



```

39     BACKGROUND_NONE = auto()
40
41     _BLUR = auto()
42     _HIGHLIGHT_BRIGHTNESS = auto()
43     _HIGHLIGHT_COLOUR = auto()
44     _CALIBRATE = auto()
45     _LIGHTMAP = auto()
46     _SHADOWMAP = auto()
47     _OCCLUSION = auto()
48     _BLEND = auto()
49     _CROP = auto()
50
51     SHADER_MAP = {
52         'default': [
53             ShaderType.BLOOM
54         ],
55         'retro': [
56             ShaderType.CRT
57         ],
58         'really_retro': [
59             ShaderType.CRT,
60             ShaderType.GRAYSCALE
61         ],
62     }
63
64     class TranspositionFlag(StrEnum):
65         LOWER = auto()
66         EXACT = auto()
67         UPPER = auto()
68
69     class Miscellaneous(StrEnum):
70         PLACEHOLDER = auto()
71         DRAW = auto()
72
73     class WidgetState(StrEnum):
74         BASE = auto()
75         HOVER = auto()
76         PRESS = auto()
77
78     BLUE_BUTTON_COLOURS = {
79         WidgetState.BASE: ['0x1c2638', '0x23495d', '0x39707a', '0x95e0cc'],
80         WidgetState.HOVER: ['0xdaf2e9', '0x23495d', '0x39707a', '0x95e0cc'],
81         WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
82     }
83
84     INPUT_COLOURS = {
85         WidgetState.BASE: ['0x1c2638', '0x39707a', '0x23495d', '0x95e0cc'],
86         WidgetState.HOVER: ['0xdaf2e9', '0x39707a', '0x23495d', '0x95e0cc'],
87         WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0x1c2638', '0x39707a']
88     }
89
90     RED_BUTTON_COLOURS = {
91         WidgetState.BASE: ['0x000000', '0x1c2638', '0x9b222b', '0xf14e52'],
92         WidgetState.HOVER: ['0xdaf2e9', '0x1c2638', '0x9b222b', '0xf14e52'],
93         WidgetState.PRESS: ['0xdaf2e9', '0x23495d', '0xf14e52', '0x95e0cc']
94     }
95
96     LOCKED_RED_BUTTON_COLOURS = {
97         WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
98         WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
99         WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0xf14e52']
100 }

```

```

101
102 LOCKED_BLUE_BUTTON_COLOURS = {
103     WidgetState.BASE: ['0x000000', '0x000000', '0x1c2638', '0x23495d'],
104     WidgetState.HOVER: ['0xdaf2e9', '0x000000', '0x1c2638', '0x23495d'],
105     WidgetState.PRESS: ['0xdaf2e9', '0x1c2638', '0x23495d', '0x39707a']
106 }
107
108 class StatusText(StrEnum):
109     PLAYER_MOVE = auto()
110     CPU_MOVE = auto()
111     WIN = auto()
112     DRAW = auto()
113
114 class EditorEventType(StrEnum):
115     MENU_CLICK = auto()
116     PICK_PIECE_CLICK = auto()
117     ROTATE_PIECE_CLICK = auto()
118     COPY_CLICK = auto()
119     EMPTY_CLICK = auto()
120     RESET_CLICK = auto()
121     BLUE_START_CLICK = auto()
122     RED_START_CLICK = auto()
123     START_CLICK = auto()
124     CONFIG_CLICK = auto()
125     ERASE_CLICK = auto()
126     MOVE_CLICK = auto()
127     HELP_CLICK = auto()
128
129 class ReviewEventType(StrEnum):
130     MENU_CLICK = auto()
131     PREVIOUS_CLICK = auto()
132     NEXT_CLICK = auto()
133     HELP_CLICK = auto()
134
135 class BrowserEventType(StrEnum):
136     MENU_CLICK = auto()
137     BROWSER_STRIP_CLICK = auto()
138     COPY_CLICK = auto()
139     DELETE_CLICK = auto()
140     REVIEW_CLICK = auto()
141     FILTER_COLUMN_CLICK = auto()
142     FILTER_ASCEND_CLICK = auto()
143     PAGE_CLICK = auto()
144     HELP_CLICK = auto()
145
146 class GameEventType(StrEnum):
147     BOARD_CLICK = auto()
148     PIECE_CLICK = auto()
149     PAUSE_CLICK = auto()
150     MENU_CLICK = auto()
151     GAME_CLICK = auto()
152     HELP_CLICK = auto()
153     TUTORIAL_CLICK = auto()
154     RESIGN_CLICK = auto()
155     DRAW_CLICK = auto()
156     REVIEW_CLICK = auto()
157     PIECE_DROP = auto()
158     UPDATE_PIECES = auto()
159     ROTATE_PIECE = auto()
160     SET_LASER = auto()
161     TIMER_END = auto()
162

```

```

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

```

```

225
226     MOVE = 4
227     POSITION = 11
228     PHAROAH_SAFETY = 31
229     CHECKMATE = 100000
230     INFINITE = 6969696969
231
232     class Rank(IntEnum):
233         ONE = 0
234         TWO = 1
235         THREE = 2
236         FOUR = 3
237         FIVE = 4
238         SIX = 5
239         SEVEN = 6
240         EIGHT = 7
241
242     class File(IntEnum):
243         A = 0
244         B = 1
245         C = 2
246         D = 3
247         E = 4
248         F = 5
249         G = 6
250         H = 7
251         I = 8
252         J = 9
253
254     class Rotation(StrEnum):
255         UP = 'a'
256         RIGHT = 'b'
257         DOWN = 'c'
258         LEFT = 'd'
259
260     def to_angle(self):
261         if self == Rotation.UP:
262             return 0
263         elif self == Rotation.RIGHT:
264             return 270
265         elif self == Rotation.DOWN:
266             return 180
267         elif self == Rotation.LEFT:
268             return 90
269
270     def get_clockwise(self):
271         if self == Rotation.UP:
272             return Rotation.RIGHT
273         elif self == Rotation.RIGHT:
274             return Rotation.DOWN
275         elif self == Rotation.DOWN:
276             return Rotation.LEFT
277         elif self == Rotation.LEFT:
278             return Rotation.UP
279
280     def get_anticlockwise(self):
281         if self == Rotation.UP:
282             return Rotation.LEFT
283         elif self == Rotation.RIGHT:
284             return Rotation.UP
285         elif self == Rotation.DOWN:
286             return Rotation.RIGHT

```

```

287         elif self == Rotation.LEFT:
288             return Rotation.DOWN
289
290     def get_opposite(self):
291         return self.get_clockwise().get_clockwise()
292
293     class RotationIndex(IntEnum):
294         FIRSTBIT = 0
295         SECONDBIT = 1
296
297     class RotationDirection(StrEnum):
298         CLOCKWISE = 'cw'
299         ANTICLOCKWISE = 'acw'
300
301     def get_opposite(self):
302         if self == RotationDirection.CLOCKWISE:
303             return RotationDirection.ANTICLOCKWISE
304         elif self == RotationDirection.ANTICLOCKWISE:
305             return RotationDirection.CLOCKWISE
306
307     class MoveType(StrEnum):
308         MOVE = 'm'
309         ROTATE = 'r'
310
311     class LaserType(IntEnum):
312         END = 0
313         STRAIGHT = 1
314         CORNER = 2
315
316     class LaserDirection(IntEnum):
317         FROM_TOP = 1
318         FROM_RIGHT = 2
319         FROM_BOTTOM = 3
320         FROM_LEFT = 4
321
322
323 1 import pygame
324 2 from data.components.widget_group import WidgetGroup
325 3 from data.managers.logs import initialise_logger
326 4 from data.managers.cursor import CursorManager
327 5 from data.managers.animation import animation
328 6 from data.managers.window import window
329 7 from data.managers.audio import audio
330 8 from data.managers.theme import theme
331 9 from data.assets import DEFAULT_FONT
332
333 10
334 11 logger = initialise_logger(__file__)
335 12
336 13 FPS = 60
337 14 SHOW_FPS = False
338 15 start_ticks = pygame.time.get_ticks()
339 16
340 17 class Control:
341 18     def __init__(self):
342 19         self.done = False
343 20         self._clock = pygame.time.Clock()
344 21
345 22     def setup_states(self, state_dict, start_state):
346 23         self.state_dict = state_dict
347 24         self.state_name = start_state
348 25
349 26         self.state = self.state_dict[self.state_name]
350 27         self.state.startup()

```

```

28
29     def flip_state(self):
30         self.state.done = False
31         persist = self.state.cleanup()
32
33         previous, self.state_name = self.state_name, self.state.next
34
35         self.state = self.state_dict[self.state_name]
36         self.state.previous = previous
37         self.state.startup(persist)
38
39     def update(self):
40         if self.state.quit:
41             self.done = True
42         elif self.state.done:
43             self.flip_state()
44
45         self._clock.tick(FPS)
46         animation.set_delta_time()
47
48         self.state.update()
49
50         if SHOW_FPS:
51             self.draw_fps()
52
53         window.update()
54
55     def main_game_loop(self):
56         while not self.done:
57             self.event_loop()
58             self.update()
59
60     def update_window(self, resize=False):
61         if resize:
62             self.update_native_window_size()
63             window.handle_resize()
64             self.state.handle_resize()
65
66         self.update()
67
68     def draw_fps(self):
69         fps = str(int(self._clock.get_fps()))
70         DEFAULT_FONT.strength = 0.1
71         DEFAULT_FONT.render_to(window.screen, (0, 0), fps, fgcolor=theme['
textError'], size=15)
72
73     def update_native_window_size(self):
74         x, y = window.size
75
76         max_window_x = 100000
77         max_window_y = x / 1.4
78         min_window_x = 400
79         min_window_y = min_window_x/1.4
80
81         if x / y < 1.4:
82             min_window_x = x
83
84         min_window_size = (min_window_x, min_window_y)
85         max_window_size = (max_window_x, max_window_y)
86         window.minimum_size = min_window_size
87         window.maximum_size = max_window_size
88

```

```

89     def event_loop(self):
90         for event in pygame.event.get():
91             if event.type == pygame.QUIT:
92                 self.done = True
93
94             if event.type == pygame.MOUSEBUTTONDOWN and event.button != 1: # ONLY
PROCESS LEFT CLICKS
95                 return
96
97                 self.state.get_event(event)
98
99 class _State:
100     def __init__(self):
101         self.next = None
102         self.previous = None
103         self.done = False
104         self.quit = False
105         self.persist = {}
106
107         self._cursor = CursorManager()
108         self._widget_group = None
109
110     def startup(self, widgets=None, music=None):
111         if widgets:
112             self._widget_group = WidgetGroup(widgets)
113             self._widget_group.handle_resize(window.size)
114
115         if music:
116             audio.play_music(music)
117
118         logger.info(f'starting {self.__class__.__name__.lower()}.py')
119
120     def cleanup(self):
121         logger.info(f'cleaning {self.__class__.__name__.lower()}.py')
122
123     def draw(self):
124         raise NotImplementedError
125
126     def get_event(self, event):
127         raise NotImplementedError
128
129     def handle_resize(self):
130         self._widget_group.handle_resize(window.size)
131
132     def update(self, **kwargs):
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
8
9 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()
13 sfx_path_2 = (Path(__file__).parent / '../resources/sfx/loading_screen/

```



```

14         loading_screen_2.wav').resolve()
15 def easeOutBack(progress):
16     """
17     Represents a cubic function for easing the logo position.
18     Starts quickly and has small overshoot, then ends slowly.
19
20     Args:
21         progress (float): x-value for cubic function ranging from 0-1.
22
23     Returns:
24         float:  $2.70x^3 + 1.70x^2 + 0x + 1$ , where x is time elapsed.
25     """
26     c2 = 1.70158
27     c3 = 2.70158
28
29     return c3 * ((progress - 1) ** 3) + c2 * ((progress - 1) ** 2) + 1
30
31 class LoadingScreen:
32     def __init__(self, target_func):
33         """
34         Creates new thread, and sets the load_state() function as its target.
35         Then starts draw loop for the loading screen.
36
37         Args:
38             target_func (Callable): function to be run on thread.
39         """
40         self._clock = pygame.time.Clock()
41         self._thread = threading.Thread(target=target_func)
42         self._thread.start()
43
44         self._logo_surface = load_gfx(logo_gfx_path)
45         self._logo_surface = pygame.transform.scale(self._logo_surface, (96, 96))
46         audio.play_sfx(load_sfx(sfx_path_1))
47         audio.play_sfx(load_sfx(sfx_path_2))
48
49         self.run()
50
51     @property
52     def logo_position(self):
53         duration = 1000
54         displacement = 50
55         elapsed_ticks = pygame.time.get_ticks() - start_ticks
56         progress = min(1, elapsed_ticks / duration)
57         center_pos = ((window.screen.size[0] - self._logo_surface.size[0]) / 2, (
58             window.screen.size[1] - self._logo_surface.size[1]) / 2)
59
60         return (center_pos[0], center_pos[1] + displacement - displacement *
61             easeOutBack(progress))
62
63     @property
64     def logo_opacity(self):
65         return min(255, (pygame.time.get_ticks() - start_ticks) / 5)
66
67     @property
68     def duration_not_over(self):
69         return (pygame.time.get_ticks() - start_ticks) < 1500
70
71     def event_loop(self):
72         """
73         Handles events for the loading screen, no user input is taken except to
74         quit the game.

```

```

72         """
73         for event in pygame.event.get():
74             if event.type == pygame.QUIT:
75                 pygame.quit()
76                 sys.exit()
77
78     def draw(self):
79         """
80         Draws logo to screen.
81         """
82         window.screen.fill((0, 0, 0))
83
84         self._logo_surface.set_alpha(self.logo_opacity)
85         window.screen.blit(self._logo_surface, self.logo_position)
86
87         window.update()
88
89     def run(self):
90         """
91         Runs while the thread is still setting up our screens, or the minimum
92         loading screen duration is not reached yet.
93         """
94         while self._thread.is_alive() or self.duration_not_over:
95             self.event_loop()
96             self.draw()
97             self._clock.tick(FPS)
98
99
100 1 from sys import platform
101 2 # Initialises Pygame
102 3 import data.setup
103 4
104 5 # Windows OS requires some configuration for Pygame to scale GUI continuously
105   while window is being resized
106 6 if platform == 'win32':
107 7     import data.windows_setup as win_setup
108 8
109 9 from data.loading_screen import LoadingScreen
110 10
111 11 states = [None, None]
112 12
113 13 def load_states():
114 14     """
115 15     Initialises instances of all screens, executed on another thread with results
116   being stored to the main thread by modifying a mutable such as the states list
117 16     """
118 17     from data.control import Control
119 18     from data.states.game.game import Game
120 19     from data.states.menu.menu import Menu
121 20     from data.states.settings.settings import Settings
122 21     from data.states.config.config import Config
123 22     from data.states.browser.browser import Browser
124 23     from data.states.review.review import Review
125 24     from data.states.editor.editor import Editor
126 25
127 26     state_dict = {
128 27         'menu': Menu(),
129 28         'game': Game(),
130 29         'settings': Settings(),
131 30         'config': Config(),
132 31         'browser': Browser(),
133 32         'review': Review(),
134 33         'editor': Editor()

```

```

34     }
35
36     app = Control()
37
38     states[0] = app
39     states[1] = state_dict
40
41     loading_screen = LoadingScreen(load_states)
42
43     def main():
44         """
45         Executed by run.py, starts main game loop
46         """
47         app, state_dict = states
48
49         if platform == 'win32':
50             win_setup.set_win_resize_func(app.update_window)
51
52         app.setup_states(state_dict, 'menu')
53         app.main_game_loop()
54
55
56 1 import pygame
57 2
58 3 pygame.mixer.init()
59 4 pygame.init()
60 5
61 6 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MAJOR_VERSION, 3)
62 7 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MINOR_VERSION, 3)
63 8 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_PROFILE_MASK, pygame.
64    GL_CONTEXT_PROFILE_CORE)
65 9 pygame.display.gl_set_attribute(pygame.GL_CONTEXT_FORWARD_COMPATIBLE_FLAG, True)
66
67 1 import win32gui
68 2 import win32con
69 3 import os
70 4 import ctypes
71 5 import sys
72 6
73 7 def wndProc(oldWndProc, draw_callback, hWnd, message, wParam, lParam):
74 8     if message == win32con.WM_SIZING or message == win32con.WM_TIMER: # Don't know
75    what WM_TIMER does
76 9         draw_callback(resize=True)
77 10        win32gui.RedrawWindow(hWnd, None, None, win32con.RDW_INVALIDATE | win32con
78    .RDW_ERASE)
79 11    elif message == win32con.WM_MOVE:
80 12        draw_callback(resize=False)
81 13
82 14    return win32gui.CallWindowProc(oldWndProc, hWnd, message, wParam, lParam)
83 15
84 16 def set_win_resize_func(resize_function):
85 17    oldWndProc = win32gui.SetWindowLong(win32gui.GetForegroundWindow(), win32con.
86    GWL_WNDPROC, lambda *args: wndProc(oldWndProc, resize_function, *args))
87 18
89 19 user32 = ctypes.windll.user32
90 20 user32.SetProcessDPIAware() # To deal with Windows High Text Size / Low Display
91    Resolution Settings
92 21
93 22 if os.name != 'nt' or sys.getwindowsversion()[0] < 6:
94 23    raise NotImplementedError("Incompatible OS!")

```

```

1 {
2     "primaryBoardColour": "0xB98766",
3     "secondaryBoardColour": "0xF3D8B8",
4     "laserColourBlue": "0x0000ff",
5     "laserColourRed": "0xff0000",
6     "displayMode": "windowed",
7     "musicVolume": 0.5,
8     "sfxVolume": 0.5,
9     "particles": true,
10    "opengl": true,
11    "shader": "default"
12 }

1 {
2     "version": 1,
3     "disable_existing_loggers": false,
4     "formatters": {
5         "simple": {
6             "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
7             "datefmt": "%Y-%m-%d %H:%M:%S"
8         }
9     },
10
11    "handlers": {
12        "console": {
13            "class": "logging.StreamHandler",
14            "formatter": "simple",
15            "stream": "ext://sys.stdout"
16        }
17    },
18
19    "root": {
20        "level": "INFO",
21        "handlers": ["console"],
22        "propagate": false
23    }
24 }

1 {
2     "version": 1,
3     "disable_existing_loggers": false,
4     "formatters": {
5         "simple": {
6             "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
7         }
8     },
9
10    "handlers": {
11        "console": {
12            "class": "logging.StreamHandler",
13            "level": "DEBUG",
14            "formatter": "simple",
15            "stream": "ext://sys.stdout"
16        },
17
18        "info_file_handler": {
19            "class": "logging.handlers.RotatingFileHandler",
20            "level": "INFO",
21            "formatter": "simple",
22            "filename": "info.log",
23            "maxBytes": 10485760,
24            "backupCount": 20,

```

```

25     "encoding": "utf8"
26 },
27
28 "error_file_handler": {
29     "class": "logging.handlers.RotatingFileHandler",
30     "level": "ERROR",
31     "formatter": "simple",
32     "filename": "errors.log",
33     "maxBytes": 10485760,
34     "backupCount": 20,
35     "encoding": "utf8"
36 }
37 },
38
39 "loggers": {
40     "my_module": {
41         "level": "ERROR",
42         "handlers": ["console"],
43         "propagate": false
44     }
45 },
46
47 "root": {
48     "level": "INFO",
49     "handlers": ["console", "info_file_handler", "error_file_handler"]
50 }
51 }

1 {
2     "colours": {
3         "text": {
4             "primary": "0xdaf2e9",
5             "secondary": "0xf14e52",
6             "error": "0xf14e52"
7         },
8         "fill": {
9             "primary": "0x1c2638",
10            "secondary": "0xf14e52",
11            "tertiary": "0xdaf2e9",
12            "error": "0x9b222b"
13        },
14        "border": {
15            "primary": "0x9b222b",
16            "secondary": ""
17        }
18    },
19    "dimensions": {
20        "borderRadius": 3,
21        "borderWidth": 5,
22        "margin": 10
23    }
24 }

1 {
2     "primaryBoardColour": "0xB98766",
3     "secondaryBoardColour": "0xF3D8B8",
4     "laserColourBlue": "0x0000ff",
5     "laserColourRed": "0xff0000",
6     "displayMode": "windowed",
7     "musicVolume": 0.085,
8     "sfxVolume": 0.336,
9     "particles": true,

```

```

10     "opengl": true,
11     "shader": "default"
12 }

1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5         self.previous = None
6
7 class CircularLinkedList:
8     def __init__(self, list_to_convert=None):
9         """
10         Initialises a CircularLinkedList object.
11
12         Args:
13             list_to_convert (list, optional): Creates a linked list from existing
14             items. Defaults to None.
15         """
16         self._head = None
17
18         if list_to_convert:
19             for item in list_to_convert:
20                 self.insert_at_end(item)
21
22     def __str__(self):
23         """
24         Returns a string representation of the circular linked list.
25
26         Returns:
27             str: Linked list formatted as string.
28         """
29         if self._head is None:
30             return '| empty |'
31
32         characters = '| -> '
33         current_node = self._head
34         while True:
35             characters += str(current_node.data) + ' -> '
36             current_node = current_node.next
37
38             if current_node == self._head:
39                 characters += '|'
40                 return characters
41
42     def insert_at_beginning(self, data):
43         """
44         Inserts a node at the beginning of the circular linked list.
45
46         Args:
47             data: The data to insert.
48         """
49         new_node = Node(data)
50
51         if self._head is None:
52             self._head = new_node
53             new_node.next = self._head
54             new_node.previous = self._head
55         else:
56             new_node.next = self._head
57             new_node.previous = self._head.previous
58             self._head.previous.next = new_node

```

```

58         self._head.previous = new_node
59
60         self._head = new_node
61
62     def insert_at_end(self, data):
63         """
64         Inserts a node at the end of the circular linked list.
65
66         Args:
67             data: The data to insert.
68         """
69         new_node = Node(data)
70
71         if self._head is None:
72             self._head = new_node
73             new_node.next = self._head
74             new_node.previous = self._head
75         else:
76             new_node.next = self._head
77             new_node.previous = self._head.previous
78             self._head.previous.next = new_node
79             self._head.previous = new_node
80
81     def insert_at_index(self, data, index):
82         """
83         Inserts a node at a specific index in the circular linked list.
84         The head node is taken as index 0.
85
86         Args:
87             data: The data to insert.
88             index (int): The index to insert the data at.
89
90         Raises:
91             ValueError: Index is out of range.
92         """
93         if index < 0:
94             raise ValueError('Invalid index! (CircularLinkedList.insert_at_index)')
95
96         if index == 0 or self._head is None:
97             self.insert_at_beginning(data)
98         else:
99             new_node = Node(data)
100             current_node = self._head
101             count = 0
102
103             while count < index - 1 and current_node.next != self._head:
104                 current_node = current_node.next
105                 count += 1
106
107             if count == (index - 1):
108                 new_node.next = current_node.next
109                 new_node.previous = current_node
110                 current_node.next = new_node
111             else:
112                 raise ValueError('Index out of range! (CircularLinkedList.insert_at_index)')
113
114     def delete(self, data):
115         """
116         Deletes a node with the specified data from the circular linked list.
117

```

```

118         Args:
119             data: The data to delete.
120
121         Raises:
122             ValueError: No nodes in the list contain the specified data.
123         """
124         if self._head is None:
125             return
126
127         current_node = self._head
128
129         while current_node.data != data:
130             current_node = current_node.next
131
132             if current_node == self._head:
133                 raise ValueError('Data not found in circular linked list! (
CircularLinkedList.delete)')
134
135         if self._head.next == self._head:
136             self._head = None
137         else:
138             current_node.previous.next = current_node.next
139             current_node.next.previous = current_node.previous
140
141     def data_in_list(self, data):
142         """
143         Checks if the specified data is in the circular linked list.
144
145         Args:
146             data: The data to check.
147
148         Returns:
149             bool: True if the data is in the list, False otherwise.
150         """
151         if self._head is None:
152             return False
153
154         current_node = self._head
155         while True:
156             if current_node.data == data:
157                 return True
158
159             current_node = current_node.next
160             if current_node == self._head:
161                 return False
162
163     def shift_head(self):
164         """
165         Shifts the head of the circular linked list to the next node.
166         """
167         self._head = self._head.next
168
169     def unshift_head(self):
170         """
171         Shifts the head of the circular linked list to the previous node.
172         """
173         self._head = self._head.previous
174
175     def get_head(self):
176         """
177         Gets the head node of the circular linked list.
178

```



```

179         Returns:
180             Node: The head node.
181         """
182         return self._head

1 import pygame
2
3 class Cursor(pygame.sprite.Sprite):
4     def __init__(self):
5         super().__init__()
6         self.image = pygame.Surface((1, 1))
7         self.image.fill((255, 0, 0))
8         self.rect = self.image.get_rect()
9
10    # def update(self):
11    #     self.rect.center = pygame.mouse.get_pos()
12
13    def get_sprite_collision(self, mouse_pos, square_group):
14        self.rect.center = mouse_pos
15        sprite = pygame.sprite.spritecollideany(self, square_group)
16
17        return sprite

1 from data.constants import GameEventType, SettingsEventType, ConfigEventType,
    BrowserEventType, EditorEventType
2
3 required_args = {
4     GameEventType.BOARD_CLICK: ['coords'],
5     GameEventType.ROTATE_PIECE: ['rotation_direction'],
6     GameEventType.SET_LASER: ['laser_result'],
7     GameEventType.UPDATE_PIECES: ['move_notation'],
8     GameEventType.TIMER_END: ['active_colour'],
9     GameEventType.PIECE_DROP: ['coords', 'piece', 'colour', 'rotation', '
    remove_overlay'],
10    SettingsEventType.COLOUR_SLIDER_SLIDE: ['colour'],
11    SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK: ['colour'],
12    SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK: ['colour'],
13    SettingsEventType.DROPDOWN_CLICK: ['selected_word'],
14    SettingsEventType.VOLUME_SLIDER_CLICK: ['volume', 'volume_type'],
15    SettingsEventType.SHADER_PICKER_CLICK: ['data'],
16    SettingsEventType.PARTICLES_CLICK: ['toggled'],
17    SettingsEventType.OPENGL_CLICK: ['toggled'],
18    ConfigEventType.TIME_TYPE: ['time'],
19    ConfigEventType.FEN_STRING_TYPE: ['time'],
20    ConfigEventType.CPU_DEPTH_CLICK: ['data'],
21    ConfigEventType.PVC_CLICK: ['data'],
22    ConfigEventType.PRESET_CLICK: ['fen_string'],
23    BrowserEventType.BROWSER_STRIP_CLICK: ['selected_index'],
24    BrowserEventType.PAGE_CLICK: ['data'],
25    EditorEventType.PICK_PIECE_CLICK: ['piece', 'active_colour'],
26    EditorEventType.ROTATE_PIECE_CLICK: ['rotation_direction'],
27 }
28
29 class CustomEvent():
30     def __init__(self, type, **kwargs):
31         self.__dict__.update(kwargs)
32         self.type = type
33
34     @classmethod
35     def create_event(event_cls, event_type, **kwargs):
36         """

```

```

37     @classmethod Factory method used to instance CustomEvent object, to check
    for required keyword arguments
38
39     Args:
40         event_cls (CustomEvent): Reference to own class.
41         event_type: The state EventType.
42
43     Raises:
44         ValueError: If required keyword argument for passed event type not
    present.
45         ValueError: If keyword argument passed is not required for passed
    event type.
46
47     Returns:
48         CustomEvent: Initialised CustomEvent instance.
49     """
50     if event_type in required_args:
51
52         for required_arg in required_args[event_type]:
53             if required_arg not in kwargs:
54                 raise ValueError(f"Argument '{required_arg}' required for {
    event_type.name} event (GameEvent.create_event)")
55
56         for kwarg in kwargs:
57             if kwarg not in required_args[event_type]:
58                 raise ValueError(f"Argument '{kwarg}' not included in
    required_args dictionary for event '{event_type}'! (GameEvent.create_event)")
59
60         return event_cls(event_type, **kwargs)
61
62     else:
63         return event_cls(event_type)

```

```

1  from data.constants import Colour
2  from data.states.game.components.move import Move
3
4  class GameEntry:
5      def __init__(self, game_states, final_fen_string):
6          self._game_states = game_states
7          self._final_fen_string = final_fen_string
8
9      def __str__(self):
10         return f'''
11 <GameEntry> :>
12     CPU_ENABLED: {self._game_states['CPU_ENABLED']}
13     CPU_DEPTH: {self._game_states['CPU_DEPTH']},
14     WINNER: {self._game_states['WINNER']},
15     TIME_ENABLED: {self._game_states['TIME_ENABLED']},
16     TIME: {self._game_states['TIME']},
17     NUMBER_OF_PLY: {len(self._game_states['MOVES'])},
18     MOVES: {self.convert_moves(self._game_states['MOVES'])}
19     FINAL_FEN_STRING: {self._final_fen_string}
20     START_FEN_STRING: {self._game_states['START_FEN_STRING']}
21 </GameEntry>
22     '''
23
24     def convert_to_row(self):
25         return (self._game_states['CPU_ENABLED'], self._game_states['CPU_DEPTH'],
    self._game_states['WINNER'], self._game_states['TIME_ENABLED'], self.
    _game_states['TIME'], len(self._game_states['MOVES']), self.convert_moves(self
    ._game_states['MOVES']), self._game_states['START_FEN_STRING'], self.
    _final_fen_string)

```

```

26
27     def convert_moves(self, moves):
28         return '|'.join([
29             f'{round(move['time'][Colour.BLUE], 4)};{round(move['time'][Colour.RED
30             ], 4)};{move['move']}'
31             for move in moves
32         ])
33
34     @staticmethod
35     def parse_moves(move_str):
36         moves = move_str.split('|')
37         return [
38             {
39                 'blue_time': move.split(';')[0],
40                 'red_time': move.split(';')[1],
41                 'move': Move.instance_from_notation(move.split(';')[2]),
42                 'unparsed_move': move.split(';')[2],
43             } for move in moves if move != ''
44         ]
45
46 # self.states = {
47 #     'CPU_ENABLED': game_config['CPU_ENABLED'],
48 #     'CPU_DEPTH': game_config['CPU_DEPTH'],
49 #     'AWAITING_CPU': False,
50 #     'WINNER': None,
51 #     'PAUSED': False,
52 #     'ACTIVE_COLOUR': Colour.BLUE,
53 #     'TIME_ENABLED': game_config['TIME_ENABLED'],
54 #     'TIME': game_config['TIME'],
55 #     'MOVES': []
56 # }
57
58 # move_item = {
59 #     'time': {
60 #         Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
61 #         Colour.RED: GAME_WIDGETS['red_timer'].get_time()
62 #     },
63 #     'move': move_notation,
64 #     'laserResult': laser_result
65 # }
66
67
68 1 import pygame
69 2 from data.managers.window import window
70 3
71 4 class WidgetGroup(pygame.sprite.Group):
72 5     def __init__(self, widget_dict):
73 6         super().__init__()
74 7
75 8         for value in widget_dict.values():
76 9             if isinstance(value, list):
77 10                 for widget in value:
78 11                     self.add(widget)
79 12             elif isinstance(value, dict):
80 13                 for widget in value.values():
81 14                     self.add(widget)
82 15             else:
83 16                 self.add(value)
84 17
85 18     def handle_resize(self, new_surface_size):
86 19         for sprite in self.sprites():
87 20             sprite.set_surface_size(new_surface_size)

```

```

21         sprite.set_image()
22         sprite.set_geometry()
23
24     def process_event(self, event):
25         for sprite in self.sprites():
26             widget_event = sprite.process_event(event)
27
28             if widget_event:
29                 return widget_event
30
31         return None
32
33     def draw(self):
34         sprites = self.sprites()
35         for spr in sprites:
36             surface = spr._surface or window.screen
37             self.spritedict[spr] = surface.blit(spr.image, spr.rect)
38         self.lostsprites = []
39         dirty = self.lostsprites
40
41         return dirty
42
43     def on_widget(self, mouse_pos):
44         test_sprite = pygame.sprite.Sprite()
45         test_sprite.rect = pygame.Rect(*mouse_pos, 1, 1)
46         return pygame.sprite.spritecollideany(test_sprite, self)

```

  

```

1  import sqlite3
2  from pathlib import Path
3
4  database_path = (Path(__file__).parent / '../database.db').resolve()
5
6  def upgrade():
7      connection = sqlite3.connect(database_path)
8      cursor = connection.cursor()
9
10     cursor.execute('''
11         ALTER TABLE games ADD COLUMN created_dt TIMESTAMP NOT NULL
12     ''')
13
14     connection.commit()
15     connection.close()
16
17  def downgrade():
18      connection = sqlite3.connect(database_path)
19      cursor = connection.cursor()
20
21     cursor.execute('''
22         ALTER TABLE games DROP COLUMN created_dt
23     ''')
24
25     connection.commit()
26     connection.close()
27
28  upgrade()
29  # downgrade()

```

  

```

1  import sqlite3
2  from pathlib import Path
3
4  database_path = (Path(__file__).parent / '../database.db').resolve()
5

```

```

6 def upgrade():
7     connection = sqlite3.connect(database_path)
8     cursor = connection.cursor()
9
10    cursor.execute('''
11        ALTER TABLE games ADD COLUMN fen_string TEXT NOT NULL
12    ''')
13
14    connection.commit()
15    connection.close()
16
17 def downgrade():
18     connection = sqlite3.connect(database_path)
19     cursor = connection.cursor()
20
21    cursor.execute('''
22        ALTER TABLE games DROP COLUMN fen_string
23    ''')
24
25    connection.commit()
26    connection.close()
27
28 upgrade()

1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     connection = sqlite3.connect(database_path)
8     cursor = connection.cursor()
9
10    cursor.execute('''
11        ALTER TABLE games ADD COLUMN start_fen_string TEXT NOT NULL
12    ''')
13
14    connection.commit()
15    connection.close()
16
17 def downgrade():
18     connection = sqlite3.connect(database_path)
19     cursor = connection.cursor()
20
21    cursor.execute('''
22        ALTER TABLE games DROP COLUMN start_fen_string
23    ''')
24
25    connection.commit()
26    connection.close()
27
28 upgrade()
29 # downgrade()

1 import sqlite3
2 from pathlib import Path
3
4 database_path = (Path(__file__).parent / '../database.db').resolve()
5
6 def upgrade():
7     """
8     Upgrade function to rename fen_string column.

```

```

9         """
10        connection = sqlite3.connect(database_path)
11        cursor = connection.cursor()
12
13        cursor.execute('''
14            ALTER TABLE games RENAME COLUMN fen_string TO final_fen_string
15        ''')
16
17        connection.commit()
18        connection.close()
19
20    def downgrade():
21        """
22        Downgrade function to revert fen_string column renaming.
23        """
24        connection = sqlite3.connect(database_path)
25        cursor = connection.cursor()
26
27        cursor.execute('''
28            ALTER TABLE games RENAME COLUMN final_fen_string TO fen_string
29        ''')
30
31        connection.commit()
32        connection.close()
33
34    upgrade()
35    # downgrade()
36
37    1 import sqlite3
38    2 from pathlib import Path
39    3
40    4 database_path = (Path(__file__).parent / '../database.db').resolve()
41    5
42    6 def upgrade():
43    7         """
44    8         Upgrade function to create games table.
45    9         """
50    10        connection = sqlite3.connect(database_path)
51    11        cursor = connection.cursor()
52    12
53    13        cursor.execute('''
54    14            CREATE TABLE games(
55    15                id INTEGER PRIMARY KEY,
56    16                cpu_enabled INTEGER NOT NULL,
57    17                cpu_depth INTEGER,
58    18                winner INTEGER,
59    19                time_enabled INTEGER NOT NULL,
60    20                time REAL,
61    21                number_of_ply INTEGER NOT NULL,
62    22                moves TEXT NOT NULL
63    23            )
64    24        ''')
65    25
66    26        connection.commit()
67    27        connection.close()
68    28
69    29 def downgrade():
70    30         """
71    31         Downgrade function to revert table creation.
72    32         """
73    33        connection = sqlite3.connect(database_path)
74    34        cursor = connection.cursor()

```

```

35
36         cursor.execute('''
37             DROP TABLE games
38         ''')
39
40         connection.commit()
41         connection.close()
42
43     upgrade()
44     # downgrade()
45
46
47 1 import pygame
48 2 from data.utils.asset_helpers import scale_and_cache
49 3
50 4 FPS = 60
51 5
52 6 class AnimationManager:
53 7     def __init__(self):
54 8         self._current_ms = 0
55 9         self._timers = []
56 10
57 11     def set_delta_time(self):
58 12         self._current_ms = pygame.time.get_ticks()
59 13
60 14         for timer in self._timers:
61 15             start_ms, target_ms, callback = timer
62 16             if self._current_ms - start_ms >= target_ms:
63 17                 callback()
64 18                 self._timers.remove(timer)
65 19
66 20     def calculate_frame_index(self, start_index, end_index, fps):
67 21         ms_per_frame = int(1000 / fps)
68 22         return start_index + ((self._current_ms // ms_per_frame) % (end_index -
69 start_index))
70 23
71 24     def draw_animation(self, screen, animation, position, size, fps=8):
72 25         frame_index = self.calculate_frame_index(0, len(animation), fps)
73 26         scaled_animation = scale_and_cache(animation[frame_index], size)
74 27         screen.blit(scaled_animation, position)
75 28
76 29     def draw_image(self, screen, image, position, size):
77 30         scaled_background = scale_and_cache(image, size)
78 31         screen.blit(scaled_background, position)
79 32
80 33     def set_timer(self, target_ms, callback):
81 34         self._timers.append((self._current_ms, target_ms, callback))
82 35
83 36 animation = AnimationManager()
84
85
86 1 import pygame
87 2 from data.utils.data_helpers import get_user_settings
88 3 from data.managers.logs import initialise_logger
89 4
90 5 logger = initialise_logger(__name__)
91 6 user_settings = get_user_settings()
92 7
93 8 class AudioManager:
94 9     def __init__(self, num_channels=16):
95 10         pygame.mixer.set_num_channels(num_channels)
96 11
97 12         self._music_volume = user_settings['musicVolume']
98 13         self._sfx_volume = user_settings['sfxVolume']

```

```

14
15         self._current_song = None
16         self._current_channels = []
17
18     def set_sfx_volume(self, volume):
19         self._sfx_volume = volume
20
21         for channel in self._current_channels:
22             channel.set_volume(self._sfx_volume)
23
24     def set_music_volume(self, volume):
25         self._music_volume = volume
26
27         pygame.mixer.music.set_volume(self._music_volume)
28
29     def pause_sfx(self):
30         pygame.mixer.pause()
31
32     def unpause_sfx(self):
33         pygame.mixer.unpause()
34
35     def stop_sfx(self, fadeout=0):
36         pygame.mixer.fadeout(fadeout)
37
38     def remove_unused_channels(self):
39         unused_channels = []
40         for channel in self._current_channels:
41             if channel.get_busy() is False:
42                 unused_channels.append(channel)
43
44         return unused_channels
45
46     def play_sfx(self, sfx, loop=False):
47         unused_channels = self.remove_unused_channels()
48
49         if len(unused_channels) == 0:
50             channel = pygame.mixer.find_channel()
51         else:
52             channel = unused_channels.pop(0)
53
54         if channel is None:
55             logger.warning('No available channel for SFX')
56             return
57
58         self._current_channels.append(channel)
59         channel.set_volume(self._sfx_volume)
60
61         if loop:
62             channel.play(sfx, loops=-1)
63         else:
64             channel.play(sfx)
65
66     def play_music(self, music_path):
67         if 'menu' in str(music_path) and 'menu' in str(self._current_song):
68             return
69
70         if music_path == self._current_song:
71             return
72
73         pygame.mixer.music.stop()
74         pygame.mixer.music.unload()
75         pygame.mixer.music.load(music_path)

```



```

76         pygame.mixer.music.set_volume(self._music_volume)
77         pygame.mixer.music.play(loops=-1)
78
79         self._current_song = music_path
80
81 audio = AudioManager()

1 import pygame
2 from data.assets import GRAPHICS
3 from data.constants import CursorMode
4
5 class CursorManager:
6     def __init__(self):
7         self._mode = CursorMode.ARROW
8         self.set_mode(CursorMode.ARROW)
9
10    def set_mode(self, mode):
11        pygame.mouse.set_visible(True)
12
13        match mode:
14            case CursorMode.ARROW:
15                pygame.mouse.set_cursor((7, 5), pygame.transform.scale(GRAPHICS['
16                arrow'], (32, 32)))
17            case CursorMode.IBEAM:
18                pygame.mouse.set_cursor((15, 5), pygame.transform.scale(GRAPHICS['
19                ibeam'], (32, 32)))
20            case CursorMode.OPENHAND:
21                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['
22                hand_open'], (32, 32)))
23            case CursorMode.CLOSEDHAND:
24                pygame.mouse.set_cursor((17, 5), pygame.transform.scale(GRAPHICS['
25                hand_closed'], (32, 32)))
26            case CursorMode.NO:
27                pygame.mouse.set_visible(False)
28
29        self._mode = mode
30
31    def get_mode(self):
32        return self._mode
33
34 cursor = CursorManager()

1 import logging.config
2 from data.utils.data_helpers import load_json
3 from pathlib import Path
4 import logging
5
6 config_path = (Path(__file__).parent / '../app_data/logs_config.json').resolve()
7 config = load_json(config_path)
8 logging.config.dictConfig(config)
9
10 def initialise_logger(file_path):
11     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 SMPProtocol
6 from data.constants import ShaderType
7

```

```

8  shader_path = (Path(__file__).parent / '../shaders/').resolve()
9
10 SHADER_PRIORITY = [
11     ShaderType.CRT,
12     ShaderType.SHAKE,
13     ShaderType.BLOOM,
14     ShaderType.CHROMATIC_ABBREVIATION,
15     ShaderType.RAYS,
16     ShaderType.GRAYSCALE,
17     ShaderType.BASE,
18 ]
19
20 pygame_quad_array = array('f', [
21     -1.0, 1.0, 0.0, 0.0,
22     1.0, 1.0, 1.0, 0.0,
23     -1.0, -1.0, 0.0, 1.0,
24     1.0, -1.0, 1.0, 1.0,
25 ])
26
27 opengl_quad_array = array('f', [
28     -1.0, -1.0, 0.0, 0.0,
29     1.0, -1.0, 1.0, 0.0,
30     -1.0, 1.0, 0.0, 1.0,
31     1.0, 1.0, 1.0, 1.0,
32 ])
33
34 class ShaderManager(SMProtocol):
35     def __init__(self, ctx: moderngl.Context, screen_size):
36         self.ctx = ctx
37         self.ctx.gc_mode = 'auto'
38
39         self._screen_size = screen_size
40         self._opengl_buffer = self.ctx.buffer(data=opengl_quad_array)
41         self._pygame_buffer = self.ctx.buffer(data=pygame_quad_array)
42         self._shader_list = [ShaderType.BASE]
43
44         self._vert_shaders = {}
45         self._frag_shaders = {}
46         self._programs = {}
47         self._vaos = {}
48         self._textures = {}
49         self._shader_passes = {}
50         self.framebuffers = {}
51
52         self.load_shader(ShaderType.BASE)
53         self.load_shader(ShaderType._CALIBRATE)
54         self.create_framebuffer(ShaderType._CALIBRATE)
55
56     def load_shader(self, shader_type, **kwargs):
57         """
58         Loads a given shader by creating a VAO reading the corresponding .frag
59         file.
60
61         Args:
62             shader_type (ShaderType): The type of shader to load.
63             **kwargs: Additional arguments passed when initialising the fragment
64             shader class.
65         """
66         self._shader_passes[shader_type] = shader_pass_lookup[shader_type](self,
67 **kwargs)
68         self.create_vao(shader_type)

```

```

67     def clear_shaders(self):
68         """
69         Clears the shader list, leaving only the base shader.
70         """
71         self._shader_list = [ShaderType.BASE]
72
73     def create_vao(self, shader_type):
74         """
75         Creates a vertex array object (VAO) for the given shader type.
76
77         Args:
78             shader_type (ShaderType): The type of shader.
79         """
80         frag_name = shader_type[1:] if shader_type[0] == '_' else shader_type
81         vert_path = Path(shader_path / 'vertex/base.vert').resolve()
82         frag_path = Path(shader_path / f'fragments/{frag_name}.frag').resolve()
83
84         self._vert_shaders[shader_type] = vert_path.read_text()
85         self._frag_shaders[shader_type] = frag_path.read_text()
86
87         program = self._ctx.program(vertex_shader=self._vert_shaders[shader_type],
88                                     fragment_shader=self._frag_shaders[shader_type])
89         self._programs[shader_type] = program
90
91         if shader_type == ShaderType._CALIBRATE:
92             self._vaos[shader_type] = self._ctx.vertex_array(self._programs[
93                 shader_type], [(self._pygame_buffer, '2f 2f', 'vert', 'texCoords')])
94         else:
95             self._vaos[shader_type] = self._ctx.vertex_array(self._programs[
96                 shader_type], [(self._opengl_buffer, '2f 2f', 'vert', 'texCoords')])
97
98     def create_framebuffer(self, shader_type, size=None, filter=moderngl.NEAREST):
99         """
100         Creates a framebuffer for the given shader type.
101
102         Args:
103             shader_type (ShaderType): The type of shader.
104             size (tuple[int, int], optional): The size of the framebuffer.
105             Defaults to screen size.
106             filter (moderngl.Filter, optional): The texture filter. Defaults to
107             NEAREST.
108         """
109         texture_size = size or self._screen_size
110         texture = self._ctx.texture(size=texture_size, components=4)
111         texture.filter = (filter, filter)
112
113         self._textures[shader_type] = texture
114         self.framebuffers[shader_type] = self._ctx.framebuffer(color_attachments=[
115             self._textures[shader_type]])
116
117     def render_to_fbo(self, shader_type, texture, output_fbo=None, program_type=
118         None, use_image=True, **kwargs):
119         """
120         Applies the shaders and renders the resultant texture to a framebuffer
121         object (FBO).
122
123         Args:
124             shader_type (ShaderType): The type of shader.
125             texture (moderngl.Texture): The texture to render.
126             output_fbo (moderngl.Framebuffer, optional): The output framebuffer.
127             Defaults to None.
128             program_type (ShaderType, optional): The program type. Defaults to

```

```

None.
120         use_image (bool, optional): Whether to use the image uniform. Defaults
        to True.
121         **kwargs: Additional uniforms for the fragment shader.
122         """
123         fbo = output_fbo or self.framebuffers[shader_type]
124         program = self._programs[program_type] if program_type else self._programs
        [shader_type]
125         vao = self._vaos[program_type] if program_type else self._vaos[shader_type]
126
127         fbo.use()
128         texture.use(0)
129
130         if use_image:
131             program['image'] = 0
132             for uniform, value in kwargs.items():
133                 program[uniform] = value
134
135         vao.render(mode=moderngl.TRIANGLE_STRIP)
136
137     def apply_shader(self, shader_type, **kwargs):
138         """
139         Applies a shader of the given type and adds it to the list.
140
141         Args:
142             shader_type (ShaderType): The type of shader to apply.
143
144         Raises:
145             ValueError: If the shader is already being applied.
146         """
147         if shader_type in self._shader_list:
148             return
149
150         self.load_shader(shader_type, **kwargs)
151         self._shader_list.append(shader_type)
152
153         # Sort shader list based on the order in SHADER_PRIORITY, so that more
        important shaders are applied first
154         self._shader_list.sort(key=lambda shader: -SHADER_PRIORITY.index(shader))
155
156     def remove_shader(self, shader_type):
157         """
158         Removes a shader of the given type from the list.
159
160         Args:
161             shader_type (ShaderType): The type of shader to remove.
162         """
163         if shader_type in self._shader_list:
164             self._shader_list.remove(shader_type)
165
166     def render_output(self):
167         """
168         Renders the final output to the screen.
169         """
170         # Render to the screen framebuffer
171         self._ctx.screen.use()
172
173         # Take the texture of the last framebuffer to be rendered to, and render
        that to the screen framebuffer
174         output_shader_type = self._shader_list[-1]
175         self.get_fbo_texture(output_shader_type).use(0)
176         self._programs[output_shader_type]['image'] = 0

```

```

177         self._vaos[output_shader_type].render(mode=moderngl.TRIANGLE_STRIP)
178
179
180     def get_fbo_texture(self, shader_type):
181         """
182         Gets the texture from the specified shader type's FBO.
183
184         Args:
185             shader_type (ShaderType): The type of shader.
186
187         Returns:
188             moderngl.Texture: The texture from the FBO.
189         """
190         return self.framebuffers[shader_type].color_attachments[0]
191
192     def calibrate_pygame_surface(self, pygame_surface):
193         """
194         Converts the Pygame window surface into an OpenGL texture.
195
196         Args:
197             pygame_surface (pygame.Surface): The finished Pygame surface.
198
199         Returns:
200             moderngl.Texture: The calibrated texture.
201         """
202         texture = self._ctx.texture(pygame_surface.size, 4)
203         texture.filter = (moderngl.NEAREST, moderngl.NEAREST)
204         texture.swizzle = 'BGRA'
205         # Take the Pygame surface's pixel array and draw it to the new texture
206         texture.write(pygame_surface.get_view('1'))
207
208         # ShaderType._CALIBRATE has a VAO containing the pygame_quad_array
209         # coordinates, as Pygame uses different texture coordinates than ModernGL
210         # textures
211         self.render_to_fbo(ShaderType._CALIBRATE, texture)
212         return self.get_fbo_texture(ShaderType._CALIBRATE)
213
214     def draw(self, surface, arguments):
215         """
216         Draws the Pygame surface with shaders applied to the screen.
217
218         Args:
219             surface (pygame.Surface): The final Pygame surface.
220             arguments (dict): A dict of { ShaderType: Args } items, containing
221             keyword arguments for every fragment shader.
222         """
223         self._ctx.viewport = (0, 0, *self._screen_size)
224         texture = self.calibrate_pygame_surface(surface)
225
226         for shader_type in self._shader_list:
227             self._shader_passes[shader_type].apply(texture, **arguments.get(
228                 shader_type, {}))
229             texture = self.get_fbo_texture(shader_type)
230
231         self.render_output()
232
233     def __del__(self):
234         """
235         Cleans up ModernGL resources when the ShaderManager object is deleted.
236         """
237         self.cleanup()

```

```

235     def cleanup(self):
236         """
237         Cleans up resources used by the ModernGL.
238         Probably unnecessary as the 'auto' garbage collection mode is used.
239         """
240         self._pygame_buffer.release()
241         self._opengl_buffer.release()
242         for program in self._programs:
243             self._programs[program].release()
244         for texture in self._textures:
245             self._textures[texture].release()
246         for vao in self._vaos:
247             self._vaos[vao].release()
248         for framebuffer in self.framebuffers:
249             self.framebuffers[framebuffer].release()
250
251     def handle_resize(self, new_screen_size):
252         """
253         Handles resizing of the screen.
254
255         Args:
256             new_screen_size (tuple[int, int]): The new screen size.
257         """
258         self._screen_size = new_screen_size
259
260         # Recreate all framebuffers to prevent scaling issues
261         for shader_type in self.framebuffers:
262             filter = self._textures[shader_type].filter[0]
263             self.create_framebuffer(shader_type, size=self._screen_size, filter=
filter)

1  from data.utils.data_helpers import get_themes, get_user_settings
2
3  themes = get_themes()
4  user_settings = get_user_settings()
5
6  def flatten_dictionary_generator(dictionary, parent_key=None):
7      """
8      Recursive depth-first search to yield all items in a dictionary.
9
10     Args:
11         dictionary (dict): Dictionary to be iterated through.
12         parent_key (str, optional): Prefix added to every key. Defaults to None.
13
14     Yields:
15         dict | tuple[str, str]: Another dictionary or key, value pair.
16     """
17     for key, value in dictionary.items():
18         if parent_key:
19             new_key = parent_key + key.capitalize()
20         else:
21             new_key = key
22
23         if isinstance(value, dict):
24             yield from flatten_dictionary(value, new_key).items()
25         else:
26             yield new_key, value
27
28     def flatten_dictionary(dictionary, parent_key=''):
29         return dict(flatten_dictionary_generator(dictionary, parent_key))
30
31     class ThemeManager:

```

```

32     def __init__(self):
33         self.__dict__.update(flatten_dictionary(themes['colours']))
34         self.__dict__.update(flatten_dictionary(themes['dimensions']))
35
36     def __getitem__(self, arg):
37         """
38         Override default class's __getitem__ dunder method, to make retrieving an
39         instance attribute nicer with [] notation.
40
41         Args:
42             arg (str): Attribute name.
43
44         Raises:
45             KeyError: Instance does not have requested attribute.
46
47         Returns:
48             str | int: Instance attribute.
49         """
50         item = self.__dict__.get(arg)
51
52         if item is None:
53             raise KeyError('(ThemeManager.__getitem__) Requested theme item not
54             found:', arg)
55
56         return item
57
58 theme = ThemeManager()
59
60
61 1 import pygame
62 2 import moderngl
63 3 from data.constants import ShaderType, SCREEN_SIZE, SHADER_MAP
64 4 from data.utils.data_helpers import get_user_settings
65 5 from data.utils.asset_helpers import draw_background
66 6 from data.managers.shader import ShaderManager
67 7
68 8 user_settings = get_user_settings()
69 9 is_opengl = user_settings['opengl']
70 10 is_fullscreen = user_settings['displayMode'] == 'fullscreen'
71 11
72 12 class WindowManager(pygame.Window):
73 13     def __init__(self, **kwargs):
74 14         super().__init__(**kwargs)
75 15         self._native_screen = self.get_surface() # Initialise convert format
76 16         self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
77 17
78 18         if is_opengl:
79 19             self._ctx = moderngl.create_context()
80 20             self._shader_manager = ShaderManager(self._ctx, screen_size=self.size)
81 21
82 22             self.shader_arguments = {
83 23                 ShaderType.BASE: {},
84 24                 ShaderType.SHAKE: {},
85 25                 ShaderType.BLOOM: {},
86 26                 ShaderType.GRAYSCALE: {},
87 27                 ShaderType.CRT: {},
88 28                 ShaderType.RAYS: {}
89 29             }
90 30
91 31             if (selected_shader := get_user_settings()['shader']) is not None:
92 32                 for shader_type in SHADER_MAP[selected_shader]:
93 33                     self.set_effect(shader_type)
94 34         else:

```

```

35         from data.assets import GRAPHICS
36         self._background_image = GRAPHICS['temp_background']
37
38     def set_effect(self, effect, **kwargs):
39         if is_opengl:
40             self._shader_manager.apply_shader(effect, **kwargs)
41
42     def set_apply_arguments(self, effect, **kwargs):
43         if is_opengl:
44             self.shader_arguments[effect] = kwargs
45
46     def clear_apply_arguments(self, effect):
47         if is_opengl:
48             self.shader_arguments[effect] = {}
49
50     def clear_effect(self, effect):
51         if is_opengl:
52             self._shader_manager.remove_shader(effect)
53             self.clear_apply_arguments(effect)
54
55     def clear_all_effects(self, clear_arguments=False):
56         if is_opengl:
57             self._shader_manager.clear_shaders()
58
59             if clear_arguments:
60                 for shader_type in self.shader_arguments:
61                     self.shader_arguments[shader_type] = {}
62
63     def draw(self):
64         if is_opengl:
65             self._shader_manager.draw(self.screen, self.shader_arguments)
66         else:
67             self._native_screen.blit(self.screen, (0, 0))
68
69         self.flip()
70
71         if is_opengl:
72             self.screen.fill((0, 0, 0, 0))
73         else:
74             self.screen.fill((0, 0, 0))
75             draw_background(self.screen, self._background_image)
76
77     def update(self):
78         self.draw()
79
80     def handle_resize(self):
81         self.screen = pygame.Surface(self.size, pygame.SRCALPHA)
82         if is_opengl:
83             self._shader_manager.handle_resize(self.size)
84         else:
85             draw_background(self.screen, self._background_image)
86
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
5
6 class SMPProtocol(Protocol):
7     def load_shader(self, shader_type: ShaderType, **kwargs) -> None: ...

```



```

8     def clear_shaders(self) -> None: ...
9     def create_vao(self, shader_type: ShaderType) -> None: ...
10    def create_framebuffer(self, shader_type: ShaderType, size: Optional[tuple[int
11    ]] = None, filter: Optional[int] = moderngl.NEAREST) -> None: ...
12    def render_to_fbo(self, shader_type: ShaderType, texture: moderngl.Texture,
13    output_fbo: Optional[moderngl.Framebuffer] = None, program_type: Optional[
14    ShaderType] = None, use_image: Optional[bool] = True, **kwargs) -> None: ...
15    def apply_shader(self, shader_type: ShaderType, **kwargs) -> None: ...
16    def remove_shader(self, shader_type: ShaderType) -> None: ...
17    def render_output(self, texture: moderngl.Texture) -> None: ...
18    def get_fbo_texture(self, shader_type: ShaderType) -> moderngl.Texture: ...
19    def calibrate_pygame_surface(self, pygame_surface: pygame.Surface) -> moderngl
20    .Texture: ...
21    def draw(self, surface: pygame.Surface, arguments: dict) -> None: ...
22    def __del__(self) -> None: ...
23    def cleanup(self) -> None: ...
24    def handle_resize(self, new_screen_size: tuple[int]) -> None: ...
25
26    _ctx: moderngl.Context
27    _screen_size: tuple[int]
28    _opengl_buffer: moderngl.Buffer
29    _pygame_buffer: moderngl.Buffer
30    _shader_stack: list[ShaderType]
31
32    _vert_shaders: dict
33    _frag_shaders: dict
34    _programs: dict
35    _vaos: dict
36    _textures: dict
37    _shader_passes: dict
38    framebuffers: dict
39
40    1 import pygame
41    2 from data.constants import ShaderType
42    3 from data.shaders.protocol import SMPProtocol
43
44    4
45    5 class Base:
46    6     def __init__(self, shader_manager: SMPProtocol):
47    7         self._shader_manager = shader_manager
48
49    8
50    9         self._shader_manager.create_framebuffer(ShaderType.BASE)
51    10        self._shader_manager.create_vao(ShaderType.BACKGROUND_WAVES)
52    11        self._shader_manager.create_vao(ShaderType.BACKGROUND_BALATRO)
53    12        self._shader_manager.create_vao(ShaderType.BACKGROUND_LASERS)
54    13        self._shader_manager.create_vao(ShaderType.BACKGROUND_GRADIENT)
55    14        self._shader_manager.create_vao(ShaderType.BACKGROUND_NONE)
56
57    15
58    16    def apply(self, texture, background_type=None):
59    17        base_texture = self._shader_manager.get_fbo_texture(ShaderType.BASE)
60
61    18
62    19        match background_type:
63    20            case ShaderType.BACKGROUND_WAVES:
64    21                self._shader_manager.render_to_fbo(
65    22                    ShaderType.BASE,
66    23                    texture=base_texture,
67    24                    program_type=ShaderType.BACKGROUND_WAVES,
68    25                    use_image=False,
69    26                    time=pygame.time.get_ticks() / 1000
70    27                )
71    28            case ShaderType.BACKGROUND_BALATRO:
72    29                self._shader_manager.render_to_fbo(
73    30                    ShaderType.BASE,

```

```

31         texture=base_texture,
32         program_type=ShaderType.BACKGROUND_BALATRO,
33         use_image=False,
34         time=pygame.time.get_ticks() / 1000,
35         screenSize=base_texture.size
36     )
37     case ShaderType.BACKGROUND_LASERS:
38         self._shader_manager.render_to_fbo(
39             ShaderType.BASE,
40             texture=base_texture,
41             program_type=ShaderType.BACKGROUND_LASERS,
42             use_image=False,
43             time=pygame.time.get_ticks() / 1000,
44             screenSize=base_texture.size
45         )
46     case ShaderType.BACKGROUND_GRADIENT:
47         self._shader_manager.render_to_fbo(
48             ShaderType.BASE,
49             texture=base_texture,
50             program_type=ShaderType.BACKGROUND_GRADIENT,
51             use_image=False,
52             time=pygame.time.get_ticks() / 1000,
53             screenSize=base_texture.size
54         )
55     case None:
56         self._shader_manager.render_to_fbo(
57             ShaderType.BASE,
58             texture=base_texture,
59             program_type=ShaderType.BACKGROUND_NONE,
60             use_image=False,
61         )
62     case _:
63         raise ValueError('(shader.py) Unknown background type:',
background_type)
64
65     self._shader_manager.get_fbo_texture(ShaderType.BASE).use(1)
66     self._shader_manager.render_to_fbo(ShaderType.BASE, texture, background=1)

1 import moderngl
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMPProtocol
4
5 class _Blend:
6     def __init__(self, shader_manager: SMPProtocol):
7         self._shader_manager = shader_manager
8
9         self._shader_manager.create_framebuffer(ShaderType._BLEND)
10
11     def apply(self, texture, texture_2, texture_2_pos):
12         self._shader_manager._ctx.blend_func = (moderngl.SRC_ALPHA, moderngl.ONE)
13
14         relative_size = (texture_2.size[0] / texture.size[0], texture_2.size[1] /
texture.size[1])
15         opengl_pos = (texture_2_pos[0], 1 - texture_2_pos[1] - relative_size[1])
16
17         texture_2.use(1)
18         self._shader_manager.render_to_fbo(ShaderType._BLEND, texture, image2=1,
image2Pos=opengl_pos, relativeSize=relative_size)
19         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 SMPProtocol
4 from data.shaders.classes.blur import _Blur
5 from data.constants import ShaderType
6
7 BLOOM_INTENSITY = 0.6
8
9 class Bloom:
10     def __init__(self, shader_manager: SMPProtocol):
11         self._shader_manager = shader_manager
12
13         shader_manager.load_shader(ShaderType._BLUR)
14         shader_manager.load_shader(ShaderType._HIGHLIGHT_BRIGHTNESS)
15         shader_manager.load_shader(ShaderType._HIGHLIGHT_COLOUR)
16
17         shader_manager.create_framebuffer(ShaderType.BLOOM)
18         shader_manager.create_framebuffer(ShaderType._BLUR)
19         shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_BRIGHTNESS)
20         shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_COLOUR)
21
22     def apply(self, texture, highlight_surface=None, highlight_colours=[],
23             surface_intensity=BLOOM_INTENSITY, brightness_intensity=BLOOM_INTENSITY,
24             colour_intensity=BLOOM_INTENSITY):
25         """
26         Applies a bloom effect to a given texture.
27
28         Args:
29             texture (moderngl.Texture): Texture to apply bloom to.
30             highlight_surface (pygame.Surface, optional): Surface to use as the
31             highlights. Defaults to None.
32             highlight_colours (list[list[int, int, int], ...], optional): Colours
33             to use as the highlights. Defaults to [].
34             surface_intensity (_type_, optional): Intensity of bloom applied to
35             the highlight surface. Defaults to BLOOM_INTENSITY.
36             brightness_intensity (_type_, optional): Intensity of bloom applied to
37             the highlight brightness. Defaults to BLOOM_INTENSITY.
38             colour_intensity (_type_, optional): Intensity of bloom applied to the
39             highlight colours. Defaults to BLOOM_INTENSITY.
40         """
41         if highlight_surface:
42             # Calibrate Pygame surface and apply blur
43             glare_texture = self._shader_manager.calibrate_pygame_surface(
44                 highlight_surface)
45             _Blur(self._shader_manager).apply(glare_texture)
46
47             self._shader_manager.get_fbo_texture(ShaderType._BLUR).use(1)
48             self._shader_manager.render_to_fbo(ShaderType.BLOOM, texture,
49                 blurred_image=1, intensity=surface_intensity)
50
51             # Set bloom-applied texture as the base texture
52             texture = self._shader_manager.get_fbo_texture(ShaderType.BLOOM)
53
54             # Extract bright colours (highlights) from the texture
55             _HighlightBrightness(self._shader_manager).apply(texture, intensity=
56                 brightness_intensity)
57             highlight_texture = self._shader_manager.get_fbo_texture(ShaderType.
58                 _HIGHLIGHT_BRIGHTNESS)
59
60             # Use colour as highlights
61             for colour in highlight_colours:
62                 _HighlightColour(self._shader_manager).apply(texture, old_highlight=
63                     highlight_texture, colour=colour, intensity=colour_intensity)
64                 highlight_texture = self._shader_manager.get_fbo_texture(ShaderType.

```

```

        _HIGHLIGHT_COLOUR)

53
54     # Apply Gaussian blur to highlights
55     _Blur(self._shader_manager).apply(highlight_texture)
56
57     # Add the pixel values for the highlights onto the base texture
58     self._shader_manager.get_fbo_texture(ShaderType._BLUR).use(1)
59     self._shader_manager.render_to_fbo(ShaderType.BLOOM, texture, blurredImage
=1, intensity=BLOOM_INTENSITY)

1 from data.shaders.protocol import SMPProtocol
2 from data.constants import ShaderType
3
4 BLUR_ITERATIONS = 4
5
6 class _Blur:
7     def __init__(self, shader_manager: SMPProtocol):
8         self._shader_manager = shader_manager
9
10        shader_manager.create_framebuffer(ShaderType._BLUR)
11
12        shader_manager.create_framebuffer("blurPing")
13        shader_manager.create_framebuffer("blurPong")
14
15    def apply(self, texture):
16        """
17        Applies Gaussian blur to a given texture.
18
19        Args:
20            texture (moderngl.Texture): Texture to blur.
21        """
22        self._shader_manager.get_fbo_texture("blurPong").write(texture.read())
23
24        for _ in range(BLUR_ITERATIONS):
25            # Apply horizontal blur
26            self._shader_manager.render_to_fbo(
27                ShaderType._BLUR,
28                texture=self._shader_manager.get_fbo_texture("blurPong"),
29                output_fbo=self._shader_manager.framebuffers["blurPing"],
30                passes=5,
31                horizontal=True
32            )
33            # Apply vertical blur
34            self._shader_manager.render_to_fbo(
35                ShaderType._BLUR,
36                texture=self._shader_manager.get_fbo_texture("blurPing"), # Use
horizontal blur result as input texture
37                output_fbo=self._shader_manager.framebuffers["blurPong"],
38                passes=5,
39                horizontal=False
40            )
41
42            self._shader_manager.render_to_fbo(ShaderType._BLUR, self._shader_manager.
get_fbo_texture("blurPong"))

1 import pygame
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMPProtocol
4
5 CHROMATIC_ABBREVIATION_INTENSITY = 2.0
6
7 class ChromaticAbbreviation:

```

```

8     def __init__(self, shader_manager: SMPProtocol):
9         self._shader_manager = shader_manager
10
11         self._shader_manager.create_framebuffer(ShaderType.CHROMATIC_ABBREVIATION)
12
13     def apply(self, texture):
14         mouse_pos = (pygame.mouse.get_pos()[0] / texture.size[0], pygame.mouse.
15 get_pos()[1] / texture.size[1])
16         self._shader_manager.render_to_fbo(ShaderType.CHROMATIC_ABBREVIATION,
17 texture, mouseFocusPoint=mouse_pos, enabled=pygame.mouse.get_pressed()[0],
18 intensity=CHROMATIC_ABBREVIATION_INTENSITY)
19
20 from data.constants import ShaderType
21 from data.shaders.protocol import SMPProtocol
22
23 class _Crop:
24     def __init__(self, shader_manager: SMPProtocol):
25         self._shader_manager = shader_manager
26
27     def apply(self, texture, relative_pos, relative_size):
28         opengl_pos = (relative_pos[0], 1 - relative_pos[1] - relative_size[1])
29         pixel_size = (int(relative_size[0] * texture.size[0]), int(relative_size
30 [1] * texture.size[1]))
31
32         self._shader_manager.create_framebuffer(ShaderType._CROP, size=pixel_size)
33
34         self._shader_manager.render_to_fbo(ShaderType._CROP, texture, relativePos=
35 opengl_pos, relativeSize=relative_size)
36
37 from data.constants import ShaderType
38 from data.shaders.protocol import SMPProtocol
39
40 class CRT:
41     def __init__(self, shader_manager: SMPProtocol):
42         self._shader_manager = shader_manager
43
44         shader_manager.create_framebuffer(ShaderType.CRT)
45
46     def apply(self, texture):
47         self._shader_manager.render_to_fbo(ShaