = 255 - particle[4] * (255 - self._opacity)
163
164
               updated_image.fill((255, 255, 255, alpha), None, pygame.
       BLEND_RGBA_MULT)
166
               particle[0] = updated_image
167
168
       def draw(self, screen):
           Draws the particles, indexing the surface and position attributes for each
171
        particle.
           Args:
           screen (pygame.Surface): The screen to draw on.
174
           screen.blits([
               (particle[0], particle[2]) for particle in self._particles
           1)
178
 1 import pygame
 2 from data.constants import EMPTY_BB, Colour, Piece
 3 from data.states.game.components.piece_sprite import PieceSprite
 4 from data.utils.board_helpers import coords_to_screen_pos
```

```
5 from data.utils import bitboard_helpers as bb_helpers
7 class PieceGroup(pygame.sprite.Group):
      def __init__(self):
          # self.square_list = []
9
          # self.valid_square_list_positions = []
10
          super().__init__()
11
12
      def initialise_pieces(self, piece_list, board_position, board_size):
13
14
          self.empty()
1.5
16
          for index, piece_and_rotation in enumerate(piece_list):
              x = index % 10
17
               y = index // 10
18
19
               if piece_and_rotation:
2.0
                   if piece_and_rotation[0].isupper():
21
22
                       colour = Colour.BLUE
23
                   else:
                       colour = Colour.RED
24
25
                   piece = PieceSprite(piece=Piece(piece_and_rotation[0].lower()),
26
      colour = colour , rotation = pie ce_and_rotation [1])
                   piece.set_coords((x, y))
27
                   piece.set_geometry(board_position, board_size[0] / 10)
28
29
                   piece.set_image()
                   self.add(piece)
3.0
31
      def set_geometry(self, board_position, board_size):
32
          for sprite in self.sprites():
3.3
34
               sprite.set_geometry(board_position, board_size[0] / 10)
3.5
36
      def handle_resize(self, board_position, board_size):
           self.set_geometry(board_position, board_size)
37
38
          for sprite in self.sprites():
39
              sprite.set_image()
40
41
      def remove_piece(self, coords):
42
          for sprite in self.sprites():
43
44
               if sprite.coords == coords:
                   sprite.kill()
45
46
47
      # def handle_resize_end(self):
            for sprite in self.sprites():
48
                 sprite.handle_resize_end()
49
      # def clear_square(self, src_bitboard):
51
            list_position = bb_helpers.bitboard_to_index(src_bitboard)
52
      #
53
      #
             self.square_list[list_position].clear_piece()
54
      # def update_squares_move(self, src, dest, new_piece_symbol, new_colour,
55
      rotation):
             self.square_list[src].clear_piece()
5.6
             self.square_list[dest].clear_piece()
            self.square_list[dest].set_piece(piece_symbol=new_piece_symbol, colour=
58
      new_colour, rotation=rotation)
6.0
      # def update_squares_rotate(self, src, piece_symbol, colour, new_rotation):
61
             self.square_list[src].clear_piece()
             self.square_list[src].set_piece(piece_symbol=piece_symbol, colour=colour
62
      , rotation=new_rotation)
```

```
# def add_valid_square_overlays(self, valid_bitboard):
64
             if valid_bitboard == EMPTY_BB:
6.5
      #
                 return
67
             list_positions = self.bitboard_to_list_positions(valid_bitboard)
68
      #
             self.valid_square_list_positions = list_positions
69
7.0
             for \ square\_position \ in \ list\_positions:
71
      #
                 square = self.square_list[square_position]
72
                 square.selected = True
      #
7.3
74
      # def remove_valid_square_overlays(self):
75
             for \ square\_position \ in \ self.valid\_square\_list\_positions:
76
      #
                 square = self.square_list[square_position]
7.7
      #
                 square.selected = False
7.8
79
      #
                 square.remove_overlay()
80
             self.valid_square_list_positions = []
81
      # def draw_valid_square_overlays(self):
83
             for \ square\_position \ in \ self.valid\_square\_list\_positions:
8.4
                 square = self.square_list[square_position]
85
                 square.draw_overlay()
      #
86
87
      # def bitboard_to_list_positions(self, bitboard):
88
            list_positions = []
89
      #
90
             for square in bb_helpers.occupied_squares(bitboard):
91
                 list_positions.append(bb_helpers.bitboard_to_index(square))
92
      #
             return list_positions
94
1 import pygame
2 from data.assets import GRAPHICS
3 from data.constants import Colour, Piece
4 from data.utils.asset_helpers import scale_and_cache
5 from data.utils.board_helpers import coords_to_screen_pos
7 class EmptyPiece(pygame.sprite.Sprite):
      def __init__(self):
9
           super().__init__()
10
           self.image = pygame.Surface((1, 1))
           self.rect = self.image.get_rect()
12
           self.rect.topleft = (0, 0)
13
14
      def set_image(self, type):
15
16
          pass
17
      def set_rect(self):
18
19
          pass
20
      def set_geometry(self, anchor_position, size):
21
22
           pass
24 class PieceSprite(pygame.sprite.Sprite):
25
     def __init__(self, piece, colour, rotation):
          super().__init__()
26
          self.colour = colour
27
          self.rotation = rotation
28
29
```

```
self.type = piece
           self.coords = None
31
           self.size = None
32
       @property
34
      def image_name(self):
3.5
           return Piece(self.type).name.lower() + '_' + str(self.colour) + '_' + self
36
       . rotation
3.7
38
      def set_image(self):
          self.image = scale_and_cache(GRAPHICS[self.image_name], (self.size, self.
3.9
       size))
40
      def set_geometry(self, new_position, square_size):
41
           self.size = square_size
42
           self.rect = pygame.FRect((0, 0, square_size, square_size))
43
44
45
           if self.coords:
               self.rect.topleft = coords_to_screen_pos(self.coords, new_position,
46
       square_size)
47
          else:
               self.rect.topleft = new_position
48
      def set_coords(self, new_coords):
50
           self.coords = new_coords
51
1 from data.constants import Piece
3 \text{ 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,
9
1.0
11
       0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
12
13
14 PSQT = {
      Piece.PYRAMID: [
15
16
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
17
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
18
19
           0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0,
20
           21
           0, 0, 0, 0, 0, 0, 0, 0, 0,
23
24
      ],
      Piece. ANUBIS: [
25
           0, 0, 0, 0, 0, 0, 0, 0, 0,
26
27
           0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
28
           0, 0, 0, 0, 0, 0, 0, 0, 0,
29
           0, 0, 0, 0, 0, 0, 0, 0, 0,
           6, 6, 6, 6, 6, 6, 6, 6, 6,
31
32
           4, 4, 4, 4, 4, 4, 4, 4, 4,
           2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
33
34
      Piece.SCARAB: [
35
           0, 0, 0, 0, 0, 0, 0, 0, 0,
36
```

```
0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
37
38
          0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
39
          0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
40
          41
42
          0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0,
43
44
     Piece.PHAROAH: [
45
          0, 0, 0, 0, 0, 0, 0, 0, 0,
46
          0, 0, 0, 0, 0, 0, 0, 0, 0,
47
48
          0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0,
49
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
5.0
          0, 0, 0, 0, 0, 0, 0, 0, 0,
51
          0, 0, 0, 2, 2, 2, 2, 0, 0, 0,
52
53
          0, 0, 0, 2, 4, 4, 2, 0, 0, 0,
54
      ],
55 }
1 from data.states.game.cpu.engines import *
2 from data.states.game.components.board import Board
3 from data.constants import Colour, Miscellaneous
4 from data.managers.logs import initialise_logger
6 logger = initialise_logger(__name__)
7 # sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa
     b
8 + scfaRa7/RaRaRaFa6/RaRaRa7/10/10/10/10/9Sa b
9 # scfa8/10/10/10/10/10/8FaSa b
10
def compare(cls1, cls2, depth, rounds):
      wins = [0, 0]
12
13
      board = Board()
14
      def callback(move):
1.5
          board.apply_move(move, add_hash=True)
17
      cpu1 = cls1(callback=callback, max_depth=depth, verbose='compact')
18
      cpu2 = cls2(callback=callback, max_depth=depth, verbose='compact')
20
21
      for i in range(rounds):
          board = Board(fen_string="scfa8/10/10/10/10/10/10/8FaSa b")
22
          ply = 0
23
24
          if i % 2 == 0:
25
             players = { Colour.BLUE: cpu1, Colour.RED: cpu2, Miscellaneous.DRAW: '
26
      DRAW' 1
27
          else:
              players = { Colour.BLUE: cpu2, Colour.RED: cpu1, Miscellaneous.DRAW: '
      DRAW' }
29
          while (winner := board.check_win()) is None:
30
              players[board.get_active_colour()].find_move(board, None)
31
32
              ply += 1
              logger.debug('PLY:', ply)
34
35
          if winner == Miscellaneous.DRAW:
              wins[0] += 0.5
36
              wins[1] += 0.5
3.7
38
          else:
              if players[winner] == cpu1:
39
```

```
wins[0] += 1
41
               else:
                   wins[1] += 1
42
           logger.debug(f'ROUND {i + 1} | WINNER: {players[winner]} | PLY: {ply}')
44
45
      logger.debug(f'{cpu1} SCORE: {wins[0]} | {cpu2} SCORE: {wins[1]}')
46
47
48 compare (TTNegamaxCPU, TTNegamaxCPU, 2, 1)
1 import time
2 from pprint import PrettyPrinter
3 from data.constants import Colour, Score, Miscellaneous
4 from data.states.game.cpu.evaluator import Evaluator
5 from data.managers.logs import initialise_logger
7 logger = initialise_logger(__name__)
8 printer = PrettyPrinter(indent=2, sort_dicts=False)
10 class BaseCPU:
     def __init__(self, callback, verbose=True):
          self._evaluator = Evaluator(verbose=False)
12
13
          self._verbose = verbose
          self._callback = callback
          self._stats = {}
1.5
16
     def initialise_stats(self):
17
18
          self._stats = {
19
               'nodes': 0,
               'leaf_nodes' : 0,
20
               'draws': 0,
21
22
               'mates': 0,
               'ms_per_node': 0,
23
24
               'time_taken': time.time()
          }
25
26
27
     def print_stats(self, score, move):
28
           \hbox{\tt Prints statistics after traversing tree.} \\
29
31
          Args:
32
               score (int): Final score obtained after traversal.
              move (Move): Best move obtained after traversal.
33
3.4
35
          if self._verbose is False:
              return
36
37
          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)
40
           # Prints stats across multiple lines
41
           if self._verbose is True:
42
               logger.info(f'\n\n'
43
                           f'{self.__str__()} Search Results:\n'
                           45
46
                           f' \ Best \ score: \ \{score\} \qquad Best \ move: \ \{move\} \backslash n'
47
48
          # Prints stats in a compacted format
49
          elif self._verbose.lower() == 'compact':
50
```

```
logger.info(self._stats)
51
               logger.info(f'Best score: {score} Best move: {move}')
52
53
      def find_move(self, board, stop_event=None):
           raise NotImplementedError
55
5.6
      def search(self, board, depth, stop_event, absolute=False, **kwargs):
57
          if stop_event and stop_event.is_set():
    raise Exception(f'Thread killed - stopping minimax function ({self.
5.8
59
      __str__}.search)')
60
61
           self._stats['nodes'] += 1
62
           if (winner := board.check_win()) is not None:
63
               self._stats['leaf_nodes'] += 1
64
               return self.process_win(winner, depth, absolute)
6.5
66
67
           if depth == 0:
               self._stats['leaf_nodes'] += 1
68
               return self._evaluator.evaluate(board, absolute), None
69
70
      def process_win(self, winner, depth, absolute):
71
           self._stats['leaf_nodes'] += 1
72
7.3
           if winner == Miscellaneous.DRAW:
74
               self._stats['draws'] += 1
75
7.6
               return 0, None
77
           elif winner == Colour.BLUE or absolute:
               self._stats['mates'] += 1
78
               return Score.CHECKMATE + depth, None
7.9
80
           elif winner == Colour.RED:
               self._stats['mates'] += 1
8.1
               return -Score.CHECKMATE - depth, None
82
83
     def __str__(self):
84
           return self.__class__._name__
1 import threading
2 import time
3 from data.managers.logs import initialise_logger
5 logger = initialise_logger(__name__)
7 class CPUThread(threading.Thread):
      def __init__(self, cpu, verbose=False):
           super().__init__()
9
10
           self._stop_event = threading.Event()
           self._running = True
11
          self._verbose = verbose
12
           self.daemon = True
13
14
1.5
           self._board = None
           self._cpu = cpu
17
      def kill_thread(self):
18
           Kills the CPU and terminates the thread by stopping the run loop.
20
21
22
           self.stop_cpu()
23
           self._running = False
24
    def stop_cpu(self):
25
```

```
Kills the CPU's move search.
27
28
           self._stop_event.set()
          self._board = None
30
31
      def start_cpu(self, board):
32
3.3
           Starts the CPU's move search.
3.4
35
36
          Args:
          board (Board): The current board state.
37
38
           self._stop_event.clear()
3.9
           self._board = board
41
     def run(self):
42
43
           Periodically checks if the board variable is set.
44
          If it is, then starts CPU search.
          0.00
46
          while self._running:
47
               if self._board and self._cpu:
                   self._cpu.find_move(self._board, self._stop_event)
49
50
                   self.stop_cpu()
               else:
51
52
                   time.sleep(1)
53
                   if self._verbose:
                       logger.debug(f'(CPUThread.run) Thread {threading.get_native_id
54
      ()} idling...')
1 from data.utils.bitboard_helpers import pop_count, occupied_squares,
      bitboard_to_index
2 from data.states.game.components.psqt import PSQT, FLIP
3 from data.managers.logs import initialise_logger
4 from data.constants import Colour, Piece, Score
6 logger = initialise_logger(__name__)
8 class Evaluator:
      def __init__(self, verbose=True):
9
10
           self._verbose = verbose
11
      def evaluate(self, board, absolute=False):
12
13
          Evaluates and returns a numerical score for the board state.
14
15
               board (Board): The current board state.
               absolute (bool): Whether to always return the absolute score from the
      active colour's perspective (for NegaMax).
19
           Returns:
20
          int: Score representing advantage/disadvantage for the player.
21
22
          blue_score = (
               self.evaluate_material(board, Colour.BLUE),
24
25
               {\tt self.evaluate\_position(board, Colour.BLUE)}\ ,
               self.evaluate_mobility(board, Colour.BLUE),
26
               {\tt self.evaluate\_pharoah\_safety(board, Colour.BLUE)}
27
          )
28
29
```

```
red_score = (
30
               self.evaluate_material(board, Colour.RED),
31
               self.evaluate_position(board, Colour.RED),
32
               self.evaluate_mobility(board, Colour.RED),
               self.evaluate_pharoah_safety(board, Colour.RED)
34
          )
3.5
36
          if self._verbose:
3.7
               logger.info(f'Material: {blue_score[0]} | {red_score[0]}')
3.8
               logger.info(f'Position: {blue_score[1]} | {red_score[1]}')
39
               logger.info(f'Mobility: {blue_score[2]} | {red_score[2]}')
40
41
               logger.info(f'Safety: {blue_score[3]} | {red_score[3]}')
               logger.info(f'Overall score: {sum(blue_score) - sum(red_score)}')
42
43
           if absolute and board.get_active_colour() == Colour.RED:
44
              return sum(red_score) - sum(blue_score)
45
46
           else:
47
               return sum(blue_score) - sum(red_score)
48
      def evaluate_material(self, board, colour):
49
50
          Evaluates the material score for a given colour.
5.1
52
53
          Args:
              board (Board): The current board state.
54
               colour (Colour): The colour to evaluate.
55
56
57
             int: Sum of all piece scores.
58
5.9
60
           return (
              Score.SPHINX * board.bitboards.get_piece_count(Piece.SPHINX, colour) +
6.1
62
               Score.PYRAMID * board.bitboards.get_piece_count(Piece.PYRAMID, colour)
               Score.ANUBIS * board.bitboards.get_piece_count(Piece.ANUBIS, colour) +
63
               Score.SCARAB * board.bitboards.get_piece_count(Piece.SCARAB, colour)
64
          )
65
66
      def evaluate_position(self, board, colour):
67
68
          Evaluates the positional score for a given colour.
69
70
71
          Args:
               board (Board): The current board state.
72
              colour (Colour): The colour to evaluate.
73
7.4
75
          int: Score representing positional advantage/disadvantage.
76
7.7
78
          score = 0
7.9
          for piece in Piece:
80
               if piece == Piece.SPHINX:
81
                   continue
82
              piece_bitboard = board.bitboards.get_piece_bitboard(piece, colour)
84
85
               for bitboard in occupied_squares(piece_bitboard):
86
87
                   index = bitboard_to_index(bitboard)
88
                   # Flip PSQT if using from blue player's perspective
                   index = FLIP[index] if colour == Colour.BLUE else index
89
90
```

```
score += PSQT[piece][index] * Score.POSITION
91
92
           return score
93
       def evaluate_mobility(self, board, colour):
95
96
           Evaluates the mobility score for a given colour.
97
98
99
           Args:
               board (Board): The current board state.
100
               colour (Colour): The colour to evaluate.
101
           Returns:
103
              int: Score on numerical representation of mobility.
104
105
           number_of_moves = board.get_mobility(colour)
106
107
           return number_of_moves * Score.MOVE
108
       def evaluate_pharoah_safety(self, board, colour):
109
           Evaluates the safety of the Pharoah for a given colour.
112
113
           Args:
               board (Board): The current board state.
114
               colour (Colour): The colour to evaluate.
115
116
           Returns:
           int: Score representing mobility of the Pharoah.
118
119
           pharoah_bitboard = board.bitboards.get_piece_bitboard(Piece.PHAROAH,
120
       colour)
122
           if pharoah_bitboard:
               pharoah_available_moves = pop_count(board.get_valid_squares(
       pharoah_bitboard, colour))
124
              return (8 - pharoah_available_moves) * Score.PHAROAH_SAFETY
           else:
               return 0
126
1 from data.states.game.cpu.evaluator import Evaluator
 2 from data.constants import Colour
 g from data.utils.bitboard_helpers import print_bitboard, pop_count
 5 class SimpleEvaluator:
       def __init__(self):
           self._evaluator = Evaluator(verbose=False)
           self._cache = {}
      def evaluate(self, board):
10
           if (hashed := board.to_hash()) in self._cache:
11
               return self._cache[hashed]
12
13
           score = self._evaluator.evaluate_material(board, board.get_active_colour()
           self._cache[hashed] = score
1.5
           return score
17
19 class MoveOrderer:
2.0
      def __init__(self):
           self._evaluator = SimpleEvaluator()
21
22
```

```
# def get_eval(self, board, move):
23
            laser_result = board.apply_move(move)
24
            score = self._evaluator.evaluate(board)
25
      #
      #
            board.undo_move(move, laser_result)
26
      #
            return score
27
28
      # def score_moves(self, board, moves):
29
            for i in range(len(moves)):
      #
3.0
31
      #
                 score = self.get_eval(board, moves[i])
                 moves[i] = (moves[i], score)
32
33
34
            return moves
35
      def best_move_to_front(self, moves, start_idx, hint):
36
          for i in range(start_idx + 1, len(moves)):
37
               if moves[i].src in hint:
38
                   moves[i], moves[start_idx] = moves[start_idx], moves[i]
39
40
41
      def get_moves(self, board, hint=None):
          colour = board.get_active_colour()
43
          moves = list(board.generate_all_moves(colour))
44
          for i in range(len(moves)):
46
47
               if hint:
                   self.best_move_to_front(moves, i, hint)
48
49
50
               yield moves[i]
1 from data.constants import Score, Colour
2 from data.states.game.cpu.transposition_table import TranspositionTable
3 from data.states.game.cpu.base import BaseCPU
4 from pprint import pprint
6 class MinimaxCPU(BaseCPU):
      def __init__(self, max_depth, callback, verbose):
          super().__init__(callback, verbose)
          self._max_depth = max_depth
9
10
      def find_move(self, board, stop_event):
11
          # No bit_length bug as None type returned, so Move __str__ called on
12
      NoneType I think (just deal with None being returned)
          try:
              best_move = self.search(board, self._max_depth, -Score.INFINITE, Score
14
      .INFINITE, stop_event)
15
16
               if self._verbose:
                   print('\nCPU Search Results:')
17
                   pprint(self._stats)
18
                   print('Best move:', best_move, '\n')
19
20
                   self._callback(self._best_move)
21
           except Exception as error:
22
              print('(MinimaxBase.find_move) Error has occured:')
23
               raise error
24
      def search(self, board, depth, alpha, beta, stop_event):
26
27
          if stop_event.is_set():
              raise Exception('Thread killed - stopping minimax function (CPU.
28
      minimax)')
29
          # cached_move, cached_score = self._transposition_table.get_entry(hash_key
30
```

```
=board.bitboards.get_hash(), depth=depth, alpha=alpha, beta=beta)
           # if cached_move or cached_score:
31
                 if depth == self._max_depth:
           #
32
           #
                     self._best_move = cached_move
33
          #
                 return cached_score
34
3.5
36
           if depth == 0:
3.7
               return self.evaluate(board)
38
39
          is_maximiser = board.get_active_colour() == Colour.BLUE
40
41
           if is_maximiser:
42
               score = -Score.INFINITE
43
               for move in board.generate_all_moves(board.get_active_colour()):
45
46
                   before, before_score = board.bitboards.get_rotation_string(), self
      .evaluate(board)
47
                   laser_result = board.apply_move(move)
48
                   new_score = self.minimax(board, depth - 1, alpha, beta, False,
49
      stop_event)
                   if new_score >= score:
5.1
                        score = new_score
52
53
54
                        if depth == self._max_depth:
55
                            self._best_move = move
56
                   board.undo_move(move, laser_result)
5.7
58
                   alpha = max(alpha, score)
59
60
                   if depth == self._max_depth: # https://stackoverflow.com/questions
      /31429974/alphabeta-pruning-alpha-equals-or-greater-than-beta-why-equals
                       if beta < alpha:</pre>
6.1
62
                            break
                   else:
63
                       if beta <= alpha:</pre>
64
                           break
65
66
                   after, after_score = board.bitboards.get_rotation_string(), self.
67
      evaluate(board)
                   if (before != after or before_score != after_score):
68
                        print('shit\n\n')
69
70
               return score
7.1
          else:
73
               score = Score.INFINITE
74
75
               for move in board.generate_all_moves(board.get_active_colour()):
7.6
77
                   bef, before_score = board.bitboards.get_rotation_string(), self.
      evaluate(board)
7.8
                   laser_result = board.apply_move(move)
                   new_score = self.minimax(board, depth - 1, alpha, beta, False,
80
      stop_event)
81
82
                   if new_score <= score:</pre>
83
                        score = new_score
                        if depth == self._max_depth:
84
                            self._best_move = move
85
```

```
board.undo_move(move, laser_result)
87
88
                    beta = min(beta, score)
                    if depth == self._max_depth:
90
                        if beta < alpha:</pre>
91
                            break
92
                    else:
93
                        if beta <= alpha:</pre>
94
95
96
97
                    after, after_score = board.bitboards.get_rotation_string(), self.
       evaluate(board)
                    if (bef != after or before_score != after_score):
9.8
                        print('shit\n\n')
99
                         raise ValueError
100
101
102
                return score
 1 from data.constants import TranspositionFlag
 3 class TranspositionEntry:
       def __init__(self, score, move, flag, hash_key, depth):
            self.score = score
            self.move = move
            self.flag = flag
           self.hash_key = hash_key
 9
           self.depth = depth
10
11 class TranspositionTable:
       def __init__(self, max_entries=100000):
12
13
            self._max_entries = max_entries
            self._table = dict()
14
15
       def calculate_entry_index(self, hash_key):
16
           Gets the dictionary key for a given Zobrist hash.
19
20
            Args:
               hash_key (int): A Zobrist hash.
21
22
23
            Returns:
            int: Key for the given hash.
24
25
            # return hash_key % self._max_entries
26
           return hash_key
27
28
29
       def insert_entry(self, score, move, hash_key, depth, alpha, beta):
30
31
           Inserts an entry into the transposition table.
32
33
            Args:
34
               score (int): The evaluation score.
                move (Move): The best move found.
35
                hash_key (int): The Zobrist hash key.
36
                depth (int): The depth of the search.
                alpha (int): The upper bound value. beta (int): The lower bound value.
38
39
40
41
           Raises:
           Exception: Invalid depth or score.
42
43
```

```
if depth == 0 or alpha < score < beta:</pre>
44
               flag = TranspositionFlag.EXACT
45
               score = score
46
           elif score <= alpha:</pre>
              flag = TranspositionFlag.UPPER
48
               score = alpha
49
           elif score >= beta:
50
               flag = TranspositionFlag.LOWER
5.1
               score = beta
52
53
               raise Exception('(TranspositionTable.insert_entry)')
54
           self._table[self.calculate_entry_index(hash_key)] = TranspositionEntry(
56
      score, move, flag, hash_key, depth)
           if len(self._table) > self._max_entries:
5.8
59
               # Removes the longest-existing entry to free up space for more up-to-
      date entries
              # Expression to remove leftmost item taken from https://docs.python.
60
      \verb|org/3/library/collections.html#| ordereddict-objects|
              (k := next(iter(self._table)), self._table.pop(k))
61
62
      def get_entry(self, hash_key, depth, alpha, beta):
63
64
           Gets an entry from the transposition table.
65
66
67
           Args:
68
               hash_key (int): The Zobrist hash key.
               depth (int): The depth of the search.
69
               alpha (int): The alpha value for pruning.
               beta (int): The beta value for pruning.
73
          Returns:
               tuple[int, Move] | tuple[None, None]: The evaluation score and the
74
      best move found, if entry exists.
75
          index = self.calculate_entry_index(hash_key)
76
           if index not in self._table:
              return None, None
79
80
           entry = self._table[index]
81
82
83
           if entry.hash_key == hash_key and entry.depth >= depth:
              if entry.flag == TranspositionFlag.EXACT:
84
8.5
                   return entry.score, entry.move
               if entry.flag == TranspositionFlag.LOWER and entry.score >= beta:
87
88
                   return entry.score, entry.move
89
               if entry.flag == TranspositionFlag.UPPER and entry.score <= alpha:</pre>
9.0
                   return entry.score, entry.move
91
92
          return None, None
93
1 from random import randint
2 from data.utils.bitboard_helpers import bitboard_to_index
3 from data.constants import Piece, Colour, Rotation
{\scriptscriptstyle 5} # Initialise random values for each piece type on every square
_{6} # (5 x 2 colours) pieces + 4 rotations, for 80 squares
7 zobrist_table = [[randint(0, 2 ** 64) for i in range(14)] for j in range(80)]
```

```
{\bf 8} # Hash for when the red player's move
9 red_move_hash = randint(0, 2 ** 64)
{\tt 11} # Maps piece to the correct random value
12 piece_lookup = {
      Colour.BLUE: {
13
          piece: i for i, piece in enumerate(Piece)
14
      }.
1.5
16
      Colour.RED: {
           piece: i + 5 for i, piece in enumerate(Piece)
17
18
19 }
20
{\scriptstyle 21} # Maps rotation to the correct random value
22 rotation_lookup = {
      rotation: i + 10 for i, rotation in enumerate(Rotation)
23
24 }
25
26 class ZobristHasher:
      def __init__(self):
27
           self.hash = 0
28
29
      def get_piece_hash(self, index, piece, colour):
30
3.1
           Gets the random value for the piece type on the given square.
32
33
3.4
           Args:
35
               index (int): The index of the square.
               piece (Piece): The piece on the square.
36
               colour (Colour): The colour of the piece.
3.7
38
           Returns:
3.9
           int: A 64-bit value.
40
41
           piece_index = piece_lookup[colour][piece]
42
           return zobrist_table[index][piece_index]
44
      def get_rotation_hash(self, index, rotation):
45
           Gets the random value for theon the given square.
47
48
49
           Args:
               index (int): The index of the square.
5.0
51
               rotation (Rotation): The rotation on the square.
               colour (Colour): The colour of the piece.
52
5.3
           Returns:
             int: A 64-bit value.
55
56
57
           rotation_index = rotation_lookup[rotation]
           return zobrist_table[index][rotation_index]
58
59
      def apply_piece_hash(self, bitboard, piece, colour):
60
6.1
           Updates the Zobrist hash with a new piece.
63
64
           Args:
               bitboard (int): The bitboard representation of the square.
               piece (Piece): The piece on the square.
66
               colour (Colour): The colour of the piece.
67
68
           index = bitboard_to_index(bitboard)
69
```

```
piece_hash = self.get_piece_hash(index, piece, colour)
           self hash ^= piece_hash
71
72
      def apply_rotation_hash(self, bitboard, rotation):
73
             "Updates the Zobrist hash with a new rotation.
74
7.5
76
          Args:
              bitboard (int): The bitboard representation of the square.
7.7
              rotation (Rotation): The rotation on the square.
78
79
           index = bitboard_to_index(bitboard)
8.0
81
           rotation_hash = self.get_rotation_hash(index, rotation)
          self.hash ^= rotation_hash
82
83
84
      def apply_red_move_hash(self):
8.5
           Applies the Zobrist hash for the red player's move.
86
87
          self.hash ^= red_move_hash
88
2 from data.states.game.cpu.move_orderer import MoveOrderer
{\tt 3} from data.states.game.cpu.base import BaseCPU
4 from data.constants import Score, Colour
5 from random import choice
6 from data.utils.bitboard_helpers import print_bitboard
7 orderer = MoveOrderer()
9 class ABMinimaxCPU(BaseCPU):
      def __init__(self, max_depth, callback, verbose=True):
10
11
           super().__init__(callback, verbose)
12
           self._max_depth = max_depth
1.3
      def initialise_stats(self):
15
          Initialises the number of prunes to the statistics dictionary to be logged
16
17
18
          super().initialise_stats()
           self._stats['beta_prunes'] = 0
          self._stats['alpha_prunes'] = 0
20
21
      def find_move(self, board, stop_event):
22
23
24
          Finds the best move for the current board state.
25
26
              board (Board): The current board state.
27
              stop_event (threading.Event): Event used to kill search from an
28
      external class.
29
3.0
          self.initialise_stats()
          best_score, best_move = self.search(board, self._max_depth, -Score.
31
      INFINITE, Score.INFINITE, stop_event)
32
          if self._verbose:
              self.print_stats(best_score, best_move)
34
35
36
           self._callback(best_move)
3.7
      def search(self, board, depth, alpha, beta, stop_event, hint=None):
38
39
```

```
Recursively DFS through minimax tree while pruning branches using the
      alpha and beta bounds.
41
           Args:
               board (Board): The current board state.
43
44
               depth (int): The current search depth.
               alpha (int): The upper bound value.
45
               beta (int): The lower bound value.
46
               stop_event (threading.Event): Event used to kill search from an
47
      external class.
48
49
           Returns:
          tuple[int, Move]: The best score and the best move found.
50
51
           if (base_case := super().search(board, depth, stop_event)):
52
               return base_case
53
54
55
          best_move = None
56
           # Blue is the maximising player
57
           if board.get_active_colour() == Colour.BLUE:
58
               max_score = -Score.INFINITE
5.9
60
               for move in orderer.get_moves(board, hint):
6.1
                   laser_result = board.apply_move(move)
62
                   new_score = self.search(board, depth - 1, alpha, beta, stop_event,
63
       laser_result.pieces_on_trajectory)[0]
                   if new_score > max_score:
65
                       max_score = new_score
66
67
                       best_move = move
68
69
                   board.undo_move(move, laser_result)
70
                   alpha = max(alpha, max_score)
71
72
                   if beta <= alpha:</pre>
73
                       self._stats['alpha_prunes'] += 1
7.4
76
               return max_score, best_move
77
78
          else:
7.9
               min_score = Score.INFINITE
80
81
               for move in orderer.get_moves(board, hint):
82
                   laser_result = board.apply_move(move)
83
                   new_score = self.search(board, depth - 1, alpha, beta, stop_event,
84
       laser_result.pieces_on_trajectory)[0]
85
                   if new_score < min_score:</pre>
86
87
                       min_score = new_score
                       best_move = move
88
89
                   board.undo_move(move, laser_result)
91
                   beta = min(beta, min_score)
92
                   if beta <= alpha:</pre>
93
                       self._stats['beta_prunes'] += 1
94
95
                       break
96
               return min_score, best_move
97
```

```
99 class ABNegamaxCPU(BaseCPU):
       def __init__(self, max_depth, callback, verbose=True):
100
           super().__init__(callback, verbose)
101
           self._max_depth = max_depth
103
       def initialise_stats(self):
104
            """Initialises the statistics for the search."""
105
106
           super().initialise_stats()
           self._stats['beta_prunes'] = 0
107
108
109
       def find_move(self, board, stop_event):
           """Finds the best move for the current board state.
111
112
           Args:
               board (Board): The current board state.
113
               stop_event (threading.Event): The event to signal stopping the search.
114
116
           self.initialise_stats()
           best_score, best_move = self.search(board, self._max_depth, -Score.
       INFINITE, Score.INFINITE, stop_event)
118
           if self._verbose:
119
                self.print_stats(best_score, best_move)
120
121
           self._callback(best_move)
123
124
       def search(self, board, depth, alpha, beta, stop_event):
           """Searches for the best move using the Alpha-Beta Negamax algorithm.
126
127
           Args:
               board (Board): The current board state.
128
129
                depth (int): The current depth in the game tree.
                alpha (int): The alpha value for pruning.
130
               beta (int): The beta value for pruning.
131
                \mathsf{stop\_event} (threading. Event): The event to signal stopping the search.
132
133
           Returns:
134
               tuple: The best score and the best move found.
136
           if (base_case := super().search(board, depth, stop_event, absolute=True)):
137
               return base_case
138
139
140
           best_move = None
           best_score = alpha
141
142
           for move in board.generate_all_moves(board.get_active_colour()):
143
               laser_result = board.apply_move(move)
144
145
146
               new_score = self.search(board, depth - 1, -beta, -best_score,
       stop_event)[0]
147
               new_score = -new_score
148
               if new_score > best_score:
149
                    best_score = new_score
                    best_move = move
151
                elif new_score == best_score:
152
                    best_move = choice([best_move, move])
153
154
               board.undo_move(move, laser_result)
156
               if best score >= beta:
157
```

```
self._stats['beta_prunes'] += 1
158
160
           return best_score, best_move
 1 from data.states.game.cpu.engines.transposition_table import
       TranspositionTableMixin
 2 from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU, ABNegamaxCPU
 3 from data constants import Score
 5 class IterativeDeepeningMixin:
      def find_move(self, board, stop_event):
           best_move = None
           for depth in range(1, self._max_depth + 1):
 9
               self.initialise_stats()
               self._stats['ID_depth'] = depth
12
13
               best_score , best_move = self.search(board , depth , -Score .INFINITE ,
       Score.INFINITE, stop_event)
14
               if self._verbose:
                   self.print_stats(best_score, best_move)
16
17
           self._callback(best_move)
18
19
20 class IDMinimaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABMinimaxCPU)
2.1
       def initialise_stats(self):
           super().initialise_stats()
22
           self._stats['cache_hits'] = 0
23
24
       def print_stats(self, score, move):
25
           self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
26
       self._stats['nodes'], 3)
           self._stats['cache_entries'] = len(self._table._table)
27
           super().print_stats(score, move)
2.8
30 class IDNegamaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABNegamaxCPU)
31
       def initialise_stats(self):
           super().initialise_stats()
32
           self._stats['cache_hits'] = 0
33
34
       def print_stats(self, score, move):
3.5
           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 from data.states.game.cpu.base import BaseCPU
 2 from data.constants import Score, Colour
 3 from random import choice
 5 class MinimaxCPU(BaseCPU):
       def __init__(self, max_depth, callback, verbose=False):
           super().__init__(callback, verbose)
           self._max_depth = max_depth
1.0
       def find_move(self, board, stop_event):
11
           Finds the best move for the current board state.
```

```
14
           Args:
               board (Board): The current board state.
1.5
               stop_event (threading.Event): Event used to kill search from an
      external class.
17
           self.initialise_stats()
18
          best_score , best_move = self.search(board, self._max_depth, stop_event)
19
20
21
           if self._verbose:
               self.print_stats(best_score, best_move)
22
23
           self._callback(best_move)
24
25
      def search(self, board, depth, stop_event):
26
27
          Recursively DFS through minimax tree with evaluation score.
28
29
3.0
           Args:
               board (Board): The current board state.
31
               depth (int): The current search depth.
32
               stop_event (threading.Event): Event used to kill search from an
3.3
      external class.
          Returns:
3.4
              tuple[int, Move]: The best score and the best move found.
35
36
          if (base_case := super().search(board, depth, stop_event)):
3.7
38
               return base_case
39
          best_move = None
40
41
           # Blue is the maximising player
42
43
           if board.get_active_colour() == Colour.BLUE:
               max_score = -Score.INFINITE
44
45
46
               for move in board.generate_all_moves(Colour.BLUE):
                   laser_result = board.apply_move(move)
47
48
                   new_score = self.search(board, depth - 1, stop_event)[0]
50
51
                   # if depth < self._max_depth:</pre>
52
                        print('DEPTH', depth, new_score, move)
53
54
                   if new_score > max_score:
55
                       max_score = new_score
5.6
57
                       best_move = move
58
                       if new_score == (Score.CHECKMATE + self._max_depth):
59
60
                            board.undo_move(move, laser_result)
6.1
                            return max_score, best_move
62
                   elif new_score == max_score:
63
                       # If evaluated scores are equal, pick a random move
64
                       best_move = choice([best_move, move])
66
                   board.undo_move(move, laser_result)
67
68
6.9
               return max_score, best_move
          else:
71
               min score = Score.INFINITE
72
```

```
for move in board.generate_all_moves(Colour.RED):
74
                   laser_result = board.apply_move(move)
7.5
                   # print('DEPTH', depth, move)
76
                   new_score = self.search(board, depth - 1, stop_event)[0]
7.7
7.8
79
                   if new_score < min_score:</pre>
                       # print('setting new', new_score, move)
8.0
                       min_score = new_score
81
                       best_move = move
82
83
                       if new_score == (-Score.CHECKMATE - self._max_depth):
                           board.undo_move(move, laser_result)
85
                           return min_score, best_move
86
87
                   elif new_score == min_score:
88
                       best_move = choice([best_move, move])
89
90
                   board.undo_move(move, laser_result)
91
               return min_score, best_move
93
1 from data.constants import Score, Colour, Miscellaneous, MoveType
2 from data.states.game.cpu.base import BaseCPU
3 from data.utils.bitboard_helpers import print_bitboard, is_occupied
4 from random import choice, randint
5 from copy import deepcopy
7 class NegamaxCPU(BaseCPU):
      def __init__(self, max_depth, callback, verbose=False):
9
           super().__init__(callback, verbose)
           self._max_depth = max_depth
      def find_move(self, board, stop_event):
12
           self.initialise_stats()
13
          best_score, best_move = self.search(board, self._max_depth, stop_event)
14
          if self._verbose:
16
               {\tt self.print\_stats(best\_score\,,\ best\_move)}
17
           self._callback(best_move)
19
20
     def search(self, board, depth, stop_event, moves=None):
21
          if (base_case := super().search(board, depth, stop_event, absolute=True)):
22
23
               return base_case
24
25
          best_move = None
          best_score = -Score.INFINITE
27
          for move in board.generate_all_moves(board.get_active_colour()):
28
              laser_result = board.apply_move(move)
29
3.0
31
              new_score = self.search(board, depth - 1, stop_event)[0]
              new_score = -new_score
32
33
               if new_score > best_score:
                   best_score = new_score
35
36
                   best_move = move
               elif new_score == best_score:
37
                   best_move = choice([best_move, move])
38
39
               board.undo_move(move, laser_result)
40
```

```
return best_score, best_move
1 from data.states.game.cpu.base import BaseCPU
2 from data.constants import Colour, Score
4 class SimpleCPU(BaseCPU):
      def __init__(self, callback, verbose=True):
           super().__init__(callback, verbose)
      def find_move(self, board, stop_event=None):
          self.initialise_stats()
9
          best_score, best_move = self.search(board, stop_event)
10
11
          if self._verbose:
12
13
               self.print_stats(best_score, best_move)
14
          self._callback(best_move)
15
16
      def search(self, board, stop_event):
17
18
          if stop_event and stop_event.is_set():
               raise Exception('Thread killed - stopping simple function (SimpleCPU.
      search)')
20
           active_colour = board.bitboards.active_colour
21
          best_score = -Score.INFINITE if active_colour == Colour.BLUE else Score.
22
      INFINITE
          best_move = None
23
24
          for move in board.generate_all_moves(active_colour):
              laser_result = board.apply_move(move)
26
27
              self._stats['nodes'] += 1
28
29
30
               if winner := board.check_win() is not None:
31
                  self.process_win(winner)
               else:
32
33
                   self._stats['leaf_nodes'] += 1
34
3.5
               score = self._evaluator.evaluate(board)
36
              if (active_colour == Colour.BLUE and score > best_score) or (
37
      active_colour == Colour.RED and score < best_score):</pre>
                   best_move = move
38
                   best_score = score
3.9
40
               board.undo_move(move, laser_result)
41
42
          return best_score, best_move
1 from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU, ABNegamaxCPU
2 from data.states.game.cpu.transposition_table import TranspositionTable
4 class TranspositionTableMixin:
      def __init__(self, *args, **kwargs):
           super().__init__(*args, **kwargs)
          self._table = TranspositionTable()
      def find_move(self, *args, **kwargs):
          self._table = TranspositionTable()
1.0
11
           super().find_move(*args, **kwargs)
```

```
def search(self, board, depth, alpha, beta, stop_event, hint=None):
13
14
          Searches transposition table for a cached move before running a full
      search if necessary.
          Caches the searched result.
16
17
          Args:
18
               board (Board): The current board state.
19
20
               depth (int): The current search depth.
               alpha (int): The upper bound value.
21
               beta (int): The lower bound value.
22
23
               stop_event (threading.Event): Event used to kill search from an
      external class.
24
25
          Returns:
              tuple[int, Move]: The best score and the best move found.
26
27
28
          hash = board.to_hash()
          score, move = self._table.get_entry(hash, depth, alpha, beta)
29
30
          if score is not None:
31
               self._stats['cache_hits'] += 1
32
               self._stats['nodes'] += 1
33
34
35
              return score, move
36
           else:
               # If board hash entry not found in cache, run a full search
37
               score, move = super().search(board, depth, alpha, beta, stop_event,
      hint)
              self._table.insert_entry(score, move, hash, depth, alpha, beta)
3.9
              return score, move
41
42
43 class TTMinimaxCPU(TranspositionTableMixin, ABMinimaxCPU):
      def initialise_stats(self):
44
45
          Initialises cache statistics to be logged.
46
47
          super().initialise_stats()
          self._stats['cache_hits'] = 0
49
50
      def print_stats(self, score, move):
51
52
53
          Logs the statistics for the search.
54
5.5
          Args:
              score (int): The best score found.
56
              move (Move): The best move found.
57
5.8
59
          # Calculate number of cached entries retrieved as a percentage of all
      nodes
          self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
      self._stats['nodes'], 3)
          self._stats['cache_entries'] = len(self._table._table)
6.1
           super().print_stats(score, move)
63
64 class TTNegamaxCPU(TranspositionTableMixin, ABNegamaxCPU):
      """Negamax CPU engine with transposition table support."""
65
66
67
      def initialise_stats(self):
          """Initialises the statistics for the search."""
68
          super().initialise_stats()
69
```

```
self._stats['cache_hits'] = 0
70
71
      def print_stats(self, score, move):
72
           """Prints the statistics for the search.
73
74
7.5
              score (int): The best score found.
76
               move (Move): The best move found.
7.7
7.8
           self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
79
      self._stats['nodes'], 3)
           self._stats['cache_entries'] = len(self._table._table)
           super().print_stats(score, move)
81
1 from data.states.game.cpu.engines.simple import SimpleCPU
2 from data.states.game.cpu.engines.negamax import NegamaxCPU
3 from data.states.game.cpu.engines.minimax import MinimaxCPU
{\tt 4 from data.states.game.cpu.engines.alpha\_beta import ABMinimaxCPU, ABNegamaxCPU}
5 from data.states.game.cpu.engines.iterative_deepening import IDMinimaxCPU,
      IDNegamaxCPU
{\tiny \texttt{6}} \quad \textbf{from} \quad \textbf{data.states.game.cpu.engines.transposition\_table} \quad \textbf{import} \quad \textbf{TTMinimaxCPU} \, ,
      TTNegamaxCPU
1 import pygame
2 from data.constants import GameEventType, MoveType, StatusText, Miscellaneous
3 from data.utils import bitboard_helpers as bb_helpers
4 from data.states.game.components.move import Move
5 from data.managers.logs import initialise_logger
7 logger = initialise_logger(__name__)
9 class GameController:
     def __init__(self, model, view, win_view, pause_view, to_menu, to_review,
1.0
      to_new_game):
          self._model = model
11
           self._view = view
12
           self._win_view = win_view
           self._pause_view = pause_view
14
15
           self._to_menu = to_menu
           self._to_review = to_review
17
18
           self._to_new_game = to_new_game
19
           self._view.initialise_timers()
20
21
           self._win_view.set_win_type('CAPTURE')
22
23
     def cleanup(self, next):
           Handles game quit, either leaving to main menu or restarting a new game.
25
26
27
           Args:
           next (str): New state to switch to.
28
29
           self._model.kill_thread()
30
3.1
           if next == 'menu':
               self._to_menu()
33
34
           elif next == 'game':
35
               self._to_new_game()
           elif next == 'review':
36
37
               self._to_review()
3.8
```

```
def make_move(self, move):
39
           0.00
40
           Handles player move.
41
43
          Args:
           move (Move): Move to make.
44
45
           self._model.make_move(move)
46
           self._view.set_overlay_coords([], None)
47
48
           if self._model.states['CPU_ENABLED']:
49
50
               self._model.make_cpu_move()
51
     def handle_pause_event(self, event):
52
53
           Processes events when game is paused.
54
55
56
           Args:
              event (GameEventType): Event to process.
57
           Raises:
59
           Exception: If event type is unrecognised.
60
61
           game_event = self._pause_view.convert_mouse_pos(event)
62
63
           if game_event is None:
64
65
               return
66
67
           match game_event.type:
               {\tt case \ GameEventType.PAUSE\_CLICK:}
68
69
                   self._model.toggle_paused()
7.0
71
               case GameEventType.MENU_CLICK:
                   self.cleanup('menu')
72
7.3
74
               case _:
                  raise Exception('Unhandled event type (GameController.handle_event
75
      ) ' )
      def handle_winner_event(self, event):
7.7
78
           Processes events when game is over.
79
8.0
81
           Args:
              event (GameEventType): Event to process.
82
8.3
           Exception: If event type is unrecognised.
85
86
87
           game_event = self._win_view.convert_mouse_pos(event)
88
89
           if game_event is None:
90
               return
9.1
           match game_event.type:
              case GameEventType.MENU_CLICK:
93
                   self.cleanup('menu')
94
95
96
               {\tt case \ GameEventType.GAME\_CLICK:}
97
                   self.cleanup('game')
98
                   return
99
```

```
case GameEventType.REVIEW_CLICK:
101
                     self.cleanup('review')
102
                case _:
                    raise Exception('Unhandled event type (GameController.handle_event
       ) ' )
106
       def handle_game_widget_event(self, event):
107
108
            Processes events for game GUI widgets.
109
110
            Args:
                event (GameEventType): Event to process.
112
            Raises:
114
                Exception: If event type is unrecognised.
115
116
            Returns:
            CustomEvent | None: A widget event.
118
119
            widget_event = self._view.process_widget_event(event)
120
121
           if widget_event is None:
123
                return None
124
125
            match widget_event.type:
126
                {\tt case \ GameEventType.ROTATE\_PIECE:}
                     src_coords = self._view.get_selected_coords()
128
                     if src_coords is None:
                        logger.info('None square selected')
130
131
                         return
132
                    move = Move.instance_from_coords(MoveType.ROTATE, src_coords,
       src_coords, rotation_direction=widget_event.rotation_direction)
                    self.make_move(move)
                case GameEventType.RESIGN_CLICK:
136
                     self._model.set_winner(self._model.states['ACTIVE_COLOUR'].
137
       get_flipped_colour())
                     self._view.handle_game_end(play_sfx=False)
                     self._win_view.set_win_type('RESIGN')
139
140
                case GameEventType.DRAW_CLICK:
141
                     self._model.set_winner(Miscellaneous.DRAW)
142
                     self._view.handle_game_end(play_sfx=False)
143
                     self._win_view.set_win_type('DRAW')
144
145
146
                {\tt case \ GameEventType.TIMER\_END:}
                    if self._model.states['TIME_ENABLED']:
147
                         self._model.set_winner(widget_event.active_colour.
148
       get_flipped_colour())
                         self._win_view.set_win_type('TIME')
149
                         self._view.handle_game_end(play_sfx=False)
150
151
                {\tt case} \quad {\tt GameEventType} \; . \; {\tt MENU\_CLICK} \; : \\
152
                     self.cleanup('menu')
154
                case GameEventType.HELP_CLICK:
                     self._view.add_help_screen()
157
```

```
case GameEventType.TUTORIAL_CLICK:
158
                    self._view.add_tutorial_screen()
160
                    raise Exception('Unhandled event type (GameController.handle_event
       ) ')
163
           return widget_event.type
164
165
       def check_cpu(self):
166
167
168
           Checks if CPU calculations are finished every frame.
169
           if self._model.states['CPU_ENABLED'] and self._model.states['AWAITING_CPU'
170
       ] is False:
               self._model.check_cpu()
172
173
       def handle_game_event(self, event):
174
           Processes Pygame events for main game.
           Args:
                event (pygame.Event): If event type is unrecognised.
178
180
           Raises:
              Exception: If event type is unrecognised.
181
182
183
           # Pass event for widgets to process
           widget_event = self.handle_game_widget_event(event)
184
185
186
           if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
       KEYDOWN]:
187
               game_event = self._view.convert_mouse_pos(event)
188
189
                else:
                    game_event = None
190
191
                if game_event is None:
192
                    if widget_event is None:
193
                        if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
194
                            # If user releases mouse click not on a widget
195
                            self._view.remove_help_screen()
196
                            self._view.remove_tutorial_screen()
197
198
                        if event.type == pygame.MOUSEBUTTONUP:
                            # If user releases mouse click on neither a widget or
199
       board
                            self._view.set_overlay_coords(None, None)
201
202
                    return
203
                match game_event.type:
204
205
                    {\tt case \ GameEventType.BOARD\_CLICK:}
                        if self._model.states['AWAITING_CPU']:
206
                            return
207
208
                        clicked_coords = game_event.coords
209
                        clicked_bitboard = bb_helpers.coords_to_bitboard(
210
       clicked_coords)
                        selected_coords = self._view.get_selected_coords()
211
212
                        if selected_coords:
213
                            if clicked_coords == selected_coords:
214
```

```
215
                                # If clicking on an already selected square, start
       dragging piece on that square
                                self._view.set_dragged_piece(*self._model.
       get_piece_info(clicked_bitboard))
217
218
                            selected_bitboard = bb_helpers.coords_to_bitboard(
219
       selected coords)
                            available_bitboard = self._model.get_available_moves(
       selected_bitboard)
221
222
                            if bb_helpers.is_occupied(clicked_bitboard,
       available_bitboard):
                                # If the newly clicked square is not the same as the
       old one, and is an empty surrounding square, make a move
                                move = Move.instance_from_coords(MoveType.MOVE,
       selected_coords , clicked_coords)
                                self.make_move(move)
                            else:
                                # If the newly clicked square is not the same as the
227
       old one, but is an invalid square, unselect the currently selected square
                                self._view.set_overlay_coords(None, None)
228
229
                        # Select hovered square if it is same as active colour
230
                        elif self._model.is_selectable(clicked_bitboard):
231
                            available_bitboard = self._model.get_available_moves(
232
       clicked bitboard)
                            self._view.set_overlay_coords(bb_helpers.
       bitboard_to_coords_list(available_bitboard), clicked_coords)
                            self._view.set_dragged_piece(*self._model.get_piece_info(
       clicked_bitboard))
236
                    case GameEventType.PIECE_DROP:
                        hovered_coords = game_event.coords
237
238
                        # if piece is dropped onto the board
239
                        if hovered_coords:
240
                            hovered_bitboard = bb_helpers.coords_to_bitboard(
241
       hovered_coords)
                            selected_coords = self._view.get_selected_coords()
242
243
                            selected_bitboard = bb_helpers.coords_to_bitboard(
       selected_coords)
                            available_bitboard = self._model.get_available_moves(
244
       selected bitboard)
245
                            if bb_helpers.is_occupied(hovered_bitboard,
246
       available_bitboard):
                                # Make a move if mouse is hovered over an empty
247
       surrounding square
248
                                move = Move.instance_from_coords(MoveType.MOVE,
       selected_coords , hovered_coords)
249
                                self.make_move(move)
                        if game_event.remove_overlay:
251
                            self._view.set_overlay_coords(None, None)
252
                        self._view.remove_dragged_piece()
                    case _:
                        raise Exception ('Unhandled event type (GameController.
257
       handle_event)', game_event.type)
```

258

```
259
       def handle_event(self, event):
260
           Passe a Pygame event to the correct handling function according to the
261
       game state.
262
263
           Args:
           event (pygame.Event): Event to process.
264
265
           if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
266
       MOUSEMOTION, pygame.KEYDOWN]:
               if self._model.states['PAUSED']:
267
268
                   self.handle_pause_event(event)
               elif self._model.states['WINNER'] is not None:
269
                   self.handle_winner_event(event)
270
271
               else:
                   self.handle_game_event(event)
272
273
274
           if event.type == pygame.KEYDOWN:
               if event.key == pygame.K_ESCAPE:
275
                    self._model.toggle_paused()
276
               elif event.key == pygame.K_l:
277
                    logger.info('\nSTOPPING CPU')
278
                    self._model._cpu_thread.stop_cpu() #temp
 1 from data.states.game.components.fen_parser import encode_fen_string
 2 from data.constants import Colour, GameEventType, EMPTY_BB
 3 from data.states.game.widget_dict import GAME_WIDGETS
 4 from data.states.game.cpu.cpu_thread import CPUThread
 5 from data.states.game.cpu.engines import ABMinimaxCPU
 6 from data.components.custom_event import CustomEvent
 7 from data.utils.bitboard_helpers import is_occupied
 8 from data.states.game.components.board import Board
 9 from data.utils import input_helpers as ip_helpers
10 from data.states.game.components.move import Move
11 from data.managers.logs import initialise_logger
12
13 logger = initialise_logger(__name__)
14
15 class GameModel:
      def __init__(self, game_config):
           self._listeners = {
17
                'game': [],
18
               'win': [],
19
               'pause': [],
20
           }
21
           self._board = Board(fen_string=game_config['FEN_STRING'])
22
23
               'CPU_ENABLED': game_config['CPU_ENABLED'],
25
               'CPU_DEPTH': game_config['CPU_DEPTH'],
26
               'AWAITING_CPU': False,
27
               'WINNER': None,
28
               'PAUSED': False,
29
               'ACTIVE_COLOUR': game_config['COLOUR'],
30
               'TIME_ENABLED': game_config['TIME_ENABLED'],
3.1
               'TIME': game_config['TIME'],
               'START_FEN_STRING': game_config['FEN_STRING'],
33
34
               'MOVES': [],
               'ZOBRIST_KEYS': []
35
           }
36
37
           self._cpu = ABMinimaxCPU(self.states['CPU_DEPTH'], self.cpu_callback,
3.8
```

```
verbose=False)
           self._cpu_thread = CPUThread(self._cpu)
39
          self._cpu_thread.start()
40
          self._cpu_move = None
41
42
           logger.info(f'Initialising CPU depth of {self.states['CPU_DEPTH']}')
43
44
      def register_listener(self, listener, parent_class):
45
46
          Registers listener method of another MVC class.
47
48
49
          Args:
              listener (callable): Listener callback function.
50
              parent_class (str): Class name.
5.1
52
           self._listeners[parent_class].append(listener)
53
54
55
      def alert_listeners(self, event):
56
          Alerts all registered classes of an event by calling their listener
      function.
5.8
59
          Args:
              event (GameEventType): Event to pass as argument.
6.0
61
62
          Exception: If an unrecgonised event tries to be passed onto listeners.
63
64
          for parent_class, listeners in self._listeners.items():
65
66
               match event.type:
67
                   case GameEventType.UPDATE_PIECES:
                       if parent_class in 'game':
68
                           for listener in listeners: listener(event)
69
70
                   case GameEventType.SET_LASER:
71
72
                       if parent_class == 'game':
                           for listener in listeners: listener(event)
7.4
                   case GameEventType.PAUSE_CLICK:
                       if parent_class in ['pause', 'game']:
76
                           for listener in listeners:
7.7
                               listener(event)
78
7.9
80
                       raise Exception ('Unhandled event type (GameModel.
81
      alert listeners)')
      def set_winner(self, colour=None):
83
84
85
          Sets winner.
86
87
              colour (Colour, optional): Describes winnner colour, or draw. Defaults
88
       to None.
          self.states['WINNER'] = colour
90
91
      def toggle_paused(self):
92
93
          Toggles pause screen, and alerts pause view.
94
95
          self.states['PAUSED'] = not self.states['PAUSED']
96
```

```
game_event = CustomEvent.create_event(GameEventType.PAUSE_CLICK)
97
           self.alert_listeners(game_event)
98
99
       def get_terminal_move(self):
100
101
           Debugging method for inputting a move from the terminal.
102
104
           Returns:
              Move: Parsed move.
           while True:
107
108
              try:
                   move_type = ip_helpers.parse_move_type(input('Input move type (m/r
109
      ): '))
                   src_square = ip_helpers.parse_notation(input("From: "))
                   dest_square = ip_helpers.parse_notation(input("To: "))
                   rotation = ip_helpers.parse_rotation(input("Enter rotation (a/b/c/
      d): "))
                   return Move.instance_from_notation(move_type, src_square,
       dest_square, rotation)
               except ValueError as error:
114
                   logger.warning('Input error (Board.get_move): ' + str(error))
115
116
       def make_move(self, move):
118
           Takes a Move object and applies it to the board.
119
120
121
           Args:
           move (Move): Move to apply.
123
           colour = self._board.bitboards.get_colour_on(move.src)
           piece = self._board.bitboards.get_piece_on(move.src, colour)
126
           # Apply move and get results of laser trajectory
           laser_result = self._board.apply_move(move, add_hash=True)
128
           self.alert_listeners(CustomEvent.create_event(GameEventType.SET_LASER,
      laser_result=laser_result))
130
           # Sets new active colour and checks for a win
131
           self.states['ACTIVE_COLOUR'] = self._board.get_active_colour()
133
           self.set_winner(self._board.check_win())
134
           move_notation = move.to_notation(colour, piece, laser_result.
135
       hit_square_bitboard)
136
           137
        move_notation=move_notation))
138
           # Adds move to move history list for review screen
139
140
           self.states['MOVES'].append({
               'time': [
141
                   Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
142
                   Colour.RED: GAME_WIDGETS['red_timer'].get_time()
143
144
               'move': move_notation,
               'laserResult': laser_result
146
          1)
147
148
149
       def make_cpu_move(self):
150
           Starts CPU calculations on the separate thread.
151
152
```

```
self.states['AWAITING_CPU'] = True
153
            self._cpu_thread.start_cpu(self.get_board())
154
       def cpu_callback(self, move):
156
157
           Callback function passed to CPU thread. Called when CPU stops processing.
158
159
160
           Args:
           move (Move): Move that CPU found.
161
            if self.states['WINNER'] is None:
163
                # CPU move passed back to main threadby reassigning variable
164
                self._cpu_move = move
                self.states['AWAITING_CPU'] = False
166
167
       def check_cpu(self):
168
169
170
           Constantly checks if CPU calculations are finished, so that make_move can
       be run on the main thread.
171
           0.000
            if self._cpu_move is not None:
172
                self.make_move(self._cpu_move)
173
                self._cpu_move = None
174
175
       def kill_thread(self):
176
177
           Interrupt and kill CPU thread.
178
179
           self._cpu_thread.kill_thread()
180
           self.states['AWAITING_CPU'] = False
181
182
       def is_selectable(self, bitboard):
183
184
           Checks if square is occupied by a piece of the current active colour.
185
186
187
               bitboard (int): Bitboard representing single square.
188
189
190
               bool: True if square is occupied by a piece of the current active
191
       colour. False if not.
192
           return is_occupied(self._board.bitboards.combined_colour_bitboards[self.
193
       states['ACTIVE_COLOUR']], bitboard)
194
       def get_available_moves(self, bitboard):
195
           Gets all surrounding empty squares. Used for drawing overlay.
197
198
199
           Args:
              bitboard (int): Bitboard representing single center square.
200
201
           Returns:
202
              int: Bitboard representing all empty surrounding squares.
203
           if (bitboard & self._board.get_all_active_pieces()) != EMPTY_BB:
205
                return self._board.get_valid_squares(bitboard)
206
207
           return EMPTY BB
208
209
       def get_piece_list(self):
210
211
```

```
Returns:
            list[Piece, ...]: Array of all pieces on the board.
214
            return self._board.get_piece_list()
215
216
       def get_piece_info(self, bitboard):
218
            Args:
219
               bitboard (int): Square containing piece.
221
           Returns:
            tuple[Colour, Rotation, Piece]: Piece information.
223
224
            colour = self._board.bitboards.get_colour_on(bitboard)
225
            rotation = self._board.bitboards.get_rotation_on(bitboard)
            piece = self._board.bitboards.get_piece_on(bitboard, colour)
228
            return (piece, colour, rotation)
       def get_fen_string(self):
230
           return encode_fen_string(self._board.bitboards)
231
       def get_board(self):
233
           return self._board
 1 import pygame
 2 from data.constants import GameEventType, Colour, StatusText, Miscellaneous,
       ShaderType
 {\tt 3} \  \  \, \textbf{from} \  \  \, \textbf{data.states.game.components.overlay\_draw} \  \  \, \textbf{import} \  \  \, \textbf{OverlayDraw}
 4 from data.states.game.components.capture_draw import CaptureDraw
 5 from data.states.game.components.piece_group import PieceGroup
 6 from data.states.game.components.laser_draw import LaserDraw
 7 from data.states.game.components.father import DragAndDrop
 8 from data.utils.bitboard_helpers import bitboard_to_coords
 9 from data.utils.board_helpers import screen_pos_to_coords
10 from data.states.game.widget_dict import GAME_WIDGETS
11 from data.components.custom_event import CustomEvent
 12 from data.components.widget_group import WidgetGroup
13 from data.managers.window import window
14 from data.managers.audio import audio
15 from data.assets import SFX
16
17 class GameView:
       def __init__(self, model):
18
            self._model = model
19
20
            self._hide_pieces = False
           self._selected_coords = None
21
            self._event_to_func_map = {
22
                GameEventType.UPDATE_PIECES: self.handle_update_pieces,
                GameEventType.SET_LASER: self.handle_set_laser,
24
                GameEventType.PAUSE_CLICK: self.handle_pause,
25
           }
26
27
            # Register model event handling with process_model_event()
28
            self._model.register_listener(self.process_model_event, 'game')
29
3.0
            # Initialise WidgetGroup with map of widgets
            self._widget_group = WidgetGroup(GAME_WIDGETS)
32
            self._widget_group.handle_resize(window.size)
33
            self.initialise_widgets()
34
3.5
            self._laser_draw = LaserDraw(self.board_position, self.board_size)
36
            self._overlay_draw = OverlayDraw(self.board_position, self.board_size)
3.7
```

```
self._drag_and_drop = DragAndDrop(self.board_position, self.board_size)
           self._capture_draw = CaptureDraw(self.board_position, self.board_size)
39
          self._piece_group = PieceGroup()
40
           self.handle_update_pieces()
41
42
           self.set_status_text(StatusText.PLAYER_MOVE)
43
44
45
      @property
      def board_position(self):
46
          return GAME_WIDGETS['chessboard'].position
47
48
49
      @property
      def board_size(self):
50
          return GAME_WIDGETS['chessboard'].size
51
52
      @property
53
      def square_size(self):
54
55
          return self.board_size[0] / 10
56
      def initialise_widgets(self):
57
58
          Run methods on widgets stored in GAME_WIDGETS dictionary to reset them.
5.9
60
          GAME_WIDGETS['move_list'].reset_move_list()
6.1
          GAME_WIDGETS['move_list'].kill()
62
           GAME_WIDGETS['help'].kill()
63
          GAME_WIDGETS['tutorial'].kill()
64
65
          GAME_WIDGETS['scroll_area'].set_image()
66
6.7
68
           GAME_WIDGETS['chessboard'].refresh_board()
69
           GAME_WIDGETS['blue_piece_display'].reset_piece_list()
           GAME_WIDGETS['red_piece_display'].reset_piece_list()
71
72
      def set_status_text(self, status):
73
74
          Sets text on status text widget.
7.5
76
7.7
          Args:
               status (StatusText): The game stage for which text should be displayed
78
       for.
7.9
80
          match status:
              case StatusText.PLAYER_MOVE:
81
                   GAME_WIDGETS['status_text'].set_text(f"{self._model.states['
82
      ACTIVE_COLOUR'].name}'s turn to move")
              case StatusText.CPU_MOVE:
83
                   GAME_WIDGETS['status_text'].set_text(f"CPU calculating a crazy
84
      move...")
               case StatusText.WIN:
85
                   if self._model.states['WINNER'] == Miscellaneous.DRAW:
86
                       GAME_WIDGETS['status_text'].set_text(f"Game is a draw! Boring
87
      . . . " )
                      GAME_WIDGETS['status_text'].set_text(f"{self._model.states['
89
      WINNER'].name} won!")
               case StatusText.DRAW:
                   GAME_WIDGETS['status_text'].set_text(f"Game is a draw! Boring...")
91
92
      def handle_resize(self):
93
94
```

```
Handles resizing of the window.
95
96
           self._overlay_draw.handle_resize(self.board_position, self.board_size)
97
            self._capture_draw.handle_resize(self.board_position, self.board_size)
            self._piece_group.handle_resize(self.board_position, self.board_size)
99
100
            self._laser_draw.handle_resize(self.board_position, self.board_size)
            self._laser_draw.handle_resize(self.board_position, self.board_size)
101
           self._widget_group.handle_resize(window.size)
102
104
            if self._laser_draw.firing:
                self.update_laser_mask()
106
       def handle_update_pieces(self, event=None):
107
108
            Callback function to update pieces after move.
           Args:
111
               event (GameEventType, optional): If updating pieces after player move,
        event contains move information. Defaults to None.
               toggle_timers (bool, optional): Toggle timers on and off for new
       active colour. Defaults to True.
114
            piece_list = self._model.get_piece_list()
           \verb|self._piece_group.initialise_pieces(piece_list, self.board_position, self.|\\
116
       board_size)
118
            if event:
                GAME_WIDGETS['move_list'].append_to_move_list(event.move_notation)
                GAME_WIDGETS['scroll_area'].set_image()
120
                audio.play_sfx(SFX['piece_move'])
            if self._model.states['ACTIVE_COLOUR'] == Colour.BLUE:
123
124
                self.set_status_text(StatusText.PLAYER_MOVE)
            elif self._model.states['CPU_ENABLED'] is False:
                self.set_status_text(StatusText.PLAYER_MOVE)
            else:
127
               self.set_status_text(StatusText.CPU_MOVE)
128
            if self._model.states['TIME_ENABLED']:
                self.toggle_timer(self._model.states['ACTIVE_COLOUR'], True)
131
                {\tt self.toggle\_timer(self.\_model.states['ACTIVE\_COLOUR']}.
       get_flipped_colour(), False)
            if self._model.states['WINNER'] is not None:
               self.handle_game_end()
135
136
       def handle_game_end(self, play_sfx=True):
137
            self.toggle_timer(self._model.states['ACTIVE_COLOUR'], False)
138
           self.toggle_timer(self._model.states['ACTIVE_COLOUR'].get_flipped_colour()
139
       , False)
140
            if self._model.states['WINNER'] == Miscellaneous.DRAW:
141
               self.set_status_text(StatusText.DRAW)
142
143
            else:
                self.set_status_text(StatusText.WIN)
145
146
            if play_sfx:
                audio.play_sfx(SFX['sphinx_destroy_1'])
147
                audio.play_sfx(SFX['sphinx_destroy_2'])
audio.play_sfx(SFX['sphinx_destroy_3'])
148
149
150
       def handle_set_laser(self, event):
151
```

```
0.00
            Callback function to draw laser after move.
154
           event (GameEventType): Contains laser trajectory information.
156
157
158
            laser_result = event.laser_result
159
160
            # If laser has hit a piece
161
            if laser_result.hit_square_bitboard:
                coords_to_remove = bitboard_to_coords(laser_result.hit_square_bitboard
       )
                self._piece_group.remove_piece(coords_to_remove)
164
                if laser_result.piece_colour == Colour.BLUE:
165
                    {\tt GAME\_WIDGETS['red\_piece\_display'].add\_piece(laser\_result.piece\_hit)}
166
       )
167
                elif laser_result.piece_colour == Colour.RED:
                    {\tt GAME\_WIDGETS['blue\_piece\_display'].add\_piece(laser\_result.}
168
       piece_hit)
169
                # Draw piece capture GFX
170
                self._capture_draw.add_capture(
171
                    laser_result.piece_hit,
                    laser_result.piece_colour,
174
                    laser_result.piece_rotation,
                    coords_to_remove,
176
                    laser_result.laser_path[0][0],
                    self._model.states['ACTIVE_COLOUR']
178
179
            self._laser_draw.add_laser(laser_result, self._model.states['ACTIVE_COLOUR
180
       '1)
            self.update_laser_mask()
181
182
       def handle_pause(self, event=None):
183
184
            Callback function for pausing timer.
185
186
187
           Args:
           event (None): Event argument not used.
188
189
            is_active = not(self._model.states['PAUSED'])
190
            self.toggle_timer(self._model.states['ACTIVE_COLOUR'], is_active)
191
192
193
       def initialise timers(self):
           Initialises both timers with the correct amount of time and starts the
       timer for the active colour.
196
            if self._model.states['TIME_ENABLED']:
197
                GAME_WIDGETS['blue_timer'].set_time(self._model.states['TIME'] * 60 *
198
       1000)
                GAME_WIDGETS['red_timer'].set_time(self._model.states['TIME'] * 60 *
       1000)
           else:
200
                GAME_WIDGETS['blue_timer'].kill()
201
                GAME_WIDGETS['red_timer'].kill()
202
            self.toggle_timer(self._model.states['ACTIVE_COLOUR'], True)
204
205
       def toggle_timer(self, colour, is_active):
206
```

```
0.00
207
208
           Stops or resumes timer.
210
           Args:
               colour (Colour): Timer to toggle.
211
212
               is_active (bool): Whether to pause or resume timer.
213
           if colour == Colour.BLUE:
214
                GAME_WIDGETS['blue_timer'].set_active(is_active)
215
            elif colour == Colour.RED:
216
                GAME_WIDGETS['red_timer'].set_active(is_active)
217
218
       def update_laser_mask(self):
219
220
           Uses pygame.mask to create a mask for the pieces.
221
           Used for occluding the ray shader.
223
224
            temp_surface = pygame.Surface(window.size, pygame.SRCALPHA)
           self._piece_group.draw(temp_surface)
           mask = pygame.mask.from_surface(temp_surface, threshold=127)
226
           mask_surface = mask.to_surface(unsetcolor=(0, 0, 0, 255), setcolor=(255,
227
       0, 0, 255))
           window.set_apply_arguments(ShaderType.RAYS, occlusion=mask_surface)
229
230
       def draw(self):
231
232
233
           Draws GUI and pieces onto the screen.
234
           self._widget_group.update()
235
236
           self._capture_draw.update()
237
238
            self._widget_group.draw()
           self._overlay_draw.draw(window.screen)
239
240
           if self._hide_pieces is False:
241
               self._piece_group.draw(window.screen)
242
243
            self._laser_draw.draw(window.screen)
            self._drag_and_drop.draw(window.screen)
245
246
            self._capture_draw.draw(window.screen)
247
248
       def process_model_event(self, event):
249
            Registered listener function for handling GameModel events.
251
           Each event is mapped to a callback function, and the appropriate one is run
253
           Args:
254
               event (GameEventType): Game event to process.
256
           Raises:
               KeyError: If an unrecgonised event type is passed as the argument.
257
258
259
           try:
               self._event_to_func_map.get(event.type)(event)
260
261
            except:
               raise KeyError ('Event type not recognized in Game View (Game View.
262
       process_model_event):', event.type)
263
       def set_overlay_coords(self, available_coords_list, selected_coord):
264
265
```

```
Set board coordinates for potential moves overlay.
266
267
268
           Args:
               available_coords_list (list[tuple[int, int]], ...): Array of
       coordinates
               selected_coord (list[int, int]): Coordinates of selected piece.
271
           self._selected_coords = selected_coord
272
273
           self._overlay_draw.set_selected_coords(selected_coord)
274
           self._overlay_draw.set_available_coords(available_coords_list)
275
276
       def get_selected_coords(self):
277
           return self._selected_coords
278
279
       def set_dragged_piece(self, piece, colour, rotation):
280
           Passes information of the dragged piece to the dragging drawing class.
281
282
283
           Args:
               piece (Piece): Piece type of dragged piece.
284
                colour (Colour): Colour of dragged piece.
285
               rotation (Rotation): Rotation of dragged piece.
286
287
           self._drag_and_drop.set_dragged_piece(piece, colour, rotation)
288
289
290
       def remove_dragged_piece(self):
291
292
           Stops drawing dragged piece when user lets go of piece.
293
           self._drag_and_drop.remove_dragged_piece()
294
295
       def convert_mouse_pos(self, event):
296
297
           Passes information of what mouse cursor is interacting with to a
298
       GameController object.
299
300
           Args:
               event (pygame.Event): Mouse event to process.
301
302
303
           Returns:
               CustomEvent | None: Contains information what mouse is doing.
304
305
           clicked_coords = screen_pos_to_coords(event.pos, self.board_position, self
306
       .board_size)
307
           if event.type == pygame.MOUSEBUTTONDOWN:
308
                if clicked_coords:
                   return CustomEvent.create_event(GameEventType.BOARD_CLICK, coords=
310
       clicked_coords)
311
312
                else:
                    return None
313
314
           elif event.type == pygame.MOUSEBUTTONUP:
315
                if self._drag_and_drop.dragged_sprite:
                    piece, colour, rotation = self._drag_and_drop.get_dragged_info()
317
                    piece_dragged = self._drag_and_drop.remove_dragged_piece()
318
                    return CustomEvent.create_event(GameEventType.PIECE_DROP, coords=
319
       clicked_coords, piece=piece, colour=colour, rotation=rotation, remove_overlay=
       piece_dragged)
320
       def add_help_screen(self):
321
```

```
0.00
322
           Draw help overlay when player clicks on the help button.
323
324
           self._widget_group.add(GAME_WIDGETS['help'])
325
           self._widget_group.handle_resize(window.size)
326
327
328
       def add_tutorial_screen(self):
329
           Draw tutorial overlay when player clicks on the tutorial button.
330
331
           self._widget_group.add(GAME_WIDGETS['tutorial'])
332
333
           self._widget_group.handle_resize(window.size)
           self._hide_pieces = True
334
335
       def remove_help_screen(self):
336
           GAME_WIDGETS['help'].kill()
337
338
339
       def remove_tutorial_screen(self):
           GAME_WIDGETS['tutorial'].kill()
340
           self._hide_pieces = False
341
342
       def process_widget_event(self, event):
343
344
           Passes Pygame event to WidgetGroup to allow individual widgets to process
345
       events.
346
347
           Args:
348
               event (pygame.Event): Event to process.
349
           Returns:
350
           CustomEvent | None: A widget event.
352
353
           return self._widget_group.process_event(event)
 1 import pygame
 2 from data.states.game.widget_dict import PAUSE_WIDGETS
 {\tt 3} from data.constants import GameEventType, PAUSE_COLOUR
 4 from data.components.widget_group import WidgetGroup
 5 from data.managers.window import window
 6 from data.managers.audio import audio
 8 class PauseView:
       def __init__(self, model):
           self._model = model
1.0
 11
           self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
12
           self._screen_overlay.fill(PAUSE_COLOUR)
13
 14
           self._widget_group = WidgetGroup(PAUSE_WIDGETS)
15
           self._widget_group.handle_resize(window.size)
16
17
           self._model.register_listener(self.process_model_event, 'pause')
18
19
           self. event to func map = {
20
                GameEventType.PAUSE_CLICK: self.handle_pause_click
21
           }
23
24
           self.states = {
               'PAUSED': False
25
26
27
       def handle_pause_click(self, event):
28
```

```
self.states['PAUSED'] = not self.states['PAUSED']
          if self.states['PAUSED']:
3.1
              audio.pause_sfx()
          else:
33
3.4
               audio.unpause_sfx()
35
      def handle_resize(self):
36
           self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
3.7
           self._screen_overlay.fill(PAUSE_COLOUR)
38
          self._widget_group.handle_resize(window.size)
3.9
40
      def draw(self):
41
          if self.states['PAUSED']:
42
               window.screen.blit(self._screen_overlay, (0, 0))
43
               self._widget_group.draw()
44
45
46
      def process_model_event(self, event):
47
               self._event_to_func_map.get(event.type)(event)
48
49
           except:
              raise KeyError ('Event type not recognized in Paused View (PauseView.
5.0
      process_model_event)', event)
5.1
      def convert_mouse_pos(self, event):
52
          return self._widget_group.process_event(event)
53
{\scriptstyle 1} from data.constants import Colour, Miscellaneous, CursorMode
2 from data.components.widget_group import WidgetGroup
3 from data.states.game.widget_dict import WIN_WIDGETS
4 from data.managers.window import window
5 from data.managers.cursor import cursor
7 class WinView:
      def __init__(self, model):
           self._model = model
10
           self._widget_group = WidgetGroup(WIN_WIDGETS)
11
12
          self._widget_group.handle_resize(window.size)
      def handle_resize(self):
14
15
           self._widget_group.handle_resize(window.size)
16
     def draw(self):
17
           if self._model.states['WINNER'] is not None:
18
              if cursor.get_mode() != CursorMode.ARROW:
19
                   cursor.set_mode(CursorMode.ARROW)
20
21
               if self._model.states['WINNER'] == Colour.BLUE:
22
                   WIN_WIDGETS['red_won'].kill()
23
                   WIN_WIDGETS['draw_won'].kill()
24
               elif self._model.states['WINNER'] == Colour.RED:
25
                   WIN_WIDGETS['blue_won'].kill()
26
                   WIN_WIDGETS['draw_won'].kill()
27
               elif self._model.states['WINNER'] == Miscellaneous.DRAW:
28
                   WIN_WIDGETS['red_won'].kill()
                   WIN_WIDGETS['blue_won'].kill()
30
31
32
               self._widget_group.draw()
33
34
      def set_win_type(self, win_type):
           WIN_WIDGETS['by_draw'].kill()
35
```

```
WIN_WIDGETS['by_timeout'].kill()
           WIN_WIDGETS['by_resignation'].kill()
37
           WIN_WIDGETS['by_checkmate'].kill()
38
           match win_type:
    case 'CAPTURE':
40
41
                   self._widget_group.add(WIN_WIDGETS['by_checkmate'])
42
                case 'DRAW':
43
                   self._widget_group.add(WIN_WIDGETS['by_draw'])
44
45
                case 'RESIGN':
                   self._widget_group.add(WIN_WIDGETS['by_resignation'])
46
47
                case 'TIME':
                   self._widget_group.add(WIN_WIDGETS['by_timeout'])
48
49
50
      def convert_mouse_pos(self, event):
           return self._widget_group.process_event(event)
5.1
1 import pygame
2 import sys
3 from random import randint
4 from data.utils.asset_helpers import get_rotational_angle
{\tt 5} \  \  \, \textbf{from} \  \  \, \textbf{data.states.menu.widget\_dict} \  \  \, \textbf{import} \  \  \, \textbf{MENU\_WIDGETS}
6 from data.constants import MenuEventType, ShaderType
7 from data.utils.asset_helpers import scale_and_cache
8 from data.managers.logs import initialise_logger
9 from data.managers.animation import animation
10 from data.assets import GRAPHICS, MUSIC, SFX
11 from data.managers.window import window
12 from data.managers.audio import audio
13 from data control import _State
15 logger = initialise_logger(__file__)
16
17 class Menu(_State):
     def __init__(self):
18
           super().__init__()
19
20
           self._fire_laser = False
           self._bloom_mask = None
21
           self._laser_mask = None
22
      def cleanup(self):
24
25
           super().cleanup()
26
           {\tt window.clear\_apply\_arguments(ShaderType.BLOOM)}
27
28
           window.clear_apply_arguments(ShaderType.SHAKE)
           window.clear_effect(ShaderType.CHROMATIC_ABBREVIATION)
29
3.0
           return None
31
32
      def startup(self, persist=None):
33
           super().startup(MENU_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
34
           \verb|window.set_apply_arguments| (ShaderType.BASE, background_type=ShaderType.
3.5
       BACKGROUND_BALATRO)
           window.set_effect(ShaderType.CHROMATIC_ABBREVIATION)
36
37
           MENU_WIDGETS['credits'].kill()
39
40
           self._fire_laser = False
           self._bloom_mask = None
41
42
           self._laser_mask = None
43
           self.draw()
44
```

```
self.update_masks()
45
46
47
       @property
       def sphinx_center(self):
           return (window.size[0] - self.sphinx_size[0] / 2, window.size[1] - self.
49
       sphinx_size[1] / 2)
50
       @property
5.1
       def sphinx_size(self):
52
           return (min(window.size) * 0.1, min(window.size) * 0.1)
53
54
55
       @property
       def sphinx_rotation(self):
56
           mouse_pos = (pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1] + 0.01)
57
           return -get_rotational_angle(mouse_pos, self.sphinx_center)
58
59
60
       def get_event(self, event):
61
           if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
                MENU_WIDGETS['credits'].kill()
62
           if event.type == pygame.MOUSEBUTTONDOWN:
64
                self._fire_laser = True
6.5
                audio.play_sfx(SFX['menu_laser_windup'])
66
                audio.play_sfx(SFX['menu_laser_loop'], loop=True)
animation.set_timer(SFX['menu_laser_loop'].get_length() * 1000 / 2,
67
68
       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:
70
                self._fire_laser = False
7.1
                window.clear_effect(ShaderType.RAYS)
7.3
74
                animation.set_timer(300, lambda: window.clear_effect(ShaderType.SHAKE)
       )
                audio.stop_sfx(1000)
7.5
76
           widget_event = self._widget_group.process_event(event)
77
7.8
           if widget_event is None:
80
               return
81
           match widget_event.type:
82
                case None:
83
84
                    return
85
                case MenuEventType.CONFIG_CLICK:
86
                    self.next = 'config
87
                    self.done = True
88
                {\tt case \ MenuEventType.SETTINGS\_CLICK:}
89
90
                    self.next = 'settings'
                    self.done = True
91
                {\tt case \ MenuEventType.BROWSER\_CLICK:}
92
                    self next = 'browser
93
                    self.done = True
94
                case MenuEventType.QUIT_CLICK:
                   pygame.quit()
96
97
                    sys.exit()
                    logger.info('quitting...')
98
                case MenuEventType.CREDITS_CLICK:
99
                    self._widget_group.add(MENU_WIDGETS['credits'])
100
101
       def draw_sphinx(self):
102
```

```
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
107
          window.screen.blit(sphinx_surface, sphinx_rect)
108
109
      def update_masks(self):
          self.draw()
112
          widget_mask = window.screen.copy()
          laser_mask = pygame.mask.from_surface(widget_mask)
114
          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)
          0, 50, 50))
118
          self._bloom_mask = widget_mask
          self._laser_mask = laser_mask
120
122
      def draw(self):
          self._widget_group.draw()
123
124
          self.draw_sphinx()
125
          if self._fire_laser:
126
              window.set_apply_arguments(ShaderType.RAYS, occlusion=self._laser_mask
      , softShadow=0.1)
128
           window.set_apply_arguments(ShaderType.BLOOM, highlight_surface=self.
129
      _bloom_mask, surface_intensity=0.3, brightness_intensity=0.6)
130
      def update(self, **kwargs):
131
          random_offset = lambda: randint(-5, 5) / 40
          if self._fire_laser:
133
              window.clear_effect(ShaderType.RAYS)
134
              136
      window.size[1]),
                  2.2,
137
                  (190, 190, 255),
138
                  0.99.
                  (self.sphinx_rotation - 2 + random_offset(), self.sphinx_rotation
140
      + 2 + random_offset())
141
              ]])
142
143
              window.set_effect(ShaderType.SHAKE)
              \verb|window.set_apply_arguments| (ShaderType.SHAKE, intensity=1)|
144
              pygame.mouse.set_pos(pygame.mouse.get_pos()[0] + random_offset(),
145
      pygame.mouse.get_pos()[1] + random_offset())
146
          super().update(**kwargs)
147
148
      def handle_resize(self):
149
          super().handle_resize()
150
          self.update_masks()
151
 1 from data.components.custom_event import CustomEvent
 2 from data.constants import MenuEventType
```

```
3 from data.managers.theme import theme
4 from data.assets import GRAPHICS
5 from data.widgets import *
7 top_right_container = Rectangle(
       relative_position = (0, 0),
       relative_size = (0.15, 0.075),
      fixed_position=(5, 5),
1.0
       anchor_x='right'
11
       scale_mode='height'
12
13 )
15 MENU_WIDGETS = {
16
       'credits':
17
           relative_position=(0, 0),
1.8
19
           relative_size=(0.7, 0.7),
20
           icon = GRAPHICS ['credits'],
           anchor_x='center',
21
           anchor_y='center',
           margin=50
23
24
       'default': [
           top_right_container,
26
27
           ReactiveIconButton(
28
               parent=top_right_container,
                relative_position = (0, 0),
29
30
                relative_size=(1, 1),
               anchor_x='right',
31
                scale_mode='height',
32
                base_icon = GRAPHICS ['quit_base'],
33
                hover_icon = GRAPHICS['quit_hover'],
34
                press_icon = GRAPHICS['quit_press'],
35
                event = CustomEvent (MenuEventType . QUIT_CLICK)
36
37
           ReactiveIconButton(
38
               parent=top_right_container,
39
                relative_position = (0, 0),
40
                relative_size=(1, 1),
                scale_mode='height',
42
                base_icon = GRAPHICS['credits_base'],
43
                hover_icon = GRAPHICS['credits_hover'],
44
                press_icon=GRAPHICS['credits_press'],
45
46
                event = CustomEvent (MenuEventType . CREDITS_CLICK)
47
48
           ReactiveIconButton(
                relative_position = (0.05, -0.2),
                relative_size=(0, 0.15),
50
                anchor_y='center'
51
52
                base_icon = GRAPHICS['play_text_base'],
                hover_icon=GRAPHICS['play_text_hover'],
press_icon=GRAPHICS['play_text_press'],
53
54
                event = CustomEvent (MenuEventType.CONFIG_CLICK)
55
           ) .
5.6
           Reactive I con Button (
               relative_position = (0.05, 0),
58
59
                relative_size=(0, 0.15),
                anchor_y='center'
60
                base_icon = GRAPHICS['review_text_base'],
6.1
                hover_icon = GRAPHICS['review_text_hover'],
62
                press_icon = GRAPHICS['review_text_press'],
63
                event = CustomEvent (MenuEventType . BROWSER_CLICK)
64
```

```
),
            ReactiveIconButton(
66
                relative_position=(0.05, 0.2),
67
                relative\_size=(0, 0.15),
                anchor_y='center'
69
                base_icon = GRAPHICS['settings_text_base'],
                hover_icon = GRAPHICS['settings_text_hover'],
71
                press_icon=GRAPHICS['settings_text_press'],
                {\tt event=CustomEvent(MenuEventType.SETTINGS\_CLICK)}
            ),
74
            Icon(
75
76
                relative_position = (0.0, 0.1),
7.7
                relative_size=(0.3, 0.2),
                anchor_x='center'
78
                fill_colour=theme['fillSecondary'],
79
                icon = GRAPHICS['title_screen_art'],
8.0
                stretch=False
81
82
            ),
       ]
83
84 }
86 # Widgets used for testing light rays effect
87 TEST_WIDGETS = {
        'default': [
88
            Rectangle (
89
                relative_position=(0.4, 0.2),
90
                relative\_size = (0.1, 0.1),
91
92
                scale_mode='height',
                visible = True,
93
                border_width=0,
94
95
                fill_colour=(255, 0, 0),
                border_radius=1000
96
            ),
97
            Rectangle (
98
                relative_position = (0.5, 0.7),
99
100
                relative_size=(0.1, 0.1),
                scale_mode='height',
101
                visible = True,
102
                border_width =0,
                fill_colour = (255, 0, 0),
104
                border\_radius=1000
105
            ),
106
            Rectangle (
107
                relative_position=(0.6, 0.6),
108
                relative_size=(0.2, 0.2),
                scale_mode='height',
                visible = True,
                border_width=0,
112
                fill_colour=(255, 0, 0),
113
114
                border_radius=1000
115
116
            Rectangle (
                relative_position = (0.4, 0.6),
                relative_size=(0.1, 0.1),
118
                scale_mode='height',
                visible = True,
120
                border_width=0,
121
                fill_colour=(255, 0, 0),
122
                border_radius=1000
123
            ),
124
            Rectangle (
125
                relative_position = (0.6, 0.4),
126
```

```
relative_size=(0.1, 0.1),
127
                 scale_mode='height',
128
                 visible = True,
                 border_width=0,
130
                 fill_colour = (255, 0, 0),
131
                 border_radius=1000
132
            ),
133
            Rectangle (
134
                 relative_position = (0.3, 0.4),
135
                 relative_size = (0.1, 0.1),
136
                 scale_mode='height',
137
138
                 visible = True,
                 border_width =0,
139
                 fill_colour = (255, 0, 0),
140
                 border_radius=1000
141
142
            Rectangle (
143
144
                relative_position = (0.475, 0.15),
                 relative_size=(0.2, 0.2),
145
                 scale_mode='height',
146
                 visible = True,
147
                 border_width=0,
148
                 fill_colour=(255, 0, 0),
                 border_radius=1000
150
            ),
151
            Rectangle (
152
                 relative_position = (0.6, 0.2),
153
154
                 relative_size = (0.1, 0.1),
                 scale_mode='height',
                 visible = True,
156
157
                 border_width =0,
                 fill_colour = (255, 0, 0),
158
                 border_radius=1000
159
            )
160
        ]
161
162 }
 1 import pygame
 2 from collections import deque
 g from data.states.game.components.capture_draw import CaptureDraw
 4 from data.states.game.components.piece_group import PieceGroup
 5 from data.constants import ReviewEventType, Colour, ShaderType
 6 from data.states.game.components.laser_draw import LaserDraw
 {\tt 7~from~data.utils.bitboard\_helpers~import~bitboard\_to\_coords}
 {\tt 8 \ from \ data.states.review.widget\_dict \ import \ REVIEW\_WIDGETS}
 9 from data.utils.browser_helpers import get_winner_string
{\scriptstyle 10} \quad \textbf{from} \quad \textbf{data.states.game.components.board} \quad \textbf{import} \quad \textbf{Board}
 11 from data.components.game_entry import GameEntry
12 from data.managers.logs import initialise_logger
13 from data.managers.window import window
14 from data.control import _State
15 from data.assets import MUSIC
17 logger = initialise_logger(__name__)
19 class Review(_State):
      def __init__(self):
20
21
            super().__init__()
22
23
            self._moves = deque()
24
            self._popped_moves = deque()
            self._game_info = {}
```

```
self._board = None
27
          self._piece_group = None
28
          self._laser_draw = None
          self._capture_draw = None
30
31
      def cleanup(self):
32
33
          Cleanup function. Clears shader effects.
3.4
35
          super().cleanup()
36
37
          window.clear_apply_arguments(ShaderType.BLOOM)
38
          window.clear_effect(ShaderType.RAYS)
3.9
40
          return None
41
42
43
      def startup(self, persist):
44
          Startup function. Initialises all objects, widgets and game data.
45
46
47
          persist (dict): Dict containing game entry data.
49
          super().startup(REVIEW_WIDGETS, MUSIC['review'])
50
51
          \verb|window.set_apply_arguments| (ShaderType.BASE, background_type=ShaderType.
52
      BACKGROUND_WAVES)
          window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
53
      Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
54
          REVIEW_WIDGETS['help'].kill()
5.5
56
          self._moves = deque(GameEntry.parse_moves(persist.pop('moves', '')))
          self._popped_moves = deque()
57
          self._game_info = persist
58
59
          self._board = Board(self._game_info['start_fen_string'])
60
          self._piece_group = PieceGroup()
6.1
          self._laser_draw = LaserDraw(self.board_position, self.board_size)
62
          self._capture_draw = CaptureDraw(self.board_position, self.board_size)
63
64
          self.initialise_widgets()
65
          self.simulate_all_moves()
66
67
          self.refresh_pieces()
          self.refresh_widgets()
68
69
70
          self.draw()
71
72
      @property
73
      def board_position(self):
          return REVIEW_WIDGETS['chessboard'].position
7.4
75
76
      @property
7.7
      def board_size(self):
          return REVIEW_WIDGETS['chessboard'].size
78
79
80
      @property
      def square_size(self):
81
          return self.board_size[0] / 10
82
83
      def initialise_widgets(self):
84
85
```

```
Initializes the widgets for a new game.
86
87
           REVIEW_WIDGETS['move_list'].reset_move_list()
88
           REVIEW_WIDGETS['move_list'].kill()
           REVIEW_WIDGETS['scroll_area'].set_image()
90
9.1
           REVIEW_WIDGETS['winner_text'].set_text(f'WINNER: {get_winner_string(self.
92
       _game_info["winner"])}')
REVIEW_WIDGETS['blue_piece_display'].reset_piece_list()
93
           REVIEW_WIDGETS['red_piece_display'].reset_piece_list()
94
9.5
           if self._game_info['time_enabled']:
96
               REVIEW_WIDGETS['timer_disabled_text'].kill()
97
98
           else:
               REVIEW_WIDGETS['blue_timer'].kill()
99
               REVIEW_WIDGETS['red_timer'].kill()
100
101
       def refresh_widgets(self):
103
           Refreshes the widgets after every move.
104
           REVIEW_WIDGETS['move_number_text'].set_text(f'MOVE NO: {(len(self._moves))
        / 2:.1f} / {(len(self._moves) + len(self._popped_moves)) / 2:.1f}')
           REVIEW_WIDGETS['move_colour_text'].set_text(f'{self.calculate_colour().
       name } TO MOVE')
108
           if self._game_info['time_enabled']:
               if len(self._moves) == 0:
                    REVIEW_WIDGETS['blue_timer'].set_time(float(self._game_info['time'
       ]) * 60 * 1000)
                    REVIEW_WIDGETS['red_timer'].set_time(float(self._game_info['time'
       ]) * 60 * 1000)
               else:
                    REVIEW_WIDGETS['blue_timer'].set_time(float(self._moves[-1]['
114
       blue_time']) * 60 * 1000)
                    REVIEW_WIDGETS['red_timer'].set_time(float(self._moves[-1]['
       red_time']) * 60 * 1000)
           REVIEW_WIDGETS['scroll_area'].set_image()
118
119
       def refresh_pieces(self):
120
           Refreshes the pieces on the board.
           self._piece_group.initialise_pieces(self._board.get_piece_list(), self.
123
       board_position, self.board_size)
       def simulate_all_moves(self):
126
           Simulates all moves at the start of every game to obtain laser results and
        fill up piece display and move list widgets.
128
           for index, move_dict in enumerate(self._moves):
               laser_result = self._board.apply_move(move_dict['move'], fire_laser=
130
       True)
               self._moves[index]['laser_result'] = laser_result
131
                if laser_result.hit_square_bitboard:
                    if laser_result.piece_colour == Colour.BLUE:
134
                        REVIEW_WIDGETS['red_piece_display'].add_piece(laser_result.
       piece_hit)
                    elif laser_result.piece_colour == Colour.RED:
136
```

```
REVIEW_WIDGETS['blue_piece_display'].add_piece(laser_result.
137
       piece_hit)
138
               REVIEW_WIDGETS['move_list'].append_to_move_list(move_dict['
139
       unparsed_move'])
140
       def calculate_colour(self):
141
142
           Calculates the current active colour to move.
143
144
           Returns:
145
           Colour: The current colour to move.
146
147
           if self._game_info['start_fen_string'][-1].lower() == 'b':
148
                initial_colour = Colour.BLUE
149
           elif self._game_info['start_fen_string'][-1].lower() == 'r':
150
               initial_colour = Colour.RED
151
152
           if len(self._moves) % 2 == 0:
153
               return initial_colour
154
           else:
               return initial_colour.get_flipped_colour()
156
157
       def handle_move(self, move, add_piece=True):
158
159
160
           Handles applying or undoing a move.
161
162
           Args:
               move (dict): The move to handle.
163
               add_piece (bool): Whether to add the captured piece to the display.
164
       Defaults to True.
166
           laser_result = move['laser_result']
           active_colour = self.calculate_colour()
167
           self._laser_draw.add_laser(laser_result, laser_colour=active_colour)
168
169
           if laser_result.hit_square_bitboard:
               if laser_result.piece_colour == Colour.BLUE:
                    if add_piece:
172
                        REVIEW_WIDGETS['red_piece_display'].add_piece(laser_result.
       piece_hit)
174
                        REVIEW_WIDGETS['red_piece_display'].remove_piece(laser_result.
       piece_hit)
                elif laser_result.piece_colour == Colour.RED:
                    if add_piece:
                        REVIEW_WIDGETS['blue_piece_display'].add_piece(laser_result.
       piece_hit)
179
                    else:
180
                        REVIEW_WIDGETS['blue_piece_display'].remove_piece(laser_result
       .piece_hit)
181
                self._capture_draw.add_capture(
182
                    laser_result.piece_hit,
183
                    laser_result.piece_colour,
                    laser_result.piece_rotation,
185
                    bitboard_to_coords(laser_result.hit_square_bitboard),
186
                    laser_result.laser_path[0][0],
187
                    active_colour,
188
189
                    shake=False
               )
190
191
```

```
def update_laser_mask(self):
193
           Updates the laser mask for the light rays effect.
194
           temp_surface = pygame.Surface(window.size, pygame.SRCALPHA)
196
197
           self._piece_group.draw(temp_surface)
           mask = pygame.mask.from_surface(temp_surface, threshold=127)
198
           mask_surface = mask.to_surface(unsetcolor=(0, 0, 0, 255), setcolor=(255,
199
       0, 0, 255))
200
            window.set_apply_arguments(ShaderType.RAYS, occlusion=mask_surface)
201
202
       def get_event(self, event):
203
204
            Processes Pygame events.
205
206
207
           Args:
           event (pygame.event.Event): The event to handle.
208
           if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
210
                REVIEW_WIDGETS['help'].kill()
211
           widget_event = self._widget_group.process_event(event)
213
214
           if widget_event is None:
215
216
               return
217
218
           match widget_event.type:
               case None:
219
220
                    return
221
                case ReviewEventType.MENU_CLICK:
223
                    self.next = 'menu'
                    self.done = True
224
                case ReviewEventType.PREVIOUS_CLICK:
226
                    if len(self._moves) == 0:
227
                        return
228
229
                    # Pop last applied move off first stack
230
                    move = self._moves.pop()
231
                    # Pushed onto second stack
232
                    \verb|self._popped_moves.append(move)|\\
233
234
                    # Undo last applied move
                    self._board.undo_move(move['move'], laser_result=move['
236
       laser_result'])
                    self.handle_move(move, add_piece=False)
237
                    REVIEW_WIDGETS['move_list'].pop_from_move_list()
238
240
                    self.refresh_pieces()
241
                    self.refresh_widgets()
                    self.update_laser_mask()
242
243
                case ReviewEventType.NEXT_CLICK:
                   if len(self._popped_moves) == 0:
245
246
                        return
247
                    # Peek at second stack to get last undone move
248
249
                    move = self._popped_moves[-1]
250
                    # Reapply last undone move
251
```

```
self._board.apply_move(move['move'])
252
                     self.handle_move(move, add_piece=True)
                     REVIEW_WIDGETS['move_list'].append_to_move_list(move['
       unparsed_move'])
                     # Pop last undone move from second stack
                     self._popped_moves.pop()
257
                     # Push onto first stack
258
                     self._moves.append(move)
260
                     self.refresh_pieces()
261
262
                     self.refresh_widgets()
                     self.update_laser_mask()
263
264
                 case ReviewEventType.HELP_CLICK:
265
                     {\tt self.\_widget\_group.add(REVIEW\_WIDGETS['help'])}
266
267
                     self._widget_group.handle_resize(window.size)
268
       def handle_resize(self):
269
270
            Handles resizing of the window.
271
            super().handle_resize()
273
            self._piece_group.handle_resize(self.board_position, self.board_size)
274
275
            self._laser_draw.handle_resize(self.board_position, self.board_size)
            self._capture_draw.handle_resize(self.board_position, self.board_size)
276
277
278
            if self._laser_draw.firing:
279
                 self.update_laser_mask()
280
       def draw(self):
282
283
            Draws all components onto the window screen.
284
            self._capture_draw.update()
285
            self._widget_group.draw()
286
            self._piece_group.draw(window.screen)
287
            self._laser_draw.draw(window.screen)
288
            self._capture_draw.draw(window.screen)
 1 from data.widgets import *
 {\tt 2 \ from \ data.components.custom\_event \ import \ CustomEvent}
 3 from data.constants import ReviewEventType, Colour
 {\tt 4} \  \  \, \textbf{from} \  \  \, \textbf{data.assets} \  \  \, \textbf{import} \  \  \, \textbf{GRAPHICS}
 6 MOVE_LIST_WIDTH = 0.2
 8 right_container = Rectangle(
       relative_position = (0.05, 0),
 9
10
       relative_size=(0.2, 0.7),
       anchor_y = 'center',
11
       anchor_x = 'right'
12
13 )
14
15 info_container = Rectangle(
      parent=right_container,
       relative_position = (0, 0.5),
17
18
       relative_size=(1, 0.5),
       visible = True
19
20 )
22 arrow_container = Rectangle(
```

```
relative_position = (0, 0.05),
       relative_size = (0.4, 0.1),
24
       anchor_x='center',
25
       anchor_y = 'bottom'
26
27 )
28
29 move_list = MoveList(
       parent=right_container,
3.0
31
       relative_position = (0, 0),
      relative_width=1,
32
       minimum_height=300,
33
34
       move_list=[]
35 )
36
37 top_right_container = Rectangle(
       relative_position = (0, 0),
3.8
       relative_size = (0.15, 0.075),
39
40
       fixed_position=(5, 5),
       anchor_x = 'right'
41
       scale_mode='height'
43 )
44
45 REVIEW_WIDGETS = {
       'help':
46
47
       Icon(
           relative_position=(0, 0),
48
           relative_size=(1.02, 1.02),
49
50
           icon=GRAPHICS['review_help'],
           anchor_x='center',
51
           anchor_y='center',
5.2
53
           border_width =0,
           fill_colour = (0, 0, 0, 0)
54
55
       'default': [
56
           arrow_container,
57
58
           right_container,
           info_container,
59
           top_right_container,
60
           ReactiveIconButton(
61
                parent=top_right_container,
62
                relative_position = (0, 0),
63
                relative_size=(1, 1),
64
                anchor_x='right',
6.5
66
                scale_mode='height'
                base_icon = GRAPHICS['home_base'],
67
                hover_icon=GRAPHICS['home_hover'],
68
                press_icon = GRAPHICS['home_press'],
69
                event = CustomEvent(ReviewEventType.MENU_CLICK)
70
           ),
71
72
           {\tt ReactiveIconButton}\,(
               parent=top_right_container,
7.3
74
                relative_position = (0, 0),
75
                relative_size=(1, 1),
                scale_mode='height'
7.6
                base_icon = GRAPHICS['help_base'],
77
                hover_icon = GRAPHICS['help_hover'],
78
                press_icon = GRAPHICS['help_press'],
79
                event = CustomEvent (ReviewEventType.HELP_CLICK)
80
           ),
8.1
82
           ReactiveIconButton(
                parent=arrow_container,
83
                relative_position=(0, 0),
84
```

```
relative_size=(1, 1),
                scale_mode='height',
86
                base_icon = GRAPHICS['left_arrow_filled_base'],
87
                hover_icon = GRAPHICS['left_arrow_filled_hover'],
                press_icon = GRAPHICS['left_arrow_filled_press'],
89
                event=CustomEvent(ReviewEventType.PREVIOUS_CLICK)
90
            ),
91
            ReactiveIconButton(
92
93
                parent=arrow_container,
                relative_position = (0, 0),
94
                relative_size=(1, 1),
9.5
96
                scale_mode='height',
                anchor_x='right',
97
                base_icon=GRAPHICS['right_arrow_filled_base'],
98
                hover_icon = GRAPHICS['right_arrow_filled_hover'],
99
                press_icon=GRAPHICS['right_arrow_filled_press'],
                event = CustomEvent(ReviewEventType.NEXT_CLICK)
101
102
            ),
       ],
       'move_list':
104
           move_list,
       'scroll_area':
       ScrollArea(
107
           parent=right_container,
108
            relative_position = (0, 0),
            relative_size=(1, 0.5),
110
            vertical=True,
112
            widget=move_list
113
        'chessboard':
114
       Chessboard(
           relative_position=(0, 0),
116
117
            relative_width=0.4,
            scale_mode='width',
118
            anchor_x='center',
119
            anchor_y='center'
120
       ),
121
        "move_number_text":
       Text(
           parent=info_container,
124
           relative_position = (0, 0),
125
           relative_size=(1, 0.3),
126
            anchor_y='bottom',
127
            text='MOVE NO:',
128
            fit_vertical=False,
129
            margin=10,
130
131
            border_width=0,
            fill_colour=(0, 0, 0, 0),
132
       ),
133
134
       'move_colour_text':
       Text(
135
136
           parent=info_container,
            relative_size=(1, 0.3),
137
           relative_position = (0, 0),
138
            anchor_y='center',
            text='TO MOVE',
140
            fit_vertical=False,
141
            margin=10,
142
            border_width=0,
143
            fill_colour=(0, 0, 0, 0),
144
145
       'winner_text':
146
```

```
Text(
147
             parent=info_container,
148
             relative_size=(1, 0.3),
149
             relative_position=(0, 0),
150
             text='WINNER:',
151
             fit_vertical=False,
152
             margin=10,
153
             border_width=0,
154
             fill_colour=(0, 0, 0, 0),
156
        'blue_timer':
157
158
        Timer(
            relative_position=(0.05, 0.05),
             anchor_y='center',
160
             relative_size = (0.1, 0.1),
161
             active_colour = Colour . BLUE,
162
163
        ),
164
        'red_timer':
        Timer(
165
166
             relative_position = (0.05, -0.05),
             anchor_y='center',
relative_size=(0.1, 0.1),
167
168
             active_colour=Colour.RED
        ),
        'timer_disabled_text':
171
        Text(
172
             relative_size=(0.2, 0.1),
173
             relative_position = (0.05, 0),
174
175
             anchor_y='center',
             fit_vertical=False,
176
177
             text = 'TIMER DISABLED',
178
179
        'blue_piece_display':
        PieceDisplay(
180
            relative_position = (0.05, 0.05),
181
182
             relative_size=(0.2, 0.1),
             anchor_y='bottom',
183
             active_colour=Colour.BLUE
184
        'red_piece_display':
186
        PieceDisplay(
187
             relative_position = (0.05, 0.05),
             relative\_size=(0.2, 0.1),
189
             active_colour=Colour.RED
190
191
192 }
 1 import pygame
 2 from random import randint
 3 from data.utils.data_helpers import get_default_settings, get_user_settings,
        update_user_settings
 {\tt 4 from\ data.constants\ import\ SettingsEventType\ ,\ WidgetState\ ,\ ShaderType\ ,\ SHADER\_MAPSER_{\tt MAPSER}}
 {\tt 5} \  \  \, \textbf{from} \  \  \, \textbf{data.states.settings.widget\_dict} \  \  \, \textbf{import} \  \  \, \textbf{SETTINGS\_WIDGETS}
 6 from data.managers.logs import initialise_logger
 7 from data.managers.window import window
 8 from data.managers.audio import audio
9 from data.widgets import ColourPicker
10 from data.control import _State
11 from data.assets import MUSIC
13 logger = initialise_logger(__name__)
```

```
15 class Settings(_State):
16
      def __init__(self):
           super().__init__()
17
          self._colour_picker = None
19
20
          self._settings = None
21
     def cleanup(self):
22
23
           super().cleanup()
24
          update_user_settings(self._settings)
25
26
27
          return None
28
      def startup(self, persist=None):
29
          super().startup(SETTINGS_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
3.0
31
32
          window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
      BACKGROUND_BALATRO)
          self._settings = get_user_settings()
33
          self.reload_settings()
34
3.5
           self.draw()
36
3.7
      def create_colour_picker(self, mouse_pos, button_type):
38
          if button_type == SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK:
39
               selected_colour = self._settings['primaryBoardColour']
40
41
               event_type = SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK
          else:
42
               selected_colour = self._settings['secondaryBoardColour']
43
               event_type = SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK
45
46
           self._colour_picker = ColourPicker(
              relative_position=(mouse_pos[0] / window.size[0], mouse_pos[1] /
47
      window.size[1]),
48
              relative_width=0.15,
               selected_colour=selected_colour,
49
               event_type=event_type
5.0
          )
51
          self._widget_group.add(self._colour_picker)
52
53
      def remove_colour_picker(self):
54
          self._colour_picker.kill()
5.5
56
      def reload_display_mode(self):
57
          relative_mouse_pos = (pygame.mouse.get_pos()[0] / window.size[0], pygame.
5.8
      mouse.get_pos()[1] / window.size[1])
59
          if self._settings['displayMode'] == 'fullscreen':
60
61
               window.set_fullscreen(desktop=True)
               window.handle_resize()
62
63
          elif self._settings['displayMode'] == 'windowed':
64
              window.set_windowed()
6.5
               window.handle_resize()
              window.restore()
67
68
           self._widget_group.handle_resize(window.size)
69
7.0
          new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
71
      relative_mouse_pos[1] * window.size[1])
          pygame.mouse.set_pos(new_mouse_pos)
72
```

```
73
74
       def reload_shaders(self):
7.5
            window.clear_all_effects()
76
           for shader_type in SHADER_MAP[self._settings['shader']]:
7.7
7.8
                window.set_effect(shader_type)
       def reload_settings(self):
8.0
            SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self.
81
       _settings['primaryBoardColour'])
       SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self._settings['secondaryBoardColour'])
82
           SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.
83
       BASE)
            SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.
84
       BASE)
           SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['
85
       musicVolume'])
           SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume
86
       17)
            SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings
87
       ['displayMode'])
            SETTINGS_WIDGETS['shader_carousel'].set_to_key(self._settings['shader'])
            SETTINGS_WIDGETS['particles_switch'].set_toggle_state(self._settings['
89
       particles'])
            SETTINGS_WIDGETS['opengl_switch'].set_toggle_state(self._settings['opengl'
90
       1)
91
            self.reload_shaders()
92
            self.reload_display_mode()
93
94
       def get_event(self, event):
9.5
96
           widget_event = self._widget_group.process_event(event)
97
            if widget_event is None:
98
                if event.type == pygame.MOUSEBUTTONDOWN and self._colour_picker:
99
                    self.remove_colour_picker()
101
                return
           match widget_event.type:
103
                {\tt case \ SettingsEventType.VOLUME\_SLIDER\_SLIDE:}
104
                    return
106
107
                {\tt case \ SettingsEventType.VOLUME\_SLIDER\_CLICK:}
                    if widget_event.volume_type == 'music':
108
                        audio.set_music_volume(widget_event.volume)
                         self._settings['musicVolume'] = widget_event.volume
                    elif widget_event.volume_type == 'sfx':
                        audio.set_sfx_volume(widget_event.volume)
                        self._settings['sfxVolume'] = widget_event.volume
114
                {\tt case \ SettingsEventType.DROPDOWN\_CLICK:}
                    selected_word = SETTINGS_WIDGETS['display_mode_dropdown'].
       get_selected_word()
                    if selected_word is None or selected_word == self._settings['
118
       displayMode']:
                    self._settings['displayMode'] = selected_word
121
                    self.reload_display_mode()
```

```
124
                case SettingsEventType.MENU_CLICK:
                    self.next = 'menu
                    self.done = True
127
128
                {\tt case \ SettingsEventType.RESET\_DEFAULT:}
                    self._settings = get_default_settings()
130
                    self.reload_settings()
131
                case SettingsEventType.RESET_USER:
                    self._settings = get_user_settings()
135
                    self.reload_settings()
136
                {\tt case SettingsEventType.PRIMARY\_COLOUR\_BUTTON\_CLICK \ | \ SettingsEventType}
137
       . SECONDARY_COLOUR_BUTTON_CLICK:
                    if self._colour_picker:
138
                        self.remove_colour_picker()
140
                    self.create_colour_picker(event.pos, widget_event.type)
141
142
                case SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK | SettingsEventType
143
       . SECONDARY_COLOUR_PICKER_CLICK:
                    if widget_event.colour:
                        r, g, b = widget_event.colour.rgb
145
                        hex_colour = f'0x\{hex(r)[2:].zfill(2)\}\{hex(g)[2:].zfill(2)\}\{
146
       hex(b)[2:].zfill(2)}'
147
                        if widget_event.type == SettingsEventType.
       PRIMARY_COLOUR_PICKER_CLICK:
                            SETTINGS_WIDGETS['primary_colour_button'].
149
       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'].
       \verb|set_state_colour(WidgetState.BASE)|
                            self._settings['secondaryBoardColour'] = hex_colour
156
                {\tt case \ SettingsEventType.SHADER\_PICKER\_CLICK:}
157
158
                    self._settings['shader'] = widget_event.data
                    self.reload_shaders()
160
                case SettingsEventType.OPENGL_CLICK:
161
                    self._settings['opengl'] = widget_event.toggled
                    self.reload_shaders()
163
                case SettingsEventType.PARTICLES_CLICK:
165
                    self._settings['particles'] = widget_event.toggled
166
167
       def draw(self):
168
           self._widget_group.draw()
 1 from data.widgets import *
 2 from data.components.custom_event import CustomEvent
 3 from data constants import SettingsEventType, SHADER_MAP
 4 from data.utils.data_helpers import get_user_settings
 5 from data.assets import GRAPHICS, DEFAULT_FONT
 6 from data.managers.theme import theme
```

```
7 from data.utils.font_helpers import text_width_to_font_size
8 from data.managers.window import window
user_settings = get_user_settings()
11 # font_size = text_width_to_font_size('Shaders (OPENGL GPU REQUIRED)',
      DEFAULT_FONT, 0.4 * window.screen.width)
12 FONT_SIZE = 21
1.3
14 carousel_widgets = {
     key: Text(
15
          relative_position=(0, 0),
16
17
           relative_size=(0.25, 0.04),
          margin=0,
18
          text=key.replace('_', '').upper(),
19
           fit_vertical=True,
20
           border_width=0,
2.1
           fill_colour=(0, 0, 0, 0),
22
23
      ) for key in SHADER_MAP.keys()
24 }
26 reset_container = Rectangle(
      relative_size=(0.2, 0.2),
2.7
      relative_position = (0, 0),
28
      fixed_position=(5, 5),
29
      anchor_x='right'
30
      anchor_y = 'bottom',
31
32 )
33
34 SETTINGS_WIDGETS = {
       'default':
3.5
36
          reset_container,
           ReactiveIconButton(
3.7
38
               relative_position = (0, 0),
               relative_size=(0.075, 0.075),
39
               anchor_x='right',
40
41
               scale_mode='height'
               base_icon = GRAPHICS['home_base'],
42
               hover_icon = GRAPHICS['home_hover'],
43
               press_icon = GRAPHICS['home_press'],
               fixed_position=(5, 5),
45
               event = CustomEvent (SettingsEventType.MENU_CLICK)
46
           ),
47
           Text(
48
               relative_position=(0.01, 0.1),
49
               text='Display mode',
50
               relative_size=(0.4, 0.04),
5.1
               center=False,
52
               border_width=0,
53
54
               margin=0,
55
               font_size=21,
               fill_colour=(0, 0, 0, 0)
56
57
           ),
           Text(
58
               relative_position=(0.01, 0.2),
5.9
               text='Music',
               relative_size = (0.4, 0.04),
61
               center=False,
62
               border_width = 0,
63
64
               margin=0,
65
               font_size = 21,
               fill_colour=(0, 0, 0, 0)
66
           ).
67
```

```
Text(
                relative_position = (0.01, 0.3),
69
                text='SFX',
70
                relative_size=(0.4, 0.04),
71
                center=False,
72
73
                border_width =0,
                margin=0,
74
                font_size = 21,
7.5
                fill_colour=(0, 0, 0, 0)
            ),
77
            Text(
78
                relative_position = (0.01, 0.4),
79
                text='Primary board colour',
80
                relative\_size=(0.4, 0.04),
81
                center=False,
82
                border_width=0,
83
84
                margin=0,
85
                font_size=21,
                fill_colour=(0, 0, 0, 0)
86
87
            ),
            Text(
88
                relative_position=(0.01, 0.5),
89
                text='Secondary board colour',
90
                relative_size=(0.4, 0.04),
91
92
                center=False
                border_width =0,
93
94
                margin=0,
95
                font_size=21,
                fill_colour=(0, 0, 0, 0)
96
            ),
97
98
            Text(
                relative_position=(0.01, 0.6),
99
100
                text='Particles'
                relative_size=(0.4, 0.04),
101
                center=False,
103
                border_width =0,
                margin=0,
104
                font_size=21,
105
                fill_colour=(0, 0, 0, 0)
106
            ),
107
            Text(
108
                relative_position = (0.01, 0.7),
109
                text='Shaders (OPENGL GPU REQUIRED)',
111
                relative_size=(0.4, 0.04),
                center=False,
112
                border_width =0,
113
114
                margin=0,
                font_size=21,
115
                fill_colour=(0, 0, 0, 0)
116
117
            ),
            Text(
118
119
                relative_position = (0.01, 0.8),
                text='Super Secret Settings',
120
                relative_size=(0.4, 0.04),
                center=False,
123
                border_width=0,
124
                margin=0,
                font_size=21,
125
                fill_colour=(0, 0, 0, 0)
126
127
            ),
            TextButton(
128
                parent=reset_container,
129
```

```
relative_position = (0, 0),
130
                 relative_size=(1, 0.5),
131
                 fit_vertical = False,
                 margin=10,
                 text = 'DISCARD CHANGES',
134
                 text_colour=theme['textSecondary'],
135
                 event = CustomEvent (SettingsEventType.RESET_USER)
136
            ),
137
            TextButton(
138
                parent=reset_container,
                 relative_position = (0, 0.5),
140
141
                 relative_size=(1, 0.5),
                fit_vertical=False,
142
                margin=10,
143
                 text='RESET TO DEFAULT',
144
                 text_colour = theme['textSecondary'],
145
                 event = CustomEvent (SettingsEventType.RESET_DEFAULT)
146
147
            )
148
       ],
       'display_mode_dropdown':
149
       Dropdown (
150
            relative_position = (0.4, 0.1),
151
            relative_width=0.2,
            word_list=['fullscreen', 'windowed'],
fill_colour=(255, 100, 100),
153
154
            event=CustomEvent(SettingsEventType.DROPDOWN_CLICK)
155
157
        'primary_colour_button':
       ColourButton(
158
            relative_position = (0.4, 0.4),
160
            relative_size=(0.08, 0.05),
            fill_colour=user_settings['primaryBoardColour'],
161
162
            border_width = 5,
            event=CustomEvent(SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK)
164
       'secondary_colour_button':
165
       ColourButton(
           relative_position = (0.4, 0.5),
167
            relative_size=(0.08, 0.05),
            fill_colour=user_settings['secondaryBoardColour'],
169
170
            border_width =5,
            event=CustomEvent(SettingsEventType.SECONDARY_COLOUR_BUTTON_CLICK)
171
       ),
172
173
        'music_volume_slider':
       VolumeSlider(
174
            relative_position = (0.4, 0.2),
175
            relative_length = (0.5),
176
            default_volume=user_settings['musicVolume'],
            border_width=5,
178
            volume_type='music'
       ),
180
181
       'sfx_volume_slider':
       VolumeSlider(
182
           relative_position = (0.4, 0.3),
183
            relative_length = (0.5),
            default_volume=user_settings['sfxVolume'],
185
            border_width =5,
186
            volume_type='sfx'
187
188
189
        'shader_carousel':
       Carousel (
190
            relative_position = (0.4, 0.8),
191
```

```
margin=5,
           border_width=0,
193
           fill_colour=(0, 0, 0, 0),
194
            widgets_dict=carousel_widgets,
           event = CustomEvent(SettingsEventType.SHADER_PICKER_CLICK),
196
       ),
197
       'particles_switch':
198
       Switch (
199
           relative_position = (0.4, 0.6),
200
           relative_height = 0.04,
201
           event=CustomEvent(SettingsEventType.PARTICLES_CLICK)
202
203
       ),
       'opengl_switch':
204
       Switch(
205
           relative_position = (0.4, 0.7),
206
           relative_height = 0.04,
207
           {\tt event=CustomEvent(SettingsEventType.OPENGL\_CLICK)}
208
209
210 }
 1 import pygame
 2 from PIL import Image
 3 from functools import cache
 4 from random import sample, randint
 5 import math
 7 @cache
 8 def scale_and_cache(image, target_size):
       Caches image when resized repeatedly.
10
11
12
       Args:
           image (pygame.Surface): Image surface to be resized.
1.3
           target_size (tuple[float, float]): New image size.
14
15
16
          pygame.Surface: Resized image surface.
18
       return pygame.transform.scale(image, target_size)
19
21 Ocache
22 def smoothscale_and_cache(image, target_size):
23
       Same as scale\_and\_cache, but with the Pygame smoothscale function.
24
25
26
27
           image (pygame.Surface): Image surface to be resized.
           target_size (tuple[float, float]): New image size.
29
30
       Returns:
       pygame.Surface: Resized image surface.
31
32
33
       return pygame.transform.smoothscale(image, target_size)
34
35 def gif_to_frames(path):
37
       Uses the PIL library to break down GIFs into individual frames.
38
39
       Args:
           path (str): Directory path to GIF file.
40
41
       Yields:
42
```

```
PIL. Image: Single frame.
44
45
          image = Image.open(path)
47
          first_frame = image.copy().convert('RGBA')
48
          yield first_frame
49
          image.seek(1)
5.0
51
          while True:
52
               current_frame = image.copy()
53
               yield current_frame
               image.seek(image.tell() + 1)
55
      except EOFError:
56
          pass
5.8
59 def get_perimeter_sample(image_size, number):
60
      Used for particle drawing class, generates roughly equally distributed points
61
      around a rectangular image surface's perimeter.
62
63
          image_size (tuple[float, float]): Image surface size.
64
          number (int): Number of points to be generated.
6.5
66
67
          list[tuple[int, int], ...]: List of random points on perimeter of image
68
      surface.
69
      perimeter = 2 * (image_size[0] + image_size[1])
71
      # 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
72
      range(0, number)]
      pos_list = []
73
74
      for perimeter_offset in perimeter_offsets:
7.5
          # For every point, add a random offset
76
          max_displacement = int(perimeter / (number * 4))
7.7
          perimeter_offset += randint(-max_displacement, max_displacement)
78
79
          if perimeter_offset > perimeter:
80
              perimeter_offset -= perimeter
81
82
          \# Convert 1D distance back into 2D points on image surface perimeter
8.3
           if perimeter_offset < image_size[0]:</pre>
              pos_list.append((perimeter_offset, 0))
85
           elif perimeter_offset < image_size[0] + image_size[1]:</pre>
86
87
              pos_list.append((image_size[0], perimeter_offset - image_size[0]))
           elif perimeter_offset < image_size[0] + image_size[1] + image_size[0]:</pre>
88
              pos_list.append((perimeter_offset - image_size[0] - image_size[1],
      image_size[1]))
90
          else:
              pos_list.append((0, perimeter - perimeter_offset))
91
      return pos_list
92
93
94 def get_angle_between_vectors(u, v, deg=True):
9.5
96
      Uses the dot product formula to find the angle between two vectors.
97
      Args:
98
```

```
u (list[int, int]): Vector 1.
99
            v (list[int, int]): Vector 2.
           deg (bool, optional): Return results in degrees. Defaults to True.
101
       Returns:
103
       float: Angle between vectors.
104
105
       dot_product = sum(i * j for (i, j) in zip(u, v))
u_magnitude = math.sqrt(u[0] ** 2 + u[1] ** 2)
106
107
       v_magnitude = math.sqrt(v[0] ** 2 + v[1] ** 2)
108
109
       cos_angle = dot_product / (u_magnitude * v_magnitude)
       radians = math.acos(min(max(cos_angle, -1), 1))
112
       if deg:
113
           return math.degrees(radians)
114
       else:
115
116
           return radians
118 def get_rotational_angle(u, v, deg=True):
119
       Get bearing angle relative to positive x-axis centered on second vector.
120
121
       Args:
           u (list[int, int]): Vector 1.
123
            v (list[int, int]): Vector 2, set as center of axes.
124
           deg (bool, optional): Return results in degrees. Defaults to True.
125
126
127
          float: Bearing angle between vectors.
128
       radians = math.atan2(u[1] - v[1], u[0] - v[0])
130
131
132
       if deg:
           return math.degrees(radians)
133
       else:
134
           return radians
135
136
137 def get_vector(src_vertex, dest_vertex):
138
       Get vector describing translation between two points.
139
140
141
       Args:
           src_vertex (list[int, int]): Source vertex.
142
           dest_vertex (list[int, int]): Destination vertex.
143
144
145
       tuple[int, int]: Vector between the two points.
146
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
       Used in particle drawing system, finds coordinates of the next corner going
       clockwise, given a point on the perimeter.
153
154
       Args:
           vertex (list[int, int]): Point on perimeter.
           image_size (list[int, int]): Image size.
156
157
158
           list[int, int]: Coordinates of corner on perimeter.
159
```

```
160
       corners = [(0, 0), (image_size[0], 0), (image_size[0], image_size[1]), (0,
161
       image_size[1])]
       if vertex in corners:
163
           return corners[(corners.index(vertex) + 1) % len(corners)]
164
165
       if vertex[1] == 0:
166
           return (image_size[0], 0)
167
       elif vertex[0] == image_size[0]:
168
169
           return image_size
       elif vertex[1] == image_size[1]:
           return (0, image_size[1])
       elif vertex[0] == 0:
172
           return (0, 0)
173
174
175 def pil_image_to_surface(pil_image):
176
       Args:
           pil_image (PIL.Image): Image to be converted.
178
179
180
       Returns:
          pygame.Surface: Converted image surface.
181
182
       return pygame.image.frombytes(pil_image.tobytes(), pil_image.size, pil_image.
183
       mode).convert()
184
185 def calculate_frame_index(elapsed_milliseconds, start_index, end_index, fps):
186
       Determine frame of animated GIF to be displayed.
187
188
189
190
           elapsed_milliseconds (int): Milliseconds since GIF started playing.
           start_index (int): Start frame of GIF.
191
           end_index (int): End frame of GIF.
192
           fps (int): Number of frames to be played per second.
193
194
       Returns:
195
          int: Displayed frame index of GIF.
196
197
       ms_per_frame = int(1000 / fps)
198
       return start_index + ((elapsed_milliseconds // ms_per_frame) % (end_index -
199
       start_index))
200
201 def draw_background(screen, background, current_time=0):
202
       Draws background to screen
203
204
205
       Args:
206
           screen (pygame.Surface): Screen to be drawn to
           background (list[pygame.Surface, ...] | pygame.Surface): Background to be
207
       \hbox{drawn, if $\operatorname{GIF}$, list of surfaces indexed to select frame to be } \hbox{drawn}
           current_time (int, optional): Used to calculate frame index for GIF.
208
       Defaults to 0.
       if isinstance(background, list):
           # Animated background passed in as list of surfaces, calculate_frame_index
211
       () used to get index of frame to be drawn
           frame_index = calculate_frame_index(current_time, 0, len(background), fps
       =8)
           scaled_background = scale_and_cache(background[frame_index], screen.size)
           screen.blit(scaled_background, (0, 0))
214
```

```
215
       else:
           scaled_background = scale_and_cache(background, screen.size)
216
           screen.blit(scaled_background, (0, 0))
217
218
219 def get_highlighted_icon(icon):
220
       Used for pressable icons, draws overlay on icon to show as pressed.
221
223
           icon (pygame.Surface): Icon surface.
224
225
226
       Returns:
       pygame.Surface: Icon with overlay drawn on top.
227
228
       icon_copy = icon.copy()
229
       overlay = pygame.Surface((icon.get_width(), icon.get_height()), pygame.
230
       SRCALPHA)
231
       overlay.fill((0, 0, 0, 128))
       icon_copy.blit(overlay, (0, 0))
232
      return icon_copy
233
 1 from data.constants import Rank, File, EMPTY_BB
 2 from data.managers.logs import initialise_logger
 4 logger = initialise_logger(__name__)
 6 def print_bitboard(bitboard):
       if (bitboard >= (2 ** 80)):
           raise ValueError('Invalid bitboard: too many bits')
 9
       characters = ''
 10
 11
       for rank in reversed(Rank):
           for file in File:
               mask = 1 << (rank * 10 + file)
14
                if (bitboard & mask) != 0:
1.5
                    characters += '1
                else:
17
                    characters += '. '
18
           characters += '\n\n'
20
21
       logger.info('\n' + characters + '\n')
22
23
24 def is_occupied(bitboard, target_bitboard):
      return (target_bitboard & bitboard) != EMPTY_BB
25
26
27 def clear_square(bitboard, target_bitboard):
       return ("target_bitboard & bitboard)
28
29
30 def set_square(bitboard, target_bitboard):
      return (target_bitboard | bitboard)
3.1
33 def index_to_bitboard(index):
       return (1 << index)
3.4
36 def coords_to_bitboard(coords):
37
       index = coords[1] * 10 + coords[0]
       return index_to_bitboard(index)
3.9
40 def bitboard_to_notation(bitboard):
       index = bitboard_to_index(bitboard)
```

```
x = index // 10
42
      y = index % 10
43
44
      return chr(y + 97) + str(x + 1)
46
47 def notation_to_bitboard(notation):
      index = (int(notation[1]) - 1) * 10 + int(ord(notation[0])) - 97
49
       return index_to_bitboard(index)
50
51
52 def bitboard_to_index(bitboard):
       return bitboard.bit_length() - 1
54
55 def bitboard_to_coords(bitboard):
       list_position = bitboard_to_index(bitboard)
56
       x = list_position % 10
5.7
       y = list_position // 10
58
59
6.0
       return x, y
61
62 def bitboard_to_coords_list(bitboard):
       list_positions = []
63
64
      for square in occupied_squares(bitboard):
65
66
           list_positions.append(bitboard_to_coords(square))
67
68
      return list_positions
69
70 def occupied_squares(bitboard):
      while bitboard:
71
72
           lsb_square = bitboard & -bitboard
           bitboard = bitboard ^ lsb_square
7.3
74
           yield lsb_square
75
76
77 def pop_count(bitboard):
      count = 0
78
       while bitboard:
7.9
          count += 1
80
           lsb_square = bitboard & -bitboard
81
           bitboard = bitboard ^ lsb_square
82
      return count
84
86 # def pop_count(bitboard):
87 #
        count = 0
88 #
         while bitboard:
89 #
            count += 1
             bitboard &= bitboard - 1
90 #
92 #
        return count
93
94 def loop_all_squares():
    for i in range(80):
9.5
          yield 1 << i
97
98 #Solar
99 def get_LSB_value(bitboard: int):
      return bitboard & -bitboard
100
101
102 def pop_count_2(bitboard):
       count = 0
103
```

```
while bitboard > 0:
104
105
           lsb_value = get_LSB_value(bitboard)
           count += 1
106
           bitboard ^= lsb_value
108
109
      return count
 1 import pygame
 2 from data.utils.data_helpers import get_user_settings
 3 from data.assets import DEFAULT_FONT
 5 user_settings = get_user_settings()
 {\tiny 7~def~create\_board(board\_size,~primary\_colour,~secondary\_colour,~font=DEFAULT\_FONT):}
       square_size = board_size[0] / 10
       board_surface = pygame.Surface(board_size)
 9
1.0
      for i in range(80):
           x = i \% 10
12
           y = i // 10
13
           if (x + y) \% 2 == 0:
15
16
               square_colour = primary_colour
           else:
17
               square_colour = secondary_colour
18
19
           square_x = x * square_size
20
21
           square_y = y * square_size
22
           pygame.draw.rect(board_surface, square_colour, (square_x, square_y,
23
       square_size + 1, square_size + 1)) # +1 to fill in black lines
           if y == 7:
2.5
               text_position = (square_x + square_size * 0.7, square_y + square_size
26
       * 0.55)
               text_size = square_size / 3
27
               font.render_to(board_surface, text_position, str(chr(x + 1 + 96)),
       fgcolor=(10, 10, 10, 175), size=text_size)
           if x == 0:
29
               text_position = (square_x + square_size * 0.1, square_y + square_size
30
       * 0.1)
               text_size = square_size / 3
               font.render_to(board_surface, text_position, str(7-y + 1), fgcolor
32
       =(10, 10, 10, 175), size=text_size)
33
       return board_surface
34
3.5
36 def create_square_overlay(square_size, colour):
       overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
37
       overlay.fill(colour)
38
39
40
       return overlay
41
42 def create_circle_overlay(square_size, colour):
43
       overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
       pygame.draw.circle(overlay, colour, (square_size / 2, square_size / 2),
       square_size / 4)
45
46
       return overlay
47
48 def coords_to_screen_pos(coords, board_position, square_size):
       x = board_position[0] + (coords[0] * square_size)
```

```
y = board_position[1] + ((7 - coords[1]) * square_size)
50
5.1
      return (x, y)
52
54 def screen_pos_to_coords(mouse_position, board_position, board_size):
      if (board_position[0] <= mouse_position[0] <= board_position[0] + board_size</pre>
5.5
      [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)
56
          y = (board_size[1] - (mouse_position[1] - board_position[1])) // (
57
      board_size[0] / 10)
          return (int(x), int(y))
59
      return None
60
1 from data.constants import Miscellaneous, Colour
3 def get_winner_string(winner):
      if winner is None:
          return 'UNFINISHED'
      elif winner == Miscellaneous.DRAW:
          return 'DRAW'
      else:
          return Colour(winner).name
1 import sqlite3
2 from pathlib import Path
3 from datetime import datetime
5 database_path = (Path(__file__).parent / '../database/database.db').resolve()
7 def insert_into_games(game_entry):
      Inserts a new row into games table.
11
      game_entry (GameEntry): GameEntry object containing game information.
13
      connection = sqlite3.connect(database_path, detect_types=sqlite3.
14
      PARSE_DECLTYPES)
      connection.row_factory = sqlite3.Row
15
16
      cursor = connection.cursor()
17
      # Datetime added for created_dt column
18
19
      game_entry = (*game_entry, datetime.now())
20
      {\tt cursor.execute(\ ^{I\ I\ I}}
21
          INSERT INTO games (cpu_enabled, cpu_depth, winner, time_enabled, time,
      number_of_ply, moves, start_fen_string, final_fen_string, created_dt)
          VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
23
      ''', game_entry)
24
25
      connection.commit()
26
27
      # Return inserted row
28
      cursor.execute('''
          SELECT * FROM games WHERE id = LAST_INSERT_ROWID()
30
31
32
      inserted_row = cursor.fetchone()
33
34
      connection.close()
35
```

```
return dict(inserted_row)
36
37
38 def get_all_games():
      Get all rows in games table.
40
41
42
      list[dict]: List of game entries represented as dictionaries.
43
44
      connection = sqlite3.connect(database_path, detect_types=sqlite3.
45
      PARSE_DECLTYPES)
      connection.row_factory = sqlite3.Row
46
      cursor = connection.cursor()
47
48
     SELECT * FROM games
      cursor.execute('''
49
5.0
51
52
      games = cursor.fetchall()
53
      connection.close()
55
      return [dict(game) for game in games]
56
57
58 def delete_all_games():
59
      Delete all rows in games table.
60
61
62
      connection = sqlite3.connect(database_path)
      cursor = connection.cursor()
63
64
65
     DELETE FROM games
      cursor.execute('''
66
67
68
      connection.commit()
6.9
70
      connection.close()
71
72 def delete_game(id):
73
      Deletes specific row in games table using id attribute.
74
75
76
      Args:
      id (int): Primary key for row.
7.7
78
      connection = sqlite3.connect(database_path)
79
      cursor = connection.cursor()
80
81
      cursor.execute('''
82
         DELETE FROM games WHERE id = ?
83
      ''', (id,))
84
8.5
86
      connection.commit()
87
      connection.close()
88
89 def get_ordered_games(column, ascend=True, start_row=1, end_row=10):
90
      Get specific number of rows from games table ordered by a specific column(s).
91
92
93
      Args:
94
          column (_type_): Column to sort by.
          ascend (bool, optional): Sort ascending or descending. Defaults to True.
95
          start_row (int, optional): First row returned. Defaults to 1.
96
```

```
end_row (int, optional): Last row returned. Defaults to 10.
97
98
       Raises:
99
           ValueError: If ascend argument or column argument are invalid types.
100
101
102
       Returns:
       list[dict]: List of ordered game entries represented as dictionaries.
103
104
       if not isinstance(ascend, bool) or not isinstance(column, str):
105
           raise ValueError('(database_helpers.get_ordered_games) Invalid input
106
       arguments!')
107
       connection = sqlite3.connect(database_path, detect_types=sqlite3.
108
       PARSE_DECLTYPES)
       connection.row_factory = sqlite3.Row
       cursor = connection.cursor()
111
112
       # Match ascend bool to correct SQL keyword
       if ascend:
113
           ascend_arg = 'ASC'
114
       else:
115
            ascend_arg = 'DESC'
116
117
       # Partition by winner, then order by time and number_of_ply
if column == 'winner':
118
119
            cursor.execute(f'''
120
                SELECT * FROM
                    (SELECT ROW_NUMBER() OVER (
122
                         PARTITION BY winner
123
                         ORDER BY time {ascend_arg}, number_of_ply {ascend_arg}
124
                    ) AS row_num, * FROM games)
                WHERE row_num >= ? AND row_num <= ?
126
            ''', (start_row, end_row))
127
128
       else:
       # Order by time or number_of_ply only
129
            cursor.execute(f'''
130
                SELECT * FROM
131
                    (SELECT ROW_NUMBER() OVER (
132
                         ORDER BY {column} {ascend_arg}
133
                ) AS row_num, * FROM games)
WHERE row_num >= ? AND row_num <= ?
134
135
            ''', (start_row, end_row))
136
137
138
       games = cursor.fetchall()
139
       connection.close()
140
141
       return [dict(game) for game in games]
142
143
144 def get_number_of_games():
145
146
       Returns:
       int: Number of rows in the games.
147
148
       connection = sqlite3.connect(database_path)
       cursor = connection.cursor()
150
151
       cursor.execute("""
           SELECT COUNT(ROWID) FROM games
153
154
       result = cursor.fetchall()[0][0]
156
```

```
157
        connection.close()
159
160
        return result
161
162 # delete_all_games()
 1 import json
 2 from pathlib import Path
 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()
 9 def load_json(path):
10
        Args:
11
            path (str): Path to JSON file.
12
13
14
       Raises:
            Exception: Invalid file.
15
16
      Returns:
17
      dict: Parsed JSON file.
18
19
20
            with open(path, 'r') as f:
21
22
                 file = json.load(f)
23
            return file
24
25
        except:
            raise Exception('Invalid JSON file (data_helpers.py)')
26
27
28 def get_user_settings():
29
        return load_json(user_file_path)
30
31 def get_default_settings():
        return load_json(default_file_path)
32
33
34 def get_themes():
        return load_json(themes_file_path)
35
36
37 def update_user_settings(data):
3.8
39
        Rewrites {\tt JSON} file for user settings with new data.
40
41
            data (dict): Dictionary storing updated user settings.
42
43
44
        Raises:
        Exception: Invalid file.
45
46
 47
            with open(user_file_path, 'w') as f:
48
                 json.dump(data, f, indent=4)
49
        except:
50
            raise Exception('Invalid JSON file (data_helpers.py)')
51
 1 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
10
12 def width_to_font_size(font, target_width):
13
      test_size = 1
      while True:
14
           glyph_metrics = font.get_metrics(' ', size=test_size)
15
16
           if (glyph_metrics[0][4] * 8) > target_width:
               return (test_size - 1)
19
           test size += 1
20
21
22 def text_width_to_font_size(text, font, target_width):
     test_size = 1
      if len(text) == 0:
24
           # print('(text_width_to_font_size) Text must have length greater than 1!')
2.5
          text = " "
26
27
     while True:
28
          text_rect = font.get_rect(text, size=test_size)
29
3.0
31
           if text_rect.width > target_width:
               return (test_size - 1)
32
3.3
           test_size += 1
3.5
36 def text_height_to_font_size(text, font, target_height):
       test_size = 1
38
      if ('(' in text) or (')' in text):
39
           \texttt{text} = \texttt{text.replace('(', 'j')} \ \# \ \texttt{Pygame} \ \texttt{freetype} \ \texttt{thinks} \ \texttt{'('} \ \texttt{or} \ \texttt{')'} \ \texttt{is}
40
       taller for some reason
          text = text.replace(')', 'j')
42
     if len(text) == 0:
43
           # print('(text_height_to_font_size) Text must have length greater than
44
      1!')
           text = "j"
45
46
      while True:
47
           text_rect = font.get_rect(text, size=test_size)
49
           if text_rect.height > target_height:
50
51
               return (test_size - 1)
52
           test_size += 1
53
54
55 def get_font_height(font, font_size):
       glyph_metrics = font.get_metrics('j', size=font_size)
       descender = font.get_sized_descender(font_size)
57
      return abs(glyph_metrics[0][3] - glyph_metrics[0][2]) - descender
5.8
1 from data.constants import MoveType, Rotation
3 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:
6
          raise ValueError('Invalid move - type - move type must be m or r!')
      return MoveType(move_type.lower())
9
10
11 def parse_notation(notation):
      if (notation[0].isalpha() is False) or (notation[1].isnumeric() is False):
12
13
          raise ValueError('Invalid notation - invalid notation input types!')
      if not (97 <= ord(notation[0]) <= 106):</pre>
14
          raise ValueError('Invalid notation - file is out of range!')
1.5
16
      elif not (0 <= int(notation[1]) <= 10):</pre>
         raise ValueError('Invalid notation - rank is out of range!')
17
1.8
      return notation
19
2.0
21 def parse_rotation(rotation):
22
      if rotation == '':
          return None
23
     if rotation.isalpha() is False:
         raise ValueError('Invalid rotation - rotation must be a string!')
25
     if rotation.lower() not in Rotation:
26
          raise ValueError('Invalid rotation - rotation is invalid!')
28
29
     return Rotation (rotation.lower())
1 import pygame
2 from pathlib import Path
4 import pygame.freetype
5 from data.utils.asset_helpers import gif_to_frames, pil_image_to_surface
7 def convert_gfx_alpha(image, colorkey=(0, 0, 0)):
      # if image.get_alpha():
          return image.convert_alpha()
9
     # else:
10
11
      #
            image = image.convert_alpha()
            image.set_colorkey(colorkey)
12
13
            return image
14
15
16 def load_gfx(path, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
      file_path = Path(path)
17
      name, extension = file_path.stem, file_path.suffix
18
19
      if extension.lower() in accept:
20
21
          if extension.lower() == '.gif':
               frames_list = []
23
               for frame in gif_to_frames(path):
24
                   image_surface = pil_image_to_surface(frame)
25
26
                   frames_list.append(image_surface)
27
              return frames_list
28
29
          if extension.lower() == '.svg':
              low_quality_image = pygame.image.load_sized_svg(path, (200, 200))
31
               image = pygame.image.load(path)
32
              image = convert_gfx_alpha(image, colorkey)
33
34
35
              return [image, low_quality_image]
36
```

```
37
           else:
               image = pygame.image.load(path)
38
               return convert_gfx_alpha(image, colorkey)
3.9
41 def load_all_gfx(directory, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".
      gif")):
      graphics = {}
42
43
      for file in Path(directory).rglob('*'):
44
           name, extension = file.stem, file.suffix
45
           path = Path(directory / file)
46
47
           if extension.lower() in accept and 'old' not in name:
48
               if name == 'piece_spritesheet':
49
                   data = load_spritesheet(
50
                       path,
5.1
                        (16, 16),
52
53
                        ['pyramid_1', 'scarab_1', 'anubis_1', 'pharoah_1', 'sphinx_1',
       'pyramid_0', 'scarab_0', 'anubis_0', 'pharoah_0', 'sphinx_0'],

['_a', '_b', '_c', '_d'])
55
                   graphics = graphics | data
56
                   continue
57
58
               data = load_gfx(path, colorkey, accept)
59
60
61
               if isinstance(data, list):
                   graphics[name] = data[0]
62
                   graphics[f'{name}_lq'] = data[1]
63
               else:
64
65
                   graphics[name] = data
66
67
      return graphics
68
69 def load_spritesheet(path, sprite_size, col_names, row_names):
       spritesheet = load_gfx(path)
70
      col_count = int(spritesheet.width / sprite_size[0])
71
      row_count = int(spritesheet.height / sprite_size[1])
73
      sprite_dict = {}
74
7.5
      for column in range(col_count):
76
           for row in range(row_count):
7.7
78
               surface = pygame.Surface(sprite_size, pygame.SRCALPHA)
               name = col_names[column] + row_names[row]
79
80
               surface.blit(spritesheet, (0, 0), (column * sprite_size[0], row *
81
      sprite_size[1], *sprite_size))
               sprite_dict[name] = surface
82
83
84
      return sprite_dict
85
86 def load_all_fonts(directory, accept=(".ttf", ".otf")):
      fonts = \{\}
87
      for file in Path(directory).rglob('*'):
89
           name, extension = file.stem, file.suffix
90
           path = Path(directory / file)
91
92
93
           if extension.lower() in accept:
               font = pygame.freetype.Font(path)
94
               fonts[name] = font
95
```

```
96
       return fonts
97
98
99 def load_all_sfx(directory, accept=(".mp3", ".wav", ".ogg")):
       sound_effects = {}
100
101
       for file in Path(directory).rglob('*'):
102
           name, extension = file.stem, file.suffix
path = Path(directory / file)
103
104
105
           if extension.lower() in accept and 'old' not in name:
106
107
                sound_effects[name] = load_sfx(path)
108
       return sound effects
109
111 def load_sfx(path, accept=(".mp3", ".wav", ".ogg")):
112
       file_path = Path(path)
113
       name, extension = file_path.stem, file_path.suffix
114
       if extension.lower() in accept:
115
           sfx = pygame.mixer.Sound(path)
116
           return sfx
118
119 def load_all_music(directory, accept=(".mp3", ".wav", ".ogg")):
120
       music_paths = {}
       for file in Path(directory).rglob('*'):
121
           name, extension = file.stem, file.suffix
123
           path = Path(directory / file)
124
           if extension.lower() in accept:
125
               music_paths[name] = path
127
128
      return music_paths
 1 import pygame
 2 from math import sqrt
 4 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.
1.0
11
           border_width (float): Border width.
           border_colour (pygame.Color): Border colour.
12
13
       Returns:
14
       pygame.Surface: Slider image surface.
15
16
17
       gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
       border_rect = pygame.FRect((0, 0, gradient_surface.width, gradient_surface.
18
       height))
19
       # Draws rectangle with a border radius half of image height, to draw an
20
       rectangle with semicurclar cap (obround)
       pygame.draw.rect(gradient_surface, fill_colour, border_rect, border_radius=int
21
       (size[1] / 2))
22
       pygame.draw.rect(gradient_surface, border_colour, border_rect, width=int(
       border_width), border_radius=int(size[1] / 2))
23
       return gradient_surface
24
```

```
26 def create_slider_gradient(size, border_width, border_colour):
27
      Draws surface for colour slider, with a full colour gradient as fill colour.
28
29
3.0
      Args:
           size (list[int, int]): Image size.
31
           border_width (float): Border width.
32
           border_colour (pygame.Color): Border colour.
33
34
3.5
      Returns:
      pygame.Surface: Slider image surface.
36
37
       gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
3.8
39
      first_round_end = gradient_surface.height / 2
40
      second_round_end = gradient_surface.width - first_round_end
41
42
      gradient_y_mid = gradient_surface.height / 2
43
      # Iterate through length of slider
44
      for i in range(gradient_surface.width):
45
           draw_height = gradient_surface.height
46
           if i < first_round_end or i > second_round_end:
    # Draw semicircular caps if x-distance less than or greater than
48
49
      radius of cap (half of image height)
               distance_from_cutoff = min(abs(first_round_end - i), abs(i -
5.0
       second_round_end))
              draw_height = calculate_gradient_slice_height(distance_from_cutoff,
51
      gradient_surface.height / 2)
52
           # Get colour from distance from left side of slider
53
54
           color = pygame.Color(0)
           color.hsva = (int(360 * i / gradient_surface.width), 100, 100, 100)
55
56
           draw_rect = pygame.FRect((0, 0, 1, draw_height - 2 * border_width))
57
           draw_rect.center = (i, gradient_y_mid)
58
5.9
           pygame.draw.rect(gradient_surface, color, draw_rect)
60
61
      border_rect = pygame.FRect((0, 0, gradient_surface.width, gradient_surface.
62
      height))
      pygame.draw.rect(gradient_surface, border_colour, border_rect , width=int(
63
      border_width), border_radius=int(size[1] / 2))
64
6.5
      return gradient surface
67 def calculate_gradient_slice_height(distance, radius):
68
69
      Calculate height of vertical slice of semicircular slider cap.
7.0
71
      Args:
           distance (float): x-distance from center of circle.
72
           radius (float): Radius of semicircle.
7.3
74
      Returns:
75
      float: Height of vertical slice.
76
      return sqrt(radius ** 2 - distance ** 2) * 2 + 2
7.8
79
80 def create_slider_thumb(radius, colour, border_colour, border_width):
```

```
Creates surface with bordered circle.
82
83
84
       Args:
           radius (float): Radius of circle.
           colour (pygame.Color): Fill colour.
86
           border_colour (pygame.Color): Border colour.
87
           border_width (float): Border width.
88
89
90
       Returns:
          pygame.Surface: Circle surface.
91
92
93
       thumb_surface = pygame.Surface((radius * 2, radius * 2), pygame.SRCALPHA)
       pygame.draw.circle(thumb_surface, border_colour, (radius, radius), radius,
94
       width=int(border_width))
       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
       Creates a square gradient for the colour picker widget, gradient transitioning
101
        between saturation and value.
       Uses smoothscale to blend between colour values for individual pixels.
102
103
104
       Args:
           side_length (float): Length of a square side.
105
106
           colour (pygame.Color): Colour with desired hue value.
       Returns:
108
       pygame.Surface: Square gradient surface.
111
       square_surface = pygame.Surface((side_length, side_length))
112
       mix_1 = pygame.Surface((1, 2))
113
       mix_1.fill((255, 255, 255))
114
       mix_1.set_at((0, 1), (0, 0, 0))
115
       mix_1 = pygame.transform.smoothscale(mix_1, (side_length, side_length))
116
117
       hue = colour.hsva[0]
118
       saturated_rgb = pygame.Color(0)
119
       saturated_rgb.hsva = (hue, 100, 100)
120
       mix_2 = pygame.Surface((2, 1))
       mix_2.fill((255, 255, 255))
123
       \label{linear_set_at} \mbox{mix\_2.set\_at((1, 0), saturated\_rgb)}
124
       mix_2 = pygame.transform.smoothscale(mix_2,(side_length, side_length))
125
126
       mix_1.blit(mix_2, (0, 0), special_flags=pygame.BLEND_MULT)
127
128
       square_surface.blit(mix_1, (0, 0))
129
130
       return square_surface
131
132
133 def create_switch(size, colour):
134
       Creates surface for switch toggle widget.
135
136
137
       Args:
           size (list[int, int]): Image size.
138
           colour (pygame.Color): Fill colour.
139
140
```

```
141
       Returns:
       pygame.Surface: Switch surface.
142
143
       switch_surface = pygame.Surface((size[0], size[1]), pygame.SRCALPHA)
       pygame.draw.rect(switch\_surface, colour, (0, 0, size[0], size[1]),\\
145
       border_radius=int(size[1] / 2))
146
       return switch surface
147
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:
           size (list[int, int]): Image size.
154
           border_width (float): Border width.
155
           colours (list[pygame.Color, ...]): List of 4 colours, representing border
156
       colour, shadow colour, flat colour and highlighted colour.
157
       Returns:
158
       pygame.Surface: Textbox surface.
160
       surface = pygame.Surface(size, pygame.SRCALPHA)
161
162
       pygame.draw.rect(surface, colours[0], (0, 0, *size))
163
       pygame.draw.rect(surface, colours[2], (border_width, border_width, size[0] - 2
164
        * border_width, size[1] - 2 * border_width))
       pygame.draw.rect(surface, colours[3], (border_width, border_width, size[0] - 2
        * border_width, border_width))
       {\tt pygame.draw.rect(surface, colours[1], (border\_width, size[1] - 2 *}
166
       border_width, size[0] - 2 * border_width, border_width))
167
168
       return surface
 1 import pygame
 2 from data.widgets.bases.widget import _Widget
 3 from data.widgets.chessboard import Chessboard
 4 from data.states.game.components.piece_group import PieceGroup
 5 from data.states.game.components.bitboard_collection import BitboardCollection
 7 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(
11
 12
               parent=self._parent,
                relative_position = (0, 0),
               scale_mode=kwargs.get('scale_mode'),
14
                relative_width = relative_width
15
16
           self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
19
20
           self.initialise_board(fen_string)
           self.set_image()
           self.set_geometry()
22
23
24
      def initialise_board(self, fen_string):
           if len(fen_string) == 0:
2.5
               piece_list = []
26
           else:
27
```

```
piece_list = BitboardCollection(fen_string).convert_to_piece_list()
          self._piece_group = PieceGroup()
3.0
          self._piece_group.initialise_pieces(piece_list, (0, 0), self.size)
32
3.3
           self._board.refresh_board()
          self.set_image()
34
3.5
36
      def set_image(self):
          self.image = pygame.transform.scale(self._empty_surface, self.size)
37
38
39
           self._board.set_image()
          self.image.blit(self._board.image, (0, 0))
40
41
           self._piece_group.draw(self.image)
42
43
      def set_geometry(self):
44
45
          super().set_geometry()
          self._board.set_geometry()
46
47
      def set_surface_size(self, new_surface_size):
48
           super().set_surface_size(new_surface_size)
49
           self._board.set_surface_size(new_surface_size)
50
          {\tt self.\_piece\_group.handle\_resize((0, 0), self.size)}
5.1
52
53
      def process_event(self, event):
5.4
          pass
1 import pygame
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.board_thumbnail import BoardThumbnail
4 from data.constants import WidgetState
5 from data.components.custom_event import CustomEvent
7 class BoardThumbnailButton(_Pressable, BoardThumbnail):
      def __init__(self, event, **kwargs):
9
           _Pressable.__init__(
               self,
               event = CustomEvent(**vars(event), fen_string = kwargs.get('fen_string')),
11
               \verb|hover_func=lambda|: self.set_state_colour(WidgetState.HOVER)|,
               down_func=lambda: self.set_state_colour(WidgetState.PRESS),
13
               up_func=lambda: self.set_state_colour(WidgetState.BASE),
14
          )
          BoardThumbnail.__init__(self, **kwargs)
16
           self.initialise_new_colours(self._fill_colour)
          self.set_state_colour(WidgetState.BASE)
1 import pygame
2 from data.utils.font_helpers import text_width_to_font_size
3 from data.utils.browser_helpers import get_winner_string
4 from data.widgets.board_thumbnail import BoardThumbnail
5 from data.utils.asset_helpers import scale_and_cache
6 from data.widgets.bases.widget import _Widget
8 FONT_DIVISION = 7
10 class BrowserItem(_Widget):
     def __init__(self, relative_width, game, **kwargs):
11
           {\tt super().\_init\_\_(relative\_size=(relative\_width\;,\; relative\_width\;*\;\;2)\;,}
12
      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),
18
               scale_mode='height',
19
               relative_width = relative_width,
2.0
21
               fen_string=self._game['final_fen_string']
           )
22
23
24
           self.set_image()
          self.set_geometry()
25
26
      def get_text_to_render(self):
27
           depth_to_text = {
28
29
               2: 'EASY',
               3: 'MEDIUM',
4: 'HARD'
30
3.1
          }
33
          format_moves = lambda no_of_moves: int(no_of_moves / 2) if (no_of_moves /
3.4
       2 % 1 == 0) else round(no_of_moves / 2, 1)
3.5
           if self._game['cpu_enabled'] == 1:
36
              depth_text = depth_to_text[self._game['cpu_depth']]
37
               cpu_text = f'PVC ({depth_text})'
38
39
           else:
               cpu_text = 'PVP'
40
41
42
           return [
              cpu_text.
43
               self._game['created_dt'].strftime('%Y-%m-%d %H:%M:%S'),
44
               f'WINNER: {get_winner_string(self._game['winner'])}',
45
               f'NO. MOVES: {format_moves(self._game['number_of_ply'])}'
46
           ٦
47
48
      def set_image(self):
49
           self.image = pygame.Surface(self.size, pygame.SRCALPHA)
50
      resized_board = scale_and_cache(self._board_thumbnail.image, (self.size [0], self.size [0] * 0.8))
51
           self.image.blit(resized_board, (0, 0))
53
           get_line_y = lambda line: (self.size[0] * 0.8) + ((self.size[0] * 0.8) /
54
      FONT_DIVISION) * (line + 0.5)
5.5
           text_to_render = self.get_text_to_render()
56
57
          for index, text in enumerate(text_to_render):
5.8
59
               self._font.render_to(self.image, (0, get_line_y(index)), text, fgcolor
      =self._text_colour, size=self.font_size)
60
      def process_event(self, event):
61
62
          pass
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.browser_item import BrowserItem
4 from data.constants import BrowserEventType
5 from data.components.custom_event import CustomEvent
7 WIDTH_FACTOR = 0.3
```

```
9 class BrowserStrip(_Widget):
      def __init__(self, relative_height, games_list, **kwargs):
10
           super().__init__(relative_size=None, **kwargs)
           self._relative_item_width = relative_height / 2
12
13
           self._get_rect = None
14
           self._games_list = []
1.5
           self._items_list = []
16
           self._selected_index = None
17
18
           self.initialise_games_list(games_list)
20
21
      @property
      def item_width(self):
22
           return self._relative_item_width * self.surface_size[1]
23
24
25
      @property
      def size(self):
26
          if self._get_rect:
27
              height = self._get_rect().height
28
29
               height = 0
30
            \mbox{width = } \mbox{max(0, len(self.\_games\_list) * (self.item\_width + self.margin) + } 
3.1
      self.margin)
32
           return (width, height)
33
34
      def register_get_rect(self, get_rect_func):
35
           self._get_rect = get_rect_func
36
37
      def initialise_games_list(self, games_list):
38
39
          self._items_list = []
           self._games_list = games_list
40
          self._selected_index = None
41
42
          for game in games_list:
43
               browser_item = BrowserItem(relative_position=(0, 0), game=game,
44
      relative_width=self._relative_item_width)
               self._items_list.append(browser_item)
45
46
           self.set_image()
47
           self.set_geometry()
48
49
      def set_image(self):
50
           self.image = pygame.Surface(self.size, pygame.SRCALPHA)
5.1
           browser_list = []
52
53
           for index, item in enumerate(self._items_list):
54
55
               item.set_image()
               browser\_list.append((item.image\,,\,\,(index\,\,*\,\,(self.item\_width\,\,+\,\,self\,.
56
      margin) + self.margin, self.margin)))
57
           self.image.blits(browser_list)
5.8
           if self._selected_index is not None:
60
               border_position = (self._selected_index * (self.item_width + self.
61
      margin), 0)
               border_size = (self.item_width + 2 * self.margin, self.size[1])
62
               pygame.draw.rect(self.image, (255, 255, 255), (*border_position, *
63
      border_size), width=int(self.item_width / 20))
```

64

```
65
      def set_geometry(self):
          super().set_geometry()
66
          for item in self._items_list:
67
              item.set_geometry()
69
70
      def set_surface_size(self, new_surface_size):
          super().set_surface_size(new_surface_size)
71
73
          for item in self. items list:
74
               item.set_surface_size(new_surface_size)
7.5
76
      def process_event(self, event, scrolled_pos):
7.7
          parent_pos = self._get_rect().topleft
          self.rect.topleft = parent_pos
7.8
          if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
8.0
81
               self._selected_index = None
               self.set_image()
82
               return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
83
      selected_index=None)
84
          if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.
8.5
      pos):
               relative_mouse_pos = (event.pos[0] - parent_pos[0], event.pos[1] -
86
      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,
89
      selected_index=self._selected_index)
1 import pygame
2 from data.widgets.reactive_icon_button import ReactiveIconButton
3 from data.components.custom_event import CustomEvent
4 from data.widgets.bases.circular import _Circular
5 from data.widgets.bases.widget import _Widget
6 from data.constants import Miscellaneous
7 from data.assets import GRAPHICS, SFX
9 class Carousel(_Circular, _Widget):
      def __init__(self, event, widgets_dict, **kwargs):
10
           _Circular.__init__(self, items_dict=widgets_dict)
          _Widget.__init__(self, relative_size=None, **kwargs)
12
13
14
          max_widget_size = (
              max([widget.rect.width for widget in widgets_dict.values()]),
15
16
               max([widget.rect.height for widget in widgets_dict.values()])
17
18
          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.
20
      arrow_size[0])) / self.surface_size[1], (max_widget_size[1]) / self.
      surface_size[1])
21
           self._left_arrow = ReactiveIconButton(
              relative_position = (0, 0),
23
               relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
24
               scale_mode='height',
25
              base_icon = GRAPHICS['left_arrow_base'],
26
              hover_icon = GRAPHICS['left_arrow_hover'],
27
              press_icon=GRAPHICS['left_arrow_press'],
28
```

```
event = CustomEvent(Miscellaneous.PLACEHOLDER),
               sfx=SFX['carousel_click']
3.0
           )
3.1
           self._right_arrow = ReactiveIconButton(
               relative_position = (0, 0),
33
               relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
34
               scale_mode='height',
35
               base_icon = GRAPHICS['right_arrow_base'],
36
               hover_icon = GRAPHICS['right_arrow_hover'],
37
               press_icon=GRAPHICS['right_arrow_press'],
38
               {\tt event=CustomEvent} \; (\; {\tt Miscellaneous} \; . \; {\tt PLACEHOLDER} \; ) \; ,
39
40
               sfx=SFX['carousel_click']
           )
41
42
           self._event = event
43
           self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
44
45
46
           self.set_image()
           self.set_geometry()
47
48
49
      @property
      def max_widget_size(self):
5.0
           return (self._relative_max_widget_size[0] * self.surface_size[1], self.
51
       _relative_max_widget_size[1] * self.surface_size[1])
52
       @property
53
54
       def arrow_size(self):
55
           height = self.max_widget_size[1] * 0.75
           width = (GRAPHICS['left_arrow_base'].width / GRAPHICS['left_arrow_base'].
56
      height) * height
           return (width, height)
5.8
59
      @property
60
      def size(self):
           return ((self.arrow_size[0] + self.margin) * 2 + self.max_widget_size[0],
6.1
       self.max_widget_size[1])
62
63
       @property
       def left_arrow_position(self):
64
          return (0, (self.size[1] - self.arrow_size[1]) / 2)
65
66
67
       @property
      \begin{array}{lll} \textbf{def} & \texttt{right\_arrow\_position(self):} \\ \end{array}
68
           return (self.size[0] - self.arrow_size[0], (self.size[1] - self.arrow_size
69
       [1]) / 2)
       def set_image(self):
7.1
           self.image = pygame.transform.scale(self._empty_surface, self.size)
72
           self.image.fill(self._fill_colour)
73
74
7.5
           if self.border_width:
               pygame.draw.rect(self.image, self._border_colour, (0, 0, *self.size),
76
      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)
79
80
81
           self.current_item.set_image()
           self.image.blit(self.current_item.image, ((self.size[0] - self.
82
       current_item.rect.size[0]) / 2, (self.size[1] - self.current_item.rect.size
       [1]) / 2))
```

83

```
self._right_arrow.set_image()
           self.image.blit(self._right_arrow.image, self.right_arrow_position)
8.5
86
       def set_geometry(self):
           super().set_geometry()
88
89
90
           self.current_item.set_geometry()
           self._left_arrow.set_geometry()
91
92
           self._right_arrow.set_geometry()
93
94
           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])
           {\tt self.\_right\_arrow.rect.topleft} \ = \ ({\tt self.position} \ [0] \ + \ {\tt self} \, .
96
       right_arrow_position[0], self.position[1] + self.right_arrow_position[1])
97
98
       def set_surface_size(self, new_surface_size):
           super().set_surface_size(new_surface_size)
99
           self._left_arrow.set_surface_size(new_surface_size)
100
           self._right_arrow.set_surface_size(new_surface_size)
101
           for item in self._items_dict.values():
103
                item.set_surface_size(new_surface_size)
104
105
       def process_event(self, event):
106
107
           self.current_item.process_event(event)
108
           left_arrow_event = self._left_arrow.process_event(event)
109
           right_arrow_event = self._right_arrow.process_event(event)
           if left_arrow_event:
                self.set_previous_item()
                self.current_item.set_surface_size(self._raw_surface_size)
113
114
115
           elif right_arrow_event:
                self.set_next_item()
116
                self.current_item.set_surface_size(self._raw_surface_size)
118
           if left_arrow_event or right_arrow_event:
               self.set_image()
120
                self.set_geometry()
121
                return CustomEvent(**vars(self._event), data=self.current_key)
124
           elif event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
125
       MOUSEMOTION]:
126
                self.set image()
                self.set_geometry()
127
 1 import pygame
 2 from data.widgets.bases.widget import _Widget
 3 from data.utils.board_helpers import create_board
 4 from data.utils.data_helpers import get_user_settings
 5 from data.constants import CursorMode
 6 from data.managers.cursor import cursor
 8 class Chessboard(_Widget):
      def __init__(self, relative_width, change_cursor=True, **kwargs):
           super().__init__(relative_size=(relative_width, relative_width * 0.8), **
10
       kwargs)
           self._board_surface = None
           self._change_cursor = change_cursor
13
```

```
self._cursor_is_hand = False
14
           self.refresh_board()
16
           self.set_image()
           self.set_geometry()
18
1.9
20
      def refresh_board(self):
           user_settings = get_user_settings()
2.1
           self._board_surface = create_board(self.size, user_settings['
22
      primaryBoardColour'], user_settings['secondaryBoardColour'])
23
24
           self.set_image()
25
      def set_image(self):
26
           self.image = pygame.transform.smoothscale(self._board_surface, self.size)
27
28
29
      def process_event(self, event):
30
           if self._change_cursor and event.type in [pygame.MOUSEMOTION, pygame.
      MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN]:
               current_cursor = cursor.get_mode()
31
32
               if self.rect.collidepoint(event.pos):
3.3
                    if current_cursor == CursorMode.ARROW:
34
                        cursor.set_mode(CursorMode.OPENHAND)
3.5
                    elif current_cursor == CursorMode.OPENHAND and (pygame.mouse.
36
      get_pressed()[0] is True or event.type == pygame.MOUSEBUTTONDOWN):
                        cursor.set_mode(CursorMode.CLOSEDHAND)
37
                    elif current_cursor == CursorMode.CLOSEDHAND and (pygame.mouse.
      get_pressed()[0] is False or event.type == pygame.MOUSEBUTTONUP):
                        cursor.set_mode(CursorMode.OPENHAND)
3.9
                   if current_cursor == CursorMode.OPENHAND or (current_cursor ==
41
      {\tt CursorMode.CLOSEDHAND} \quad {\tt and} \quad {\tt event.type} \  \, {\tt ==} \quad {\tt pygame.MOUSEBUTTONUP)}:
                        cursor.set_mode(CursorMode.ARROW)
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.constants import WidgetState
6 class ColourButton(_Pressable, _Widget):
7     def __init__(self, event, **kwargs):
           _Pressable.__init__(
               self,
9
10
                event = event.
               hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
11
12
               down_func=lambda: self.set_state_colour(WidgetState.PRESS),
               up_func=lambda: self.set_state_colour(WidgetState.BASE),
               sfx = None
14
15
16
           _Widget.__init__(self, **kwargs)
           self._empty_surface = pygame.Surface(self.size)
18
19
           self.initialise_new_colours(self._fill_colour)
20
           self.set_state_colour(WidgetState.BASE)
22
23
           self.set_image()
24
           self.set_geometry()
2.5
26
      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 import pygame
2 from data.widgets.bases.widget import _Widget
4 class _ColourDisplay(_Widget):
     def __init__(self, **kwargs):
    super().__init__(**kwargs)
           self._colour = None
           self._empty_surface = pygame.Surface(self.size)
10
11
12
      def set_colour(self, new_colour):
           self._colour = new_colour
13
14
      def set_image(self):
15
           self.image = pygame.transform.scale(self._empty_surface, self.size)
16
17
           self.image.fill(self._colour)
      def process_event(self, event):
19
           pass
20
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.colour_square import _ColourSquare 4 from data.widgets.colour_slider import _ColourSlider
5 from data.widgets.colour_display import _ColourDisplay
6 from data.components.custom_event import CustomEvent
8 class ColourPicker(_Widget):
      def __init__(self, relative_width, event_type, **kwargs):
           super().__init__(relative_size=(relative_width, relative_width),
10
       scale_mode='width', **kwargs)
           self.image = pygame.Surface(self.size)
self.rect = self.image.get_rect()
12
13
           self._square = _ColourSquare(
15
16
                parent=self,
                relative_position = (0.1, 0.1),
17
                relative_width=0.5,
18
19
                event_type=event_type
20
21
           self._square.set_colour(kwargs.get('selected_colour'))
           self._slider = _ColourSlider(
23
               parent=self,
24
                relative_position=(0.0, 0.7),
25
                relative_width=1.0,
26
27
                border_width = self.border_width,
                border_colour=self._border_colour
28
29
           self._slider.set_colour(kwargs.get('selected_colour'))
31
32
           self._display = _ColourDisplay(
               parent=self,
33
                relative_position = (0.7, 0.1),
34
35
                relative_size=(0.2, 0.5)
36
```

```
self._display.set_colour(kwargs.get('selected_colour'))
37
38
          self._event_type = event_type
39
          self._hover_event_type = event_type
40
41
42
          self.set_image()
          self.set_geometry()
43
44
      def global_to_relative_pos(self, global_pos):
45
          return (global_pos[0] - self.position[0], global_pos[1] - self.position
46
      [1])
47
      def set_image(self):
48
           self.image = pygame.Surface(self.size)
49
          self.image.fill(self._fill_colour)
50
5.1
          self._square.set_image()
52
           self._square.set_geometry()
53
          \verb|self.image.blit(self.\_square.image, self.global\_to\_relative\_pos(self.
54
      _square.position))
55
          self._slider.set_image()
5.6
           self._slider.set_geometry()
          self.image.blit(self._slider.image, self.global_to_relative_pos(self.
5.8
      _slider.position))
6.0
           self._display.set_image()
61
           self._display.set_geometry()
          self.image.blit(self._display.image, self.global_to_relative_pos(self.
62
      _display.position))
63
          pygame.draw.rect(self.image, self._border_colour, (0, 0, self.size[0],
64
      self.size[1]), width=int(self.border_width))
      def set_surface_size(self, new_surface_size):
66
           super().set_surface_size(new_surface_size)
67
           self._square.set_surface_size(self.size)
68
           self._slider.set_surface_size(self.size)
69
          self._display.set_surface_size(self.size)
71
      def get_picker_position(self):
72
          return self.position
73
7.4
75
      def process_event(self, event):
          slider_colour = self._slider.process_event(event)
76
          square_colour = self._square.process_event(event)
78
79
          if square_colour:
               self._display.set_colour(square_colour)
80
81
               self.set_image()
82
          if slider_colour:
               self._square.set_colour(slider_colour)
84
               self.set_image()
8.5
           if event.type in [pygame.MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN, pygame.
87
      MOUSEMOTION] and self.rect.collidepoint(event.pos):
               return CustomEvent(self._event_type, colour=square_colour)
1 import pygame
2 from data.utils.widget_helpers import create_slider_gradient
3 from data.utils.asset_helpers import smoothscale_and_cache
```

```
{\tt 4 from data.widgets.slider\_thumb \ import \ \_SliderThumb}
5 from data.widgets.bases.widget import _Widget
6 from data.constants import WidgetState
8 class _ColourSlider(_Widget):
      def __init__(self, relative_width, **kwargs):
          super().__init__(relative_size=(relative_width, relative_width * 0.2), **
10
      kwargs)
11
12
          # Initialise slider thumb.
          self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
13
      _border_colour)
14
          self._selected_percent = 0
15
          self._last_mouse_x = None
16
          self._gradient_surface = create_slider_gradient(self.gradient_size, self.
18
      border_width, self._border_colour)
          self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
19
21
      @property
      def gradient_size(self):
22
          return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
23
24
25
      @property
      def gradient_position(self):
26
          return (self.size[1] / 2, self.size[1] / 4)
27
28
      @property
29
      def thumb_position(self):
3.0
31
          return (self.gradient_size[0] * self._selected_percent, 0)
32
33
      @property
      def selected_colour(self):
34
          colour = pygame.Color(0)
3.5
          colour.hsva = (int(self._selected_percent * 360), 100, 100, 100)
36
          return colour
37
3.8
      def calculate_gradient_percent(self, mouse_pos):
39
40
          Calculate what percentage slider thumb is at based on change in mouse
41
      position.
42
43
              mouse_pos (list[int, int]): Position of mouse on window screen.
44
45
46
          float: Slider scroll percentage.
47
48
49
          if self._last_mouse_x is None:
5.0
              return
51
          x_change = (mouse_pos[0] - self._last_mouse_x) / (self.gradient_size[0] -
52
      2 * self.border_width)
          return max(0, min(self._selected_percent + x_change, 1))
54
      def relative_to_global_position(self, position):
55
56
          Transforms position from being relative to widget rect, to window screen.
5.7
58
59
          Args:
              position (list[int, int]): Position relative to widget rect.
60
```

```
61
           Returns:
62
              list[int, int]: Position relative to window screen.
63
           relative_x , relative_y = position
65
           return (relative_x + self.position[0], relative_y + self.position[1])
66
67
       def set_colour(self, new_colour):
68
69
           Sets selected_percent based on the new colour's hue.
70
71
           Args:
           new_colour (pygame.Color): New slider colour.
73
7.4
           colour = pygame.Color(new_colour)
           hue = colour.hsva[0]
7.6
           self._selected_percent = hue / 360
7.7
78
           self.set_image()
7.9
       def set_image(self):
80
81
           Draws colour slider to widget image.
82
83
           # Scales initalised gradient surface instead of redrawing it everytime
84
       set_image is called
           gradient_scaled = smoothscale_and_cache(self._gradient_surface, self.
85
       gradient_size)
           self.image = pygame.transform.scale(self._empty_surface, (self.size))
87
           self.image.blit(gradient_scaled, self.gradient_position)
88
89
           # Resets thumb colour, image and position, then draws it to the widget
9.0
       image
           self._thumb.initialise_new_colours(self.selected_colour)
91
           self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
92
       border_width)
           self._thumb.set_position(self.relative_to_global_position((self.
93
       thumb_position[0], self.thumb_position[1])))
           thumb_surface = self._thumb.get_surface()
95
96
           self.image.blit(thumb_surface, self.thumb_position)
97
       def process_event(self, event):
98
99
           Processes Pygame events.
100
101
           Args:
               event (pygame.Event): Event to process.
103
104
           Returns:
           pygame.Color: Current colour slider is displaying.
106
107
           if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
108
       MOUSEBUTTONUP1:
           # Gets widget state before and after event is processed by slider thumb
111
           before_state = self._thumb.state
112
113
           self._thumb.process_event(event)
114
           after_state = self._thumb.state
115
           # If widget state changes (e.g. hovered -> pressed), redraw widget
116
```

```
if before_state != after_state:
117
               self.set_image()
118
119
           if event.type == pygame.MOUSEMOTION:
120
               if self._thumb.state == WidgetState.PRESS:
121
                    # Recalculates slider colour based on mouse position change
                    selected_percent = self.calculate_gradient_percent(event.pos)
123
                    self._last_mouse_x = event.pos[0]
124
125
                    if selected_percent is not None:
126
                        self._selected_percent = selected_percent
127
128
                        return self.selected_colour
129
130
           if event.type == pygame.MOUSEBUTTONUP:
131
               # When user stops scrolling, return new slider colour
132
133
               self._last_mouse_x = None
134
               return self.selected_colour
135
           if event.type == pygame.MOUSEBUTTONDOWN or before_state != after_state:
136
               # Redraws widget when slider thumb is hovered or pressed
137
               return self.selected_colour
138
1 import pygame
 2 from data.widgets.bases.widget import _Widget
 3 from data.utils.widget_helpers import create_square_gradient
 5 class _ColourSquare(_Widget):
      def __init__(self, relative_width, **kwargs):
           super().__init__(relative_size=(relative_width, relative_width),
       scale_mode='width', **kwargs)
 8
           self._colour = None
       def set_colour(self, new_colour):
11
12
           self._colour = pygame.Color(new_colour)
13
      def get_colour(self):
14
15
           return self._colour
16
1.7
       def set_image(self):
           self.image = create_square_gradient(side_length=self.size[0], colour=self.
       _colour)
19
20
       def process_event(self, event):
           if event.type == pygame.MOUSEBUTTONDOWN:
2.1
               relative_mouse_pos = (event.pos[0] - self.position[0], event.pos[1] -
22
       self.position[1])
23
                   0 > relative_mouse_pos[0] or
25
                    self.size[0] < relative_mouse_pos[0] or
26
                    0 > relative_mouse_pos[1] or
27
                    self.size[1] < relative_mouse_pos[1]
28
29
               ): return None
30
               self.set_colour(self.image.get_at(relative_mouse_pos))
31
               return self._colour
33
34
           return None
```

```
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.constants import WidgetState
{\tt 5} \  \  \, \textbf{from} \  \  \, \textbf{data.utils.data\_helpers} \  \  \, \textbf{import} \  \  \, \textbf{get\_user\_settings}
6 from data.utils.font_helpers import text_width_to_font_size,
      text_height_to_font_size
7 from data.assets import GRAPHICS, FONTS
9 user_settings = get_user_settings()
10
11 class Dropdown(_Pressable, _Widget):
     def __init__(self, word_list, event=None, **kwargs):
12
13
           _Pressable.__init__(
14
               self.
               event = event.
1.5
               hover_func=self.hover_func,
16
               down_func=lambda: self.set_state_colour(WidgetState.PRESS),
               up_func=self.up_func,
18
               sfx = None
20
           _Widget.__init__(self, relative_size=None, **kwargs)
2.1
           if kwargs.get('relative_width'):
23
               self._relative_font_size = text_width_to_font_size(max(word_list, key=
24
      len), self._font, kwargs.get('relative_width') * self.surface_size[0] - self.
      margin) / self.surface_size[1]
25
           elif kwargs.get('relative_height'):
               self._relative_font_size = text_height_to_font_size(max(word_list, key
26
      =len), self._font, kwargs.get('relative_height') * self.surface_size[1] - self
      .margin) / self.surface_size[1]
27
           self._word_list = [word_list[0].capitalize()]
28
          self._word_list_copy = [word.capitalize() for word in word_list]
29
3.0
           self._expanded = False
31
          self._hovered_index = None
32
3.3
           self._empty_surface = pygame.Surface((0, 0))
          self._background_colour = self._fill_colour
35
36
          self.initialise_new_colours(self._fill_colour)
37
          self.set_state_colour(WidgetState.BASE)
38
39
           self.set_image()
40
41
          self.set_geometry()
      @property
43
44
      def size(self):
45
          max_word = sorted(self._word_list_copy, key=len)[-1]
           max_word_rect = self._font.get_rect(max_word, size=self.font_size)
46
           all_words_rect = pygame.FRect(0, 0, max_word_rect.size[0], (max_word_rect.
47
      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)
48
           return (all_words_rect.size[0] + max_word_rect.size[1], all_words_rect.
      size[1])
5.0
      def get_selected_word(self):
51
          return self._word_list[0].lower()
52
53
      def toggle_expanded(self):
54
          if self._expanded:
55
```

```
self._word_list = [self._word_list_copy[0]]
57
           else:
                self._word_list = [*self._word_list_copy]
58
           self._expanded = not(self._expanded)
60
61
       def hover_func(self):
62
           mouse_position = pygame.mouse.get_pos()
63
           relative_position = (mouse_position[0] - self.position[0], mouse_position
64
       [1] - self.position[1])
           self._hovered_index = self.calculate_hovered_index(relative_position)
6.5
           self.set_state_colour(WidgetState.HOVER)
67
68
       def set_selected_word(self, word):
            index = self._word_list_copy.index(word.capitalize())
69
           selected_word = self._word_list_copy.pop(index)
7.0
71
           self._word_list_copy.insert(0, selected_word)
72
           if self._expanded:
7.3
               self._word_list.pop(index)
74
                self._word_list.insert(0, selected_word)
75
7.6
           else:
                self._word_list = [selected_word]
7.7
7.8
79
           self.set_image()
80
81
       def up_func(self):
           if self.get_widget_state() == WidgetState.PRESS:
82
                if self._expanded and self._hovered_index is not None:
83
                    \verb|self.set_selected_word(self._word_list_copy[self._hovered_index]||\\
8.4
85
                self.toggle_expanded()
86
87
           self._hovered_index = None
88
89
           self.set_state_colour(WidgetState.BASE)
90
           self.set_geometry()
91
92
       def calculate_hovered_index(self, mouse_pos):
93
           return int(mouse_pos[1] // (self.size[1] / len(self._word_list)))
94
95
96
       def set_image(self):
           text_surface = pygame.transform.scale(self._empty_surface, self.size)
97
98
           self.image = text_surface
99
           fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
100
           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.
104
       _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)
```

```
111
           if self._expanded:
               self.image.blit(closed_arrow_surface, arrow_position)
113
               self.image.blit(open_arrow_surface, arrow_position)
115
           for index, word in enumerate(self._word_list):
116
               word_position = (self.margin, self.margin + (word_box_height + self.
117
       margin) * index)
               self._font.render_to(self.image, word_position, word, fgcolor=self.
118
       _text_colour, size=self.font_size)
119
           if self._hovered_index is not None:
               overlay_surface = pygame.Surface((self.size[0], word_box_height + 2 *
121
       self.margin), pygame.SRCALPHA)
               overlay_surface.fill((*self._fill_colour.rgb, 128))
               overlay_position = (0, (word_box_height + self.margin) * self.
       _hovered_index)
124
               self.image.blit(overlay_surface, overlay_position)
 1 import pygame
 2 from data.widgets.bases.widget import _Widget
 3 from data.utils.widget_helpers import create_text_box
 5 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:
 9
               aspect_ratio = icon.width / icon.height
10
               self._relative_size = (self._relative_size[1] * aspect_ratio, self.
       _relative_size[1])
12
           self._icon = icon
           self._is_mask = is_mask
14
           self._stretch = stretch
1.5
           self._smooth = smooth
           self._box_colours = box_colours
17
18
           self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
20
21
           self.set_image()
           self.set_geometry()
22
23
24
       def set_icon(self, icon):
           self._icon = icon
25
26
           self.set_image()
27
28
      def set_image(self):
           if self._box_colours:
29
               self.image = create_text_box(self.size, self.border_width, self.
30
       _box_colours)
31
           else:
               self.image = pygame.transform.scale(self._empty_surface, self.size)
32
33
               if self._fill_colour:
                   pygame.draw.rect(self.image, self._fill_colour, self.image.
35
       get_rect(), border_radius=int(self.border_radius))
36
3.7
           if self _stretch:
38
               if self._smooth:
                   scaled_icon = pygame.transform.smoothscale(self._icon, (self.size
3.9
```

```
[0] - (2 * self.margin), self.size[1] - (2 * self.margin)))
40
                else:
                   scaled_icon = pygame.transform.scale(self._icon, (self.size[0] -
41
       (2 * self.margin), self.size[1] - (2 * self.margin)))
42
               icon_position = (self.margin, self.margin)
43
44
           else:
               max_height = self.size[1] - (2 * self.margin)
max_width = self.size[0] - (2 * self.margin)
45
46
               scale_factor = min(max_width / self._icon.width, max_height / self.
47
       _icon.height)
48
                if self._smooth:
49
                   scaled_icon = pygame.transform.smoothscale_by(self._icon, (
5.0
       scale_factor, scale_factor))
5.1
                else:
                    scaled_icon = pygame.transform.scale_by(self._icon, (scale_factor,
52
       scale_factor))
               icon_position = ((self.size[0] - scaled_icon.width) / 2, (self.size[1]
53
        - scaled_icon.height) / 2)
54
           if self._is_mask:
5.5
               self.image.blit(scaled_icon, icon_position, None, pygame.
56
       BLEND_RGBA_MULT)
57
           else:
               self.image.blit(scaled_icon, icon_position)
58
59
60
           if self._box_colours is None and self.border_width:
               pygame.draw.rect(self.image, self._border_colour, self.image.get_rect
61
       (), width=int(self.border_width), border_radius=int(self.border_radius))
62
       def process_event(self, event):
63
64
           pass
{\scriptsize \texttt{1} \  \  \, \textbf{from} \  \  \, \textbf{data.widgets.bases.pressable} \  \  \, \textbf{import} \  \  \, \textbf{\_Pressable}}
2 from data.widgets.bases.box import _Box
3 from data.widgets.icon import Icon
4 from data.constants import WidgetState, RED_BUTTON_COLOURS
6 class IconButton(_Box, _Pressable, Icon):
      def __init__(self, event, box_colours=RED_BUTTON_COLOURS, **kwargs):
           _Box.__init__(self, box_colours=box_colours)
           _Pressable.__init__(
9
               self,
10
11
                event = event.
               hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
                down_func=lambda: self.set_state_colour(WidgetState.PRESS),
13
14
               up_func=lambda: self.set_state_colour(WidgetState.BASE),
15
           Icon.__init__(self, box_colours=box_colours[WidgetState.BASE], **kwargs)
16
           self.initialise_new_colours(self._fill_colour)
18
           \verb|self.set_state_colour(WidgetState.BASE)| \\
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.utils.font_helpers import width_to_font_size
5 class MoveList(_Widget):
      def __init__(self, relative_width, minimum_height=0, move_list=[], **kwargs):
6
           super().__init__(relative_size=None, **kwargs)
```

```
self._relative_width = relative_width * self.surface_size[0] / self.
9
      surface_size[1]
          self._relative_minimum_height = minimum_height / self.surface_size[1]
          self._move_list = move_list
11
          self._relative_font_size = width_to_font_size(self._font, self.
12
      surface_size[0] / 3.5) / self.surface_size[1]
13
          self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
14
15
16
          self.set_image()
17
          self.set_geometry()
18
      @property
19
20
      def size(self):
          font_metrics = self._font.get_metrics('j', size=self.font_size)
21
22
23
          width = self._relative_width * self.surface_size[1]
24
          minimum_height = self._relative_minimum_height * self.surface_size[1]
          row_gap = font_metrics[0][3] - font_metrics[0][2]
25
          number_of_rows = 2 * ((len(self._move_list) + 1) // 2) + 1
26
27
          return (width, max(minimum_height, row_gap * number_of_rows))
28
29
      def register_get_rect(self, get_rect_func):
3.0
31
          pass
32
33
      def reset_move_list(self):
34
          self._move_list = []
          self.set_image()
35
          self.set_geometry()
36
37
      def append_to_move_list(self, new_move):
38
39
          self._move_list.append(new_move)
          self.set_image()
40
          self.set_geometry()
41
42
      def pop_from_move_list(self):
43
          self._move_list.pop()
44
          self.set_image()
          self.set_geometry()
46
47
48
      def set_image(self):
          self.image = pygame.transform.scale(self._empty_surface, self.size)
49
50
          self.image.fill(self._fill_colour)
51
          font_metrics = self._font.get_metrics('j', size=self.font_size)
52
          row_gap = font_metrics[0][3] - font_metrics[0][2]
53
54
55
          for index, move in enumerate(self._move_list):
56
               if index % 2 == 0:
                   text_position = (self.size[0] / 7, row_gap * (1 + 2 * (index // 2)
57
      ))
58
                   text_position = (self.size[0] * 4 / 7, row_gap * (1 + 2 * (index
59
      // 2)))
60
               self._font.render_to(self.image, text_position, text=move, size=self.
61
      font_size, fgcolor=self._text_colour)
62
               move_number = (index // 2) + 1
63
               move_number_position = (self.size[0] / 14, row_gap * (1 + 2 * (index
64
      // 2)))
```

```
self._font.render_to(self.image, move_number_position, text=str(
      move_number), size=self.font_size, fgcolor=self._text_colour)
66
      def process_event(self, event, scrolled_pos=None):
68
1 import pygame
2 from data.constants import WidgetState, LOCKED_BLUE_BUTTON_COLOURS,
      LOCKED_RED_BUTTON_COLOURS, RED_BUTTON_COLOURS, BLUE_BUTTON_COLOURS
3 from data.components.custom_event import CustomEvent
4 from data widgets bases circular import _Circular
5 from data.widgets.icon_button import IconButton
6 from data.widgets.bases.box import _Box
8 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)
11
12
      self._fill_colour_copy = self._fill_colour
13
14
1.5
      self._locked = None
17
    def set_locked(self, is_locked):
18
      self._locked = is_locked
      if self._locked:
        r, g, b, a = pygame.Color(self._fill_colour_copy).rgba
20
        if self._box_colours_dict == BLUE_BUTTON_COLOURS:
21
          _Box.__init__(self, box_colours=LOCKED_BLUE_BUTTON_COLOURS)
22
        elif self._box_colours_dict == RED_BUTTON_COLOURS:
23
          _Box.__init__(self, box_colours=LOCKED_RED_BUTTON_COLOURS)
24
        else:
25
          self.initialise_new_colours((max(r + 50, 0), max(g + 50, 0), max(b + 50,
26
      0), a))
27
      else:
        if self._box_colours_dict == LOCKED_BLUE_BUTTON_COLOURS:
28
          _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
29
        elif self._box_colours_dict == LOCKED_RED_BUTTON_COLOURS:
3.0
31
          _Box.__init__(self, box_colours=RED_BUTTON_COLOURS)
        else:
32
33
          self.initialise_new_colours(self._fill_colour_copy)
     if self.rect.collidepoint(pygame.mouse.get_pos()):
35
        self.set_state_colour(WidgetState.HOVER)
36
37
        self.set_state_colour(WidgetState.BASE)
38
39
40
    def set_next_icon(self):
      super().set_next_item()
41
     self._icon = self.current_item
     self.set_image()
43
44
    def process_event(self, event):
45
     widget_event = super().process_event(event)
46
47
      if widget_event:
48
        return CustomEvent(**vars(widget_event), data=self.current_key)
49
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.states.game.components.piece_sprite import PieceSprite
```

```
4 from data.constants import Score, Rotation, WidgetState, Colour,
      BLUE_BUTTON_COLOURS, RED_BUTTON_COLOURS
5 from data.utils.widget_helpers import create_text_box
6 from data.utils.asset_helpers import scale_and_cache
8 class PieceDisplay(_Widget):
     def __init__(self, active_colour, **kwargs):
          super().__init__(**kwargs)
1.0
11
12
          self._active_colour = active_colour
          self._piece_list = []
13
          self._piece_surface = None
          self._box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour
15
       == Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]
          self.initialise_piece_surface()
18
19
          self.set_image()
          self.set_geometry()
20
21
22
     def add_piece(self, piece):
          self._piece_list.append(piece)
23
          self._piece_list.sort(key=lambda piece: Score[piece.name])
24
          self.initialise_piece_surface()
2.5
26
27
      def remove_piece(self, piece):
          self._piece_list.remove(piece)
28
29
          self.initialise_piece_surface()
30
      def reset_piece_list(self):
31
32
          self._piece_list = []
          self.initialise_piece_surface()
33
34
35
      def initialise_piece_surface(self):
          self._piece_surface = pygame.Surface((self.size[0] - 2 * self.margin, self
36
      .size[1] - 2 * self.margin), pygame.SRCALPHA)
37
          if (len(self._piece_list) == 0):
3.8
              self.set_image()
39
              return
40
41
          piece_width = min(self.size[1] - 2 * self.margin, (self.size[0] - 2 * self
42
      .margin) / len(self._piece_list))
43
          piece_list = []
44
          for index, piece in enumerate(self._piece_list):
45
               piece_instance = PieceSprite(piece, self._active_colour.
      get_flipped_colour(), Rotation.UP)
47
              piece_instance.set_geometry((0, 0), piece_width)
48
               piece_instance.set_image()
49
              piece_list.append((piece_instance.image, (piece_width * index, (self.
      _piece_surface.height - piece_width) / 2)))
50
           self._piece_surface.fblits(piece_list)
5.1
          self.set_image()
53
54
55
      def set_image(self):
56
          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]
58
```

```
- 2 * self.margin, self.size[1] - 2 * self.margin))
          self.image.blit(resized_piece_surface, (self.margin, self.margin))
59
60
      def process_event(self, event):
61
          pass
62
1 from data.components.custom_event import CustomEvent
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.bases.circular import _Circular
4 from data.widgets.bases.widget import _Widget
5 from data.constants import WidgetState
7 class ReactiveButton(_Pressable, _Circular, _Widget):
     def __init__(self, widgets_dict, event, center=False, **kwargs):
          # Multiple inheritance used here, to combine the functionality of multiple
       super classes
          _Pressable.__init__(
              self,
               event = event,
12
              hover_func=lambda: self.set_to_key(WidgetState.HOVER),
13
               down_func=lambda: self.set_to_key(WidgetState.PRESS),
              up_func=lambda: self.set_to_key(WidgetState.BASE),
15
16
              **kwargs
17
          # Aggregation used to cycle between external widgets
1.8
19
           _Circular.__init__(self, items_dict=widgets_dict)
          _Widget.__init__(self, **kwargs)
20
21
22
          self._center = center
23
          self.initialise_new_colours(self._fill_colour)
24
25
      @property
26
27
      def position(self):
28
          Overrides position getter method, to always position icon in the center if
29
       self._center is True.
30
31
          Returns:
          list[int, int]: Position of widget.
33
34
          position = super().position
35
          if self._center:
36
              self._size_diff = (self.size[0] - self.rect.width, self.size[1] - self
37
      .rect.height)
38
               return (position[0] + self._size_diff[0] / 2, position[1] + self.
      _size_diff[1] / 2)
          else:
39
               return position
40
41
      def set_image(self):
42
43
          Sets current icon to image.
44
45
          self.current_item.set_image()
          self.image = self.current_item.image
47
48
49
      def set_geometry(self):
5.0
51
          Sets size and position of widget.
52
```

```
super().set_geometry()
          self.current_item.set_geometry()
54
          self.current_item.rect.topleft = self.rect.topleft
5.5
      def set_surface_size(self, new_surface_size):
57
5.8
          Overrides base method to resize every widget state icon, not just the
59
      current one.
60
61
          Args:
          new_surface_size (list[int, int]): New surface size.
62
63
          super().set_surface_size(new_surface_size)
64
          for item in self._items_dict.values():
6.5
               item.set_surface_size(new_surface_size)
66
67
68
     def process_event(self, event):
69
          Processes Pygame events.
7.0
71
72
          Args:
              event (pygame.Event): Event to process.
7.3
74
          Returns:
7.5
          CustomEvent: CustomEvent of current item, with current key included
76
          widget_event = super().process_event(event)
7.8
79
          self.current_item.process_event(event)
80
          if widget_event:
8.1
              return CustomEvent(**vars(widget_event), data=self.current_key)
1 from data.widgets.reactive_button import ReactiveButton
2 from data.constants import WidgetState
3 from data.widgets.icon import Icon
5 class ReactiveIconButton(ReactiveButton):
      def __init__(self, base_icon, hover_icon, press_icon, **kwargs):
          # Composition is used here, to initialise the Icon widgets for each widget
       state
          widgets_dict = {
8
               WidgetState.BASE: Icon(
9
                  parent=kwargs.get('parent'),
10
                   relative_size=kwargs.get('relative_size'),
11
                   relative_position = (0, 0),
                   icon=base_icon,
13
14
                   fill_colour=(0, 0, 0, 0),
                   border_width=0,
                   margin=0,
16
17
                   fit_icon=True,
18
               WidgetState.HOVER: Icon(
19
20
                  parent=kwargs.get('parent'),
                   relative_size=kwargs.get('relative_size'),
21
                   relative_position = (0, 0),
22
                   icon=hover_icon,
                   fill_colour=(0, 0, 0, 0),
24
25
                   border_width=0,
26
                   margin=0,
                   fit_icon=True,
27
28
               ) .
               WidgetState.PRESS: Icon(
29
```

```
parent=kwargs.get('parent'),
                   relative_size=kwargs.get('relative_size'),
31
                   relative_position = (0, 0),
32
                   icon=press_icon,
                   fill_colour=(0, 0, 0, 0),
34
                   border_width = 0,
3.5
                   margin=0,
36
                   fit_icon=True,
37
               )
3.8
          }
39
40
41
           super().__init__(
               widgets_dict=widgets_dict,
42
43
               **kwargs
1 import pygame
2 from data.widgets.bases.widget import _Widget
4 class Rectangle(_Widget):
      def __init__(self, visible=False, **kwargs):
           super().__init__(**kwargs)
           self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
9
          self._visible = visible
10
11
           self.set_image()
          self.set_geometry()
12
13
     def set_image(self):
14
           self.image = pygame.transform.scale(self._empty_surface, self.size)
15
16
           if self._visible:
              pygame.draw.rect(self.image, self._fill_colour, self.image.get_rect(),
       border_radius=int(self.border_radius))
19
               if self.border_width:
                   pygame.draw.rect(self.image, self._border_colour, self.image.
2.0
      get_rect(), width=int(self.border_width), border_radius=int(self.border_radius
21
      def process_event(self, event):
22
23
          pass
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.bases.pressable import _Pressable
4 from data.constants import WidgetState, Miscellaneous
6 # self.set_state_colour(WidgetState.HOVER)
7 class _Scrollbar(_Pressable, _Widget):
      def __init__(self, vertical, **kwargs):
           _Pressable.__init__(
               self,
               event = Miscellaneous. PLACEHOLDER,
11
               hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
12
               down_func=self.down_func,
               up_func=self.up_func,
14
15
               prolonged=True,
               sfx = None
16
           _Widget.__init__(self, **kwargs)
18
19
```

```
self._vertical = vertical
          self._last_mouse_px = None
21
22
           self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
24
           self.initialise_new_colours(self._fill_colour)
2.5
          self.set_state_colour(WidgetState.BASE)
26
27
           self.set_image()
28
29
           self.set_geometry()
3.0
31
      def down_func(self):
          if self._vertical:
32
               self._last_mouse_px = pygame.mouse.get_pos()[1]
33
34
               self._last_mouse_px = pygame.mouse.get_pos()[0]
3.5
36
37
           self.set_state_colour(WidgetState.PRESS)
38
      def up_func(self):
39
           self._last_mouse_px = None
40
           self.set_state_colour(WidgetState.BASE)
4.1
42
      def set_relative_position(self, relative_position):
43
44
           self._relative_position = relative_position
           self.set_geometry()
45
46
47
      def set_relative_size(self, new_relative_size):
           self._relative_size = new_relative_size
48
49
50
      def set_image(self):
          self.image = pygame.transform.scale(self._empty_surface, self.size)
5.1
52
53
           if self._vertical:
              rounded_radius = self.size[0] / 2
54
           else:
55
              rounded_radius = self.size[1] / 2
56
5.7
           pygame.draw.rect(self.image, self._fill_colour, (0, 0, self.size[0], self.
58
      size[1]), border_radius=int(rounded_radius))
59
      def process_event(self, event):
60
          before_state = self.get_widget_state()
6.1
           widget_event = super().process_event(event)
62
          after_state = self.get_widget_state()
63
64
          if event.type == pygame.MOUSEMOTION and self._last_mouse_px:
               if self._vertical:
66
                   offset_from_last_frame = event.pos[1] - self._last_mouse_px
67
68
                   self._last_mouse_px = event.pos[1]
6.9
70
                   return offset_from_last_frame
71
                   offset_from_last_frame = event.pos[0] - self._last_mouse_px
                   self._last_mouse_px = event.pos[0]
73
74
                   return offset_from_last_frame
7.5
76
7.7
78
          if widget_event or before_state != after_state:
              return 0
79
```

```
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.scrollbar import _Scrollbar
4 from data.managers.theme import theme
6 SCROLLBAR_WIDTH_FACTOR = 0.05
8 class ScrollArea(_Widget):
      def __init__(self, widget, vertical, scroll_factor=15, **kwargs):
9
          super().__init__(**kwargs)
10
          if vertical is False:
11
12
               self._relative_size = kwargs.get('relative_size')
13
          self._relative_scroll_factor = scroll_factor / self.surface_size[1]
14
15
          self._scroll_percentage = 0
16
17
           self._widget = widget
18
          self._vertical = vertical
19
           self._widget.register_get_rect(self.calculate_widget_rect)
20
21
          if self._vertical:
22
               anchor_x = 'right'
23
               anchor_y = 'top'
24
               scale_mode = 'height'
25
           else:
26
               anchor_x = 'left'
27
               anchor_y = 'bottom'
28
               scale_mode = 'width'
29
3.0
31
           self._scrollbar = _Scrollbar(
              parent=self,
32
33
               relative_position = (0, 0),
               relative_size=None,
34
               anchor_x=anchor_x,
3.5
36
               anchor_y = anchor_y ,
               fill_colour=theme['borderPrimary'],
37
               scale_mode = scale_mode,
3.8
               vertical=vertical,
          )
40
41
          self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
42
43
44
           self.set_image()
          self.set_geometry()
45
46
47
     @property
      def scroll_factor(self):
48
          return self._relative_scroll_factor * self.surface_size[1]
49
50
      @property
5.1
52
      def scrollbar_size(self):
          if self._vertical:
53
              return (self.size[0] * SCROLLBAR_WIDTH_FACTOR, min(1, self.size[1] /
5.4
      self._widget.rect.height) * self.size[1])
          else:
55
              return (min(1, self.size[0] / (self._widget.rect.width + 0.001)) *
56
      self.size[0], self.size[1] * SCROLLBAR_WIDTH_FACTOR)
5.7
58
      Oproperty
      def size(self):
59
          if self._vertical is False:
60
```

```
return (self._relative_size[0] * self.surface_size[0], self.
61
       _relative_size[1] * self.surface_size[1]) # scale with horizontal width to
       always fill entire length of screen
           else:
62
               return super().size
63
64
       def calculate_scroll_percentage(self, offset, scrollbar=False):
65
           if self._vertical:
66
               widget_height = self._widget.rect.height
67
68
               if widget_height < self.size[1]:</pre>
6.9
                    return 0
71
72
               if scrollbar:
                    self._scroll_percentage += offset / (self.size[1] - self.
       scrollbar_size[1] + 0.001)
74
               else:
                    max_scroll_height = widget_height - self.size[1]
                    current_scroll_height = self._scroll_percentage *
76
       max_scroll_height
                   self._scroll_percentage = (current_scroll_height + offset) / (
77
       max_scroll_height + 0.001)
           else:
               widget_width = self._widget.rect.width
7.9
80
               if widget_width < self.size[0]:</pre>
81
82
                   return 0
83
               if scrollbar:
84
                    self._scroll_percentage += offset / (self.size[0] - self.
8.5
       scrollbar_size[0] + 0.001)
               else:
86
87
                    max_scoll_width = widget_width - self.size[0]
                    current_scroll_width = self._scroll_percentage * max_scoll_width
88
                    self._scroll_percentage = (current_scroll_width + offset) /
89
       max_scoll_width
90
           return min(1, max(0, self._scroll_percentage))
9.1
92
       def calculate_widget_rect(self):
93
           widget_position = self.calculate_widget_position()
94
           return pygame.FRect(widget_position[0] - self.position[0], self.position
95
       [1] + widget_position[1], self.size[0], self.size[1])
96
       def calculate_widget_position(self):
97
           if self._vertical:
9.8
               return (0, -self._scroll_percentage * (self._widget.rect.height - self
       .size[1]))
100
           else:
101
               return (-self._scroll_percentage * (self._widget.rect.width - self.
       size[0]), 0)
       def calculate_relative_scrollbar_position(self):
103
           if self. vertical:
104
               vertical_offset = (self.size[1] - self.scrollbar_size[1]) * self.
       _scroll_percentage
               scrollbar_position = (0, vertical_offset)
107
               horizontal_offset = (self.size[0] - self.scrollbar_size[0]) * self.
108
       _scroll_percentage
               scrollbar_position = (horizontal_offset, 0)
```

```
return (scrollbar_position[0] / self.size[0], scrollbar_position[1] / self
111
       .size[1])
       def set_widget(self, new_widget):
            self._widget = new_widget
114
            self.set_image()
115
116
            self.set_geometry()
118
       def set_image(self):
            self.image = pygame.transform.scale(self._empty_surface, self.size)
119
           self.image.fill(theme['fillPrimary'])
120
121
            self._widget.set_image()
            self.image.blit(self._widget.image, self.calculate_widget_position())
124
            {\tt self.\_scrollbar.set\_relative\_position} \ (\, {\tt self.}
       {\tt calculate\_relative\_scrollbar\_position())} \ \ {\tt \#WRONG\ USING\ RELATIVE}
           self._scrollbar.set_relative_size((self.scrollbar_size[0] / self.size[1],
126
       self.scrollbar_size[1] / self.size[1]))
            self._scrollbar.set_image()
127
            relative_scrollbar_position = (self._scrollbar.rect.left - self.position
128
       [0], self._scrollbar.rect.top - self.position[1])
            self.image.blit(self._scrollbar.image, relative_scrollbar_position)
130
131
       def set_geometry(self):
           super().set_geometry()
132
133
            self._widget.set_geometry()
            self._scrollbar.set_geometry()
135
       def set_surface_size(self, new_surface_size):
136
137
            super().set_surface_size(new_surface_size)
            self._widget.set_surface_size(new_surface_size)
138
139
            self._scrollbar.set_surface_size(new_surface_size)
140
       def process_event(self, event):
141
            # WAITING FOR PYGAME-CE 2.5.3 TO RELEASE TO FIX SCROLL FLAGS
142
            # self.image.scroll(0, SCROLL_FACTOR)
143
           # self.image.scroll(0, -SCROLL_FACTOR)
144
145
            offset = self._scrollbar.process_event(event)
146
147
            if offset is not None:
148
                self.set_image()
149
150
                if abs(offset) > 0:
                    self._scroll_percentage = self.calculate_scroll_percentage(offset,
        scrollbar=True)
            if self.rect.collidepoint(pygame.mouse.get_pos()):
154
                if event.type == pygame.MOUSEBUTTONDOWN:
                    if event.button == 4:
                        self._scroll_percentage = self.calculate_scroll_percentage(-
157
       self.scroll_factor)
                        self.set_image()
158
                        return
                    elif event.button == 5:
                        if self._scroll_percentage == 100:
161
                        self._scroll_percentage = self.calculate_scroll_percentage(
164
       self.scroll_factor)
                        self.set image()
```

```
166
                        return
167
           widget_event = self._widget.process_event(event, scrolled_pos=self.
168
       calculate_widget_position())
          if widget_event is not None:
169
170
                self.set_image()
           return widget_event
171
 1 import pygame
 2 from data.widgets.bases.pressable import _Pressable
 3 from data.constants import WidgetState
 4 from data.utils.widget_helpers import create_slider_thumb
 {\scriptstyle 5} from data.managers.theme {\scriptstyle \mbox{import}} theme
 7 class _SliderThumb(_Pressable):
       def __init__(self, radius, border_colour=theme['borderPrimary'], fill_colour=
       theme['fillPrimary']):
           super() __init__(
 9
10
               event = None,
                down_func=self.down_func,
11
12
               up_func=self.up_func,
               hover_func=self.hover_func,
13
               prolonged=True,
14
15
               sfx = None
           )
16
           self._border_colour = border_colour
17
           self._radius = radius
18
           self._percent = None
19
20
           self.state = WidgetState.BASE
21
           self.initialise_new_colours(fill_colour)
22
23
      def get_position(self):
24
           return (self.rect.x, self.rect.y)
25
26
27
       def set_position(self, position):
           self.rect = self._thumb_surface.get_rect()
28
29
            self.rect.topleft = position
30
31
      def get_surface(self):
32
           return self._thumb_surface
33
       def set_surface(self, radius, border_width):
34
           self._thumb_surface = create_slider_thumb(radius, self._colours[self.state
35
       ], self._border_colour, border_width)
       def get_pressed(self):
37
            return self._pressed
38
39
       def down_func(self):
40
41
           self.state = WidgetState.PRESS
42
      def up_func(self):
43
            self.state = WidgetState.BASE
45
      def hover_func(self):
46
           self.state = WidgetState.HOVER
 1 import pygame
 2 from data.widgets.bases.widget import _Widget
 {\tt 3} from data.widgets.bases.pressable import _Pressable
 4 from data.constants import WidgetState
```

```
5 from data.utils.widget_helpers import create_switch
6 from data.components.custom_event import CustomEvent
7 from data.managers.theme import theme
9 class Switch(_Pressable, _Widget):
      def __init__(self, relative_height, event, fill_colour=theme['fillTertiary'],
10
      on_colour=theme['fillSecondary'], off_colour=theme['fillPrimary'], **kwargs):
          _Pressable.__init__(
12
               self.
13
               event = event,
               hover_func=self.hover_func,
14
               down_func=lambda: self.set_state_colour(WidgetState.PRESS),
               up_func=self.up_func,
16
          )
           _Widget.__init__(self, relative_size=(relative_height * 2, relative_height
18
      ), scale_mode='height',fill_colour=fill_colour, **kwargs)
19
20
           self._on_colour = on_colour
          self._off_colour = off_colour
21
          self._background_colour = None
23
          self._is_toggled = None
24
          self.set_toggle_state(False)
25
26
          self.initialise_new_colours(self._fill_colour)
27
          self.set_state_colour(WidgetState.BASE)
28
29
30
           self.set_image()
          self.set_geometry()
31
32
33
      def hover_func(self):
          self.set_state_colour(WidgetState.HOVER)
34
35
      def set_toggle_state(self, is_toggled):
36
          self._is_toggled = is_toggled
3.7
          if is_toggled:
38
              self._background_colour = self._on_colour
39
          else:
40
               self._background_colour = self._off_colour
41
42
           self.set_image()
43
44
      def up_func(self):
45
           if self.get_widget_state() == WidgetState.PRESS:
46
               toggle_state = not(self._is_toggled)
47
               self.set_toggle_state(toggle_state)
48
49
           self.set_state_colour(WidgetState.BASE)
50
51
      def draw_thumb(self):
52
          margin = self.size[1] * 0.1
53
           thumb_radius = (self.size[1] / 2) - margin
54
55
          if self._is_toggled:
5.6
               thumb_center = (self.size[0] - margin - thumb_radius, self.size[1] /
      2)
5.8
           else:
               thumb_center = (margin + thumb_radius, self.size[1] / 2)
59
6.0
          pygame.draw.circle(self.image, self._fill_colour, thumb_center,
61
      thumb_radius)
```

62

```
def set_image(self):
          self.image = create_switch(self.size, self._background_colour)
64
          self.draw_thumb()
65
      def process_event(self, event):
67
68
          data = super().process_event(event)
69
          if data:
7.0
               return CustomEvent(**vars(data), toggled=self._is_toggled)
7.1
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.constants import WidgetState
4 from data.utils.font_helpers import text_width_to_font_size,
      {\tt text\_height\_to\_font\_size} \ , \ {\tt height\_to\_font\_size}
5 from data.utils.widget_helpers import create_text_box
6 from data.assets import GRAPHICS
8 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)
10
11
          self._text = text
          self._fit_vertical = fit_vertical
12
          self._strength = strength
1.3
          self._box_colours = box_colours
15
16
          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]
18
              self._relative_font_size = text_width_to_font_size(self._text, self.
19
      _font, (self.size[0] - 2 * (self.margin + self.border_width))) / self.
      surface_size[1]
20
21
          if font_size:
              self._relative_font_size = font_size / self.surface_size[1]
22
23
          self._center = center
          self.rect = self._font.get_rect(self._text, size=self.font_size)
25
26
          self.rect.topleft = self.position
27
          self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
28
29
          self.set image()
30
31
          self.set_geometry()
33
      def resize_text(self):
          if self._fit_vertical:
34
35
              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:
36
37
               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.
38
      size[0] - (self.margin + self.border_width))) / self.surface_size[1]
39
40
               if new_font_size < ideal_font_size:</pre>
                   self._relative_font_size = new_font_size
41
42
```

```
43
                  self._relative_font_size = ideal_font_size
44
      def set_text(self, new_text):
45
          self._text = new_text
47
48
          self.resize_text()
          self.set_image()
49
5.0
51
      def set_image(self):
52
          if self._box_colours:
              self.image = create_text_box(self.size, self.border_width, self.
53
      _box_colours)
54
          else:
              text_surface = pygame.transform.scale(self._empty_surface, self.size)
5.5
              self.image = text_surface
56
5.7
58
              if self._fill_colour:
59
                  fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
                  pygame.draw.rect(self.image, self._fill_colour, fill_rect,
60
      border_radius=int(self.border_radius))
61
          self._font.strength = self._strength
62
          font_rect_size = self._font.get_rect(self._text, size=self.font_size).size
63
          if self._center:
64
              font_position = ((self.size[0] - font_rect_size[0]) / 2, (self.size[1]
65
       - font_rect_size[1]) / 2)
66
              font_position = (self.margin / 2, (self.size[1] - font_rect_size[1]) /
          self._font.render_to(self.image, font_position, self._text, fgcolor=self.
68
      _text_colour, size=self.font_size)
69
70
          if self._box_colours is None and self.border_width:
              fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
71
              (self.border_width), border_radius=int(self.border_radius))
73
      def process_event(self, event):
7.4
          pass
1 from data.widgets.bases.pressable import _Pressable
2 from data.widgets.bases.box import _Box
3 from data.widgets.text import Text
4 from data.constants import WidgetState, BLUE_BUTTON_COLOURS
6 class TextButton(_Box, _Pressable, Text):
7     def __init__(self, event, **kwargs):
           _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
          _Pressable.__init__(
              self.
10
11
              event = event,
              hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
12
1.3
              down_func=lambda: self.set_state_colour(WidgetState.PRESS),
              up_func=lambda: self.set_state_colour(WidgetState.BASE),
15
          Text.__init__(self, box_colours=BLUE_BUTTON_COLOURS[WidgetState.BASE], **
16
      kwargs)
17
          self.initialise_new_colours(self._fill_colour)
          self.set_state_colour(WidgetState.BASE)
19
1 import pyperclip
```

```
2 import pygame
3 from data.constants import WidgetState, CursorMode, INPUT_COLOURS
4 from data.components.custom_event import CustomEvent
5 from data.widgets.bases.pressable import _Pressable
6 from data.managers.logs import initialise_logger
7 from data.managers.animation import animation
8 from data.widgets.bases.box import _Box
9 from data.managers.cursor import cursor
10 from data.managers.theme import theme
11 from data.widgets.text import Text
13 logger = initialise_logger(__name__)
14
15 class TextInput(_Box, _Pressable, Text):
      def __init__(self, event, blinking_interval=530, validator=(lambda x: True),
      default='', placeholder='PLACEHOLDER TEXT', placeholder_colour=(200, 200, 200)
      , cursor_colour=theme['textSecondary'], **kwargs):
          self._cursor_index = None
          \mbox{\tt\#} Multiple inheritance used here, adding the functionality of pressing,
18
      and custom box colours, to the text widget
          _Box.__init__(self, box_colours=INPUT_COLOURS)
19
2.0
           _Pressable.__init__(
              self,
21
               event = None.
22
              hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
23
               down_func=lambda: self.set_state_colour(WidgetState.PRESS),
24
25
              up_func=lambda: self.set_state_colour(WidgetState.BASE),
26
               sfx = None
27
          Text.__init__(self, text="", center=False, box_colours=INPUT_COLOURS[
28
      WidgetState.BASE], **kwargs)
29
30
          self.initialise_new_colours(self._fill_colour)
          self.set_state_colour(WidgetState.BASE)
31
32
          pygame.key.set_repeat(500, 50)
33
34
          self._blinking_fps = 1000 / blinking_interval
3.5
          self._cursor_colour = cursor_colour
          self._cursor_colour_copy = cursor_colour
37
          self._placeholder_colour = placeholder_colour
38
          self._text_colour_copy = self._text_colour
39
40
41
          self._placeholder_text = placeholder
          self._is_placeholder = None
42
          if default:
43
               self._text = default
              self.is_placeholder = False
45
46
          else:
47
              self._text = self._placeholder_text
48
               self.is_placeholder = True
49
          self._event = event
50
          self._validator = validator
5.1
          self._blinking_cooldown = 0
53
          self._empty_cursor = pygame.Surface((0, 0), pygame.SRCALPHA)
5.4
56
          self.resize_text()
5.7
          self.set_image()
          self.set_geometry()
58
```

59

```
60
       @property
       # Encapsulated getter method
61
       def is_placeholder(self):
62
           return self._is_placeholder
64
6.5
       @is_placeholder.setter
       # Encapsulated setter method, used to replace text colour if placeholder text
66
       is shown
       def is_placeholder(self, is_true):
67
           self._is_placeholder = is_true
68
6.9
70
           if is_true:
               self._text_colour = self._placeholder_colour
71
           else:
72
               self._text_colour = self._text_colour_copy
73
7.4
75
       @property
76
       def cursor_size(self):
           cursor\_height = (self.size[1] - self.border\_width * 2) * 0.75
7.7
78
           return (cursor_height * 0.1, cursor_height)
79
80
       @property
       def cursor_position(self):
81
           current_width = (self.margin / 2)
82
           for index, metrics in enumerate(self._font.get_metrics(self._text, size=
83
       self.font_size)):
84
               if index == self._cursor_index:
                    return (current_width - self.cursor_size[0], (self.size[1] - self.
       cursor_size[1]) / 2)
86
               glyph_width = metrics[4]
               current_width += glyph_width
88
           return (current_width - self.cursor_size[0], (self.size[1] - self.
89
       cursor_size[1]) / 2)
90
91
       @property
       def text(self):
92
           if self.is_placeholder:
93
               return '
95
           return self._text
96
       def relative_x_to_cursor_index(self, relative_x):
98
99
           Calculates cursor index using mouse position relative to the widget
100
       position.
102
           Args:
              relative_x (int): Horizontal distance of the mouse from the left side
103
       of the widget.
104
105
           Returns:
           int: Cursor index.
106
107
           current_width = 0
109
           for index, metrics in enumerate(self._font.get_metrics(self._text, size=
      self.font_size)):
               glyph_width = metrics[4]
112
               if current_width >= relative_x:
113
                   return index
114
```

```
current_width += glyph_width
116
           return len(self._text)
119
       def set_cursor_index(self, mouse_pos):
120
121
           Sets cursor index based on mouse position.
123
124
           mouse_pos (list[int, int]): Mouse position relative to window screen.
126
           if mouse_pos is None:
127
               self._cursor_index = mouse_pos
128
129
130
           relative_x = mouse_pos[0] - (self.margin / 2) - self.rect.left
131
132
            relative_x = max(0, relative_x)
           self._cursor_index = self.relative_x_to_cursor_index(relative_x)
133
134
       def focus_input(self, mouse_pos):
135
136
           Draws cursor and sets cursor index when user clicks on widget.
137
138
139
           Args:
           mouse_pos (list[int, int]): Mouse position relative to window screen.
140
141
142
           if self.is_placeholder:
               self._text = ''
143
               self.is_placeholder = False
144
           self.set_cursor_index(mouse_pos)
146
147
            self.set_image()
           cursor.set_mode(CursorMode.IBEAM)
148
149
150
       def unfocus_input(self):
151
           Removes cursor when user unselects widget.
152
           0.000
153
           if self._text == '':
154
                self._text = self._placeholder_text
155
                self.is_placeholder = True
156
               self.resize_text()
157
158
           self.set_cursor_index(None)
159
160
           self.set_image()
            cursor.set_mode(CursorMode.ARROW)
161
162
163
       def set_text(self, new_text):
164
           Called by a state object to change the widget text externally.
165
166
167
           Args:
               new_text (str): New text to display.
168
           Returns:
               CustomEvent: Object containing the new text to alert state of a text
171
       update.
172
173
           super().set_text(new_text)
           return CustomEvent(**vars(self._event), text=self.text)
174
175
```

```
176
       def process_event(self, event):
            Processes Pygame events.
178
180
            Args:
                event (pygame.Event): Event to process.
181
182
            Returns:
183
                CustomEvent: Object containing the new text to alert state of a text
184
       update.
185
            previous_state = self.get_widget_state()
            super().process_event(event)
187
            current_state = self.get_widget_state()
188
189
            match event.type:
190
                {\tt case pygame.MOUSEMOTION:}
191
192
                     if self._cursor_index is None:
                         return
194
                     # If mouse is hovering over widget, turn mouse cursor into an I-
       beam
                     if self.rect.collidepoint(event.pos):
196
                         if cursor.get_mode() != CursorMode.IBEAM:
197
198
                              cursor.set_mode(CursorMode.IBEAM)
199
                         if cursor.get_mode() == CursorMode.IBEAM:
200
201
                              cursor.set_mode(CursorMode.ARROW)
202
                     return
203
204
                {\tt case \ pygame.MOUSEBUTTONUP:}
205
206
                     # When user selects widget
                     if previous_state == WidgetState.PRESS:
207
                         self.focus_input(event.pos)
208
                     # When user unselects widget
209
                     if current_state == WidgetState.BASE and self._cursor_index is not
210
         None:
                         self.unfocus_input()
                         return CustomEvent(**vars(self._event), text=self.text)
212
213
                case pygame.KEYDOWN:
214
                    if self._cursor_index is None:
215
216
                         return
217
                     \mbox{\tt\#} Handling Ctrl-C and Ctrl-V shortcuts
218
                     if event.mod & (pygame.KMOD_CTRL):
219
                         if event.key == pygame.K_c:
220
                             pyperclip.copy(self.text)
221
222
                              logger.info(f'COPIED {self.text}')
224
                         elif event.key == pygame.K_v:
                             pasted_text = pyperclip.paste()
pasted_text = ''.join(char for char in pasted_text if 32
       <= ord(char) <= 127)
                             self._text = self._text[:self._cursor_index] + pasted_text
227
        + self._text[self._cursor_index:]
                              self._cursor_index += len(pasted_text)
228
                         elif event.key == pygame.K_BACKSPACE or event.key == pygame.
230
       K_DELETE:
                              self._text = ''
231
```

```
self._cursor_index = 0
232
233
                         self.resize_text()
                         self.set_image()
235
                         self.set_geometry()
237
238
239
240
                    match event.key:
                         case pygame K_BACKSPACE:
241
                             if self._cursor_index > 0:
242
243
                                 self._text = self._text[:self._cursor_index - 1] +
       self._text[self._cursor_index:]
                             self._cursor_index = max(0, self._cursor_index - 1)
244
245
                         {\tt case pygame.K\_RIGHT:}
246
                             self._cursor_index = min(len(self._text), self.
247
       _cursor_index + 1)
248
                         case pygame.K_LEFT:
249
                             self._cursor_index = max(0, self._cursor_index - 1)
250
251
                         case pygame.K_ESCAPE:
252
                             self.unfocus_input()
                             return CustomEvent(**vars(self._event), text=self.text)
254
255
                         {\tt case pygame.K\_RETURN:}
257
                             self.unfocus_input()
                             return CustomEvent(**vars(self._event), text=self.text)
258
260
                         case _:
                            if not event.unicode:
261
262
                                 return
263
                             potential_text = self._text[:self._cursor_index] + event.
264
       unicode + self._text[self._cursor_index:]
265
                             # Validator lambda function used to check if inputted text
266
        is valid before displaying
                             # e.g. Time control input has a validator function
267
       checking if text represents a float
                             if self._validator(potential_text) is False:
268
269
                                 return
270
                             self._text = potential_text
271
                             self._cursor_index += 1
273
                    self._blinking_cooldown += 1
274
                    \verb|animation.set_timer| (500, \verb|lambda|: self.subtract_blinking_cooldown|
275
       (1))
277
                    self.resize_text()
                    self.set_image()
278
                    self.set_geometry()
280
       def subtract_blinking_cooldown(self, cooldown):
281
282
            Subtracts blinking cooldown after certain timeframe. When
283
       \verb|blinking_cooldown| is 1, cursor is able to be drawn.
284
            Args:
285
                cooldown (float): Duration before cursor can no longer be drawn.
286
```

```
287
           self._blinking_cooldown = self._blinking_cooldown - cooldown
288
289
       def set_image(self):
291
292
           Draws text input widget to image.
293
           super().set_image()
294
295
296
           if self._cursor_index is not None:
               scaled_cursor = pygame.transform.scale(self._empty_cursor, self.
297
       cursor_size)
               scaled_cursor.fill(self._cursor_colour)
298
299
                self.image.blit(scaled_cursor, self.cursor_position)
300
       def update(self):
301
302
303
           Overrides based update method, to handle cursor blinking.
304
           super().update()
305
           # Calculate if cursor should be shown or not
306
           cursor_frame = animation.calculate_frame_index(0, 2, self._blinking_fps)
307
           if cursor_frame == 1 and self._blinking_cooldown == 0:
               self._cursor_colour = (0, 0, 0, 0)
309
310
           else:
311
               self._cursor_colour = self._cursor_colour_copy
312
           self.set_image()
 1 import pygame
 2 from data.constants import WidgetState, Colour, BLUE_BUTTON_COLOURS,
       RED_BUTTON_COLOURS
 3 from data.components.custom_event import CustomEvent
 4 from data.managers.animation import animation
 5 from data.widgets.text import Text
 7 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
11
           self._active_colour = active_colour
 12
           self._active = False
13
           self._timer_running = False
14
 15
           self._event = event
16
17
           super().__init__(text=self.format_to_text(), fit_vertical=False,
       box_colours=box_colours, **kwargs)
18
       def set_active(self, is_active):
19
           if self._active == is_active:
20
21
               return
           if is_active and self._timer_running is False:
23
                self._timer_running = True
24
                animation.set_timer(1000, self.decrement_second)
26
27
           self._active = is_active
28
29
       def set_time(self, milliseconds):
30
           self._current_ms = milliseconds
           self._text = self.format_to_text()
31
```

```
self.set_image()
32
          self.set_geometry()
33
3.4
      def get_time(self):
35
          return self._current_ms / (1000 * 60)
36
3.7
38
      def decrement_second(self):
          if self._active:
39
               self.set_time(self._current_ms - 1000)
40
41
42
              if self._current_ms <= 0:</pre>
43
                   self._active = False
                   self._timer_running = False
44
45
                   self.set_time(0)
                   pygame.event.post(pygame.event.Event(pygame.MOUSEMOTION, pos=
      pygame.mouse.get_pos())) # RANDOM EVENT TO TRIGGER process_event
47
               else:
48
                   animation.set_timer(1000, self.decrement_second)
49
          else:
               self._timer_running = False
50
51
      def format_to_text(self):
52
          raw_seconds = self._current_ms / 1000
53
          minutes, seconds = divmod(raw_seconds, 60)
54
          return f'{str(int(minutes)).zfill(2)}:{str(int(seconds)).zfill(2)}'
55
56
5.7
      def process_event(self, event):
58
           if self._current_ms <= 0:</pre>
              return CustomEvent(**vars(self._event), active_colour=self.
59
      _active_colour)
1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.slider_thumb import _SliderThumb
4 from data.components.custom_event import CustomEvent
5 from data.constants import SettingsEventType
6 from data.constants import WidgetState
7 from data.utils.widget_helpers import create_slider
8 from data.utils.asset_helpers import scale_and_cache
9 from data.managers.theme import theme
10
11 class VolumeSlider(_Widget):
      def __init__(self, relative_length, default_volume, volume_type, thumb_colour=
12
      theme['fillSecondary'], **kwargs):
          super().__init__(relative_size=(relative_length, relative_length * 0.2),
      **kwargs)
14
           self._volume_type = volume_type
          self._selected_percent = default_volume
16
          self._last_mouse_x = None
17
18
          self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
19
      _border_colour, fill_colour=thumb_colour)
          self._gradient_surface = create_slider(self.calculate_slider_size(), self.
20
      _fill_colour, self.border_width, self._border_colour)
          self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
22
23
24
      @property
      def position(self):
2.5
           '''Minus so easier to position slider by starting from the left edge of
26
      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])
      {\tt def} \ \ {\tt calculate\_slider\_position(self)}:
          return (self size[1] / 2, self size[1] / 4)
30
31
      def calculate_slider_size(self):
32
          return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
33
34
35
      def calculate_selected_percent(self, mouse_pos):
          if self._last_mouse_x is None:
36
37
              return
38
           x_change = (mouse_pos[0] - self._last_mouse_x) / (self.
39
      calculate_slider_size()[0] - 2 * self.border_width)
          return max(0, min(self._selected_percent + x_change, 1))
40
41
      def calculate_thumb_position(self):
42
          gradient_size = self.calculate_slider_size()
43
          x = gradient_size[0] * self._selected_percent
          y = 0
45
46
          return (x, y)
47
48
      def relative_to_global_position(self, position):
49
          relative_x , relative_y = position
50
          return (relative_x + self.position[0], relative_y + self.position[1])
51
      def set_image(self):
53
          gradient_scaled = scale_and_cache(self._gradient_surface, self.
5.4
      calculate_slider_size())
          gradient_position = self.calculate_slider_position()
5.5
56
           self.image = pygame.transform.scale(self._empty_surface, (self.size))
          {\tt self.image.blit(gradient\_scaled, gradient\_position)}
58
           thumb_position = self.calculate_thumb_position()
60
          self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
61
      border width)
          self._thumb.set_position(self.relative_to_global_position((thumb_position
62
      [0], thumb_position[1])))
           thumb_surface = self._thumb.get_surface()
64
65
           self.image.blit(thumb_surface, thumb_position)
66
6.7
      def set_volume(self, volume):
           self._selected_percent = volume
           self.set_image()
69
70
71
      def process_event(self, event):
           if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
72
      MOUSEBUTTONUP]:
73
7.4
          before_state = self._thumb.state
          self._thumb.process_event(event)
76
7.7
          after_state = self._thumb.state
          if before_state != after_state:
7.9
80
               self.set_image()
81
              if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:
82
```

```
self._last_mouse_x = None
                      return CustomEvent(SettingsEventType.VOLUME_SLIDER_CLICK, volume=
84
       round(self._selected_percent, 3), volume_type=self._volume_type)
            if self._thumb.state == WidgetState.PRESS:
86
                 selected_percent = self.calculate_selected_percent(event.pos)
87
                 self._last_mouse_x = event.pos[0]
89
                 if selected_percent:
90
91
                      self._selected_percent = selected_percent
                      self.set_image()
92
93
                      return CustomEvent(SettingsEventType.VOLUME_SLIDER_SLIDE)
1 from data.widgets.bases.widget import _Widget
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.bases.circular import _Circular
4 from data.widgets.bases.box import _Box
5 from data.widgets.colour_display import _ColourDisplay
6 from data.widgets.colour_square import _ColourSquare
7 from data.widgets.colour_slider import _ColourSlider
{\tt 8} \  \  \, \textbf{from} \  \  \, \textbf{data.widgets.slider\_thumb} \  \  \, \textbf{import} \  \  \, \textbf{\_SliderThumb}
9 from data.widgets.scrollbar import _Scrollbar
11 from data.widgets.board_thumbnail_button import BoardThumbnailButton
12 from data.widgets.multiple_icon_button import MultipleIconButton
13 from data.widgets.reactive_icon_button import ReactiveIconButton
{\tt 14} \  \  \, \textbf{from} \  \  \, \textbf{data.widgets.board\_thumbnail} \  \  \, \textbf{import} \  \  \, \textbf{BoardThumbnail}
15 from data.widgets.reactive_button import ReactiveButton
16 from data.widgets.volume_slider import VolumeSlider
17 from data.widgets.colour_picker import ColourPicker
18 from data.widgets.colour_button import ColourButton
19 from data.widgets.browser_strip import BrowserStrip
20 from data.widgets.piece_display import PieceDisplay
21 from data.widgets.browser_item import BrowserItem
22 from data.widgets.text_button import TextButton
23 from data.widgets.icon_button import IconButton
{\tt 24} \  \  \, \textbf{from} \  \  \, \textbf{data.widgets.scroll\_area} \  \  \, \textbf{import} \  \  \, \textbf{ScrollArea}
25 from data.widgets.chessboard import Chessboard
26 from data.widgets.text_input import TextInput
27 from data.widgets.rectangle import Rectangle
28 from data.widgets.move_list import MoveList
29 from data.widgets.dropdown import Dropdown
30 from data.widgets.carousel import Carousel
31 from data.widgets.switch import Switch
32 from data.widgets.timer import Timer
33 from data.widgets.text import Text
34 from data.widgets.icon import Icon
36 __all__ = ['Text', 'TextButton', 'ColourPicker', 'ColourButton', 'Switch', '
       Dropdown', 'IconButton', 'Icon', 'VolumeSlider', 'TextInput', '
MultipleIconButton', 'Carousel', 'Timer', 'Rectangle', 'Chessboard', '
ScrollArea', 'MoveList', 'BoardThumbnail', 'BrowserStrip', 'BrowserItem', '
PieceDisplay', 'BoardThumbnailButton', 'ReactiveButton', 'ReactiveIconButton']
 1 from data.constants import WidgetState
3 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 from data.components.circular_linked_list import CircularLinkedList
3 class _Circular:
      def __init__(self, items_dict, **kwargs):
           # The key, value pairs are stored within a dictionary, while the keys to
      access them are stored within circular linked list.
          self._items_dict = items_dict
          self._keys_list = CircularLinkedList(list(items_dict.keys()))
      @property
      def current_key(self):
10
11
          Gets the current head node of the linked list, and returns a key stored as
       the node data.
          Returns:
          Data of linked list head.
14
15
          return self._keys_list.get_head().data
17
18
     @property
      def current_item(self):
19
2.0
          Gets the value in self._items_dict with the key being self.current_key.
21
22
23
          Returns:
          Value stored with key being current head of linked list.
24
25
          return self._items_dict[self.current_key]
26
27
      def set_next_item(self):
28
29
          Sets the next item in as the current item.
30
3.1
32
          self._keys_list.shift_head()
33
      def set_previous_item(self):
34
          Sets the previous item as the current item.
36
37
           self._keys_list.unshift_head()
38
3.9
40
      def set_to_key(self, key):
41
42
          Sets the current item to the specified key.
43
44
          Args:
45
              key: The key to set as the current item.
46
47
          Raises:
              ValueError: If no nodes within the circular linked list contains the
48
      key as its data.
40
           if self._keys_list.data_in_list(key) is False:
              raise ValueError('(_Circular.set_to_key) Key not found:', key)
51
52
          for _ in range(len(self._items_dict)):
53
              if self.current_key == key:
54
55
                   self.set_image()
                   self.set_geometry()
56
```

```
57
                   return
58
               self.set_next_item()
59
1 import pygame
2 from data.constants import WidgetState
3 from data.managers.audio import audio
4 from data.assets import SFX
6 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
9
10
           self._hover_func = hover_func
           self._pressed = False
11
           self._prolonged = prolonged
12
           self._sfx = sfx
14
           self._event = event
15
           self._widget_state = WidgetState.BASE
17
18
           self._colours = {}
19
2.0
21
      def set_state_colour(self, state):
           self._fill_colour = self._colours[state]
22
23
24
           self.set_image()
25
     def initialise_new_colours(self, colour):
26
27
           r, g, b, a = pygame.Color(colour).rgba
28
29
           self._colours = {
               WidgetState.BASE: pygame.Color(r, g, b, a),
30
               WidgetState. \texttt{HOVER: pygame.Color(min(r + 25, 255), min(g + 25, 255),}
3.1
      min(b + 25, 255), a),
               WidgetState.PRESS: pygame.Color(min(r + 50, 255), min(g + 50, 255),
32
      min(b + 50, 255), a)
34
35
      def get_widget_state(self):
           return self._widget_state
36
3.7
38
      def process_event(self, event):
           match event.type:
39
               {\tt case \ pygame.MOUSEBUTTONDOWN:}
40
                   if self.rect.collidepoint(event.pos):
41
                       self._down_func()
42
43
                        self._widget_state = WidgetState.PRESS
44
               {\tt case \ pygame.MOUSEBUTTONUP:}
45
46
                   if self.rect.collidepoint(event.pos):
                       if self._widget_state == WidgetState.PRESS:
47
                            if self._sfx:
48
                                audio.play_sfx(self._sfx)
50
51
                            self._up_func()
                            self._widget_state = WidgetState.HOVER
52
53
                            return self._event
54
                        elif self._widget_state == WidgetState.BASE:
55
```

```
self._hover_func()
57
                    elif self._prolonged and self._widget_state == WidgetState.PRESS:
58
                        if self._sfx:
                            audio.play_sfx(self._sfx)
60
6.1
                        self._up_func()
                        self._widget_state = WidgetState.BASE
62
                        return self._event
63
64
                case pygame.MOUSEMOTION:
65
                    if self.rect.collidepoint(event.pos):
66
67
                        if self._widget_state == WidgetState.PRESS:
68
                             return
                        elif self._widget_state == WidgetState.BASE:
69
                             self._hover_func()
                             self._widget_state = WidgetState.HOVER
7.1
                         elif self._widget_state == WidgetState.HOVER:
73
                             self._hover_func()
                    else:
7.4
                        if self._prolonged is False:
75
                            if self._widget_state in [WidgetState.PRESS, WidgetState.
      HOVER1:
                                 self._widget_state = WidgetState.BASE
                                 self._up_func()
7.8
                             elif self._widget_state == WidgetState.BASE:
80
                                 return
                         elif self._prolonged is True:
81
                              \  \  \, \textbf{if} \  \  \, \textbf{self.\_widget\_state} \  \  \, \textbf{in} \  \, [\,\textbf{WidgetState.PRESS}\,\,,\,\,\, \textbf{WidgetState}\,.
       BASE]:
83
                                 return
84
                             else:
                                 self._widget_state = WidgetState.BASE
8.5
86
                                 self._up_func()
1 import pygame
_2 from data.constants <code>import</code> <code>SCREEN_SIZE</code>
3 from data.managers.theme import theme
4 from data.assets import DEFAULT_FONT
6 DEFAULT_SURFACE_SIZE = SCREEN_SIZE
7 REQUIRED_KWARGS = ['relative_position', 'relative_size']
9 class _Widget(pygame.sprite.Sprite):
      def __init__(self, **kwargs):
1.0
11
           Every widget has the following attributes:
12
13
           surface (pygame.Surface): The surface the widget is drawn on.
           raw_surface_size (tuple[int, int]): The initial size of the window screen,
15
       remains constant.
          parent (_Widget, optional): The parent widget position and size is
16
       relative to.
17
           Relative to current surface:
18
           relative_position (tuple[float, float]): The position of the widget
19
       relative to its surface.
          relative_size (tuple[float, float]): The scale of the widget relative to
20
      its surface.
21
           Remains constant, relative to initial screen size:
22
           relative_font_size (float, optional): The relative font size of the widget
23
```

```
relative_margin (float): The relative margin of the widget.
           relative_border_width (float): The relative border width of the widget.
           relative_border_radius (float): The relative border radius of the widget.
26
           anchor_x (str): The horizontal anchor direction ('left', 'right', 'center
28
      ').
           anchor_y (str): The vertical anchor direction ('top', 'bottom', 'center').
           fixed_position (tuple[int, int], optional): The fixed position of the
3.0
      widget in pixels.
          border_colour (pygame.Color): The border color of the widget.
31
           text_colour (pygame.Color): The text color of the widget.
fill_colour (pygame.Color): The fill color of the widget.
32
33
           font (pygame.freetype.Font): The font used for the widget.
34
3.5
36
           super().__init__()
37
           for required_kwarg in REQUIRED_KWARGS:
38
39
               if required_kwarg not in kwargs:
                   raise KeyError(f'(_Widget.__init__) Required keyword "{
40
      required_kwarg}" not in base kwargs')
41
           self._surface = None # Set in WidgetGroup, as needs to be reassigned every
42
       frame
          self._raw_surface_size = DEFAULT_SURFACE_SIZE
43
44
45
           self._parent = kwargs.get('parent')
46
47
           self._relative_font_size = None # Set in subclass
48
           self._relative_position = kwargs.get('relative_position')
49
50
           self._relative_margin = theme['margin'] / self._raw_surface_size[1]
           self._relative_border_width = theme['borderWidth'] / self.
5.1
      _raw_surface_size[1]
           self._relative_border_radius = theme['borderRadius'] / self.
      _raw_surface_size[1]
53
           self._border_colour = pygame.Color(theme['borderPrimary'])
54
           self._text_colour = pygame.Color(theme['textPrimary'])
5.5
           self._fill_colour = pygame.Color(theme['fillPrimary'])
           self._font = DEFAULT_FONT
57
5.8
           self._anchor_x = kwargs.get('anchor_x') or 'left'
59
           self._anchor_y = kwargs.get('anchor_y') or 'top'
60
61
           self._fixed_position = kwargs.get('fixed_position')
           scale_mode = kwargs.get('scale_mode') or 'both'
62
63
           if kwargs.get('relative_size'):
               match scale_mode:
65
66
                   case 'height':
67
                       self._relative_size = kwargs.get('relative_size')
                   case 'width':
68
                       self._relative_size = ((kwargs.get('relative_size')[0] * self.
69
      surface_size[0]) / self.surface_size[1], (kwargs.get('relative_size')[1] *
      self.surface_size[0]) / self.surface_size[1])
                   case 'both':
                       self._relative_size = ((kwargs.get('relative_size')[0] * self.
71
      surface_size[0]) / self.surface_size[1], kwargs.get('relative_size')[1])
                   case _:
                       raise ValueError('(_Widget.__init__) Unknown scale mode:',
7.3
      scale_mode)
          else:
               self._relative_size = (1, 1)
7.5
```

```
if 'margin' in kwargs:
7.7
                self._relative_margin = kwargs.get('margin') / self._raw_surface_size
7.8
       [1]
79
                if (self._relative_margin * 2) > min(self._relative_size[0], self.
80
       _relative_size[1]):
                   raise ValueError('(_Widget.__init__) Margin larger than specified
8.1
       size!')
82
           if 'border_width' in kwargs:
83
                self._relative_border_width = kwargs.get('border_width') / self.
       _raw_surface_size[1]
8.5
            if 'border_radius' in kwargs:
86
                self._relative_border_radius = kwargs.get('border_radius') / self.
87
       _raw_surface_size[1]
88
            if 'border_colour' in kwargs:
89
                self._border_colour = pygame.Color(kwargs.get('border_colour'))
90
91
           if 'fill_colour' in kwargs:
92
                self._fill_colour = pygame.Color(kwargs.get('fill_colour'))
93
94
           if 'text_colour' in kwargs:
95
               self._text_colour = pygame.Color(kwargs.get('text_colour'))
96
97
            if 'font' in kwargs:
98
                self._font = kwargs.get('font')
99
100
101
       @property
       def surface_size(self):
102
103
           Gets the size of the surface widget is drawn on.
104
           Can be either the window size, or another widget size if assigned to a
       parent.
           Returns:
107
               tuple[int, int]: The size of the surface.
108
109
           if self._parent:
111
               return self._parent.size
           else:
113
                return self._raw_surface_size
114
115
       @property
       def position(self):
           Gets the position of the widget.
118
           Accounts for fixed position attribute, where widget is positioned in
       pixels regardless of screen size.
           Acounts for anchor direction, where position attribute is calculated
120
       relative to one side of the screen.
           Returns:
               tuple[int, int]: The position of the widget.
123
124
           x, y = None, None
            \  \  \, \textbf{if} \  \  \, \textbf{self.\_fixed\_position} : \\
126
127
               x, y = self._fixed_position
            if x is None:
128
                x = self._relative_position[0] * self.surface_size[0]
129
```

```
130
           if y is None:
                y = self._relative_position[1] * self.surface_size[1]
131
132
           if self._anchor_x == 'left':
134
                x = x
            elif self._anchor_x == 'right':
135
               x = self.surface_size[0] - x - self.size[0]
136
            elif self._anchor_x == 'center':
137
                x = (self.surface_size[0] / 2 - self.size[0] / 2) + x
138
139
           if self._anchor_y == 'top':
140
141
                у = у
           elif self._anchor_y == 'bottom':
142
                y = self.surface_size[1] - y - self.size[1]
143
           elif self._anchor_y == 'center':
144
               y = (self.surface_size[1] / 2 - self.size[1] / 2) + y
145
146
147
            # Position widget relative to parent, if exists.
           if self._parent:
148
                return (x + self._parent.position[0], y + self._parent.position[1])
149
150
            else:
               return (x, y)
151
152
       @property
153
154
       def size(self):
           return (self._relative_size[0] * self.surface_size[1], self._relative_size
155
       [1] * self.surface_size[1])
       @property
157
       def margin(self):
158
           return self._relative_margin * self._raw_surface_size[1]
160
161
       @property
       def border_width(self):
162
           return self._relative_border_width * self._raw_surface_size[1]
163
164
       @property
       def border_radius(self):
166
           return self._relative_border_radius * self._raw_surface_size[1]
167
168
169
       @property
       def font_size(self):
170
           return self._relative_font_size * self.surface_size[1]
172
       def set_image(self):
173
174
175
            Abstract method to draw widget.
176
           raise NotImplementedError
177
178
       def set_geometry(self):
179
180
           Sets the position and size of the widget.
181
182
           self.rect = self.image.get_rect()
184
           if self._anchor_x == 'left':
185
               if self._anchor_y == 'top':
186
                self.rect.topleft = self.position
elif self._anchor_y == 'bottom':
187
188
                    self.rect.topleft = self.position
189
                elif self._anchor_y == 'center':
190
```

```
self.rect.topleft = self.position
191
           elif self _anchor_x == 'right':
192
               if self._anchor_y == 'top':
193
                    self.rect.topleft = self.position
194
                elif self._anchor_y == 'bottom':
195
                    self.rect.topleft = self.position
196
                elif self._anchor_y == 'center':
197
                    self.rect.topleft = self.position
198
           elif self._anchor_x == 'center':
199
               if self._anchor_y == 'top':
200
                    self.rect.topleft = self.position
201
                elif self._anchor_y == 'bottom':
202
                self.rect.topleft = self.position
elif self._anchor_y == 'center':
203
204
205
                    self.rect.topleft = self.position
206
      def set_surface_size(self, new_surface_size):
207
208
           Sets the new size of the surface widget is drawn on.
209
210
           Args:
211
            new_surface_size (tuple[int, int]): The new size of the surface.
212
213
            self._raw_surface_size = new_surface_size
214
215
      def process_event(self, event):
216
217
218
            Abstract method to handle events.
219
           Args:
220
           event (pygame.Event): The event to process.
221
222
           raise NotImplementedError
223
```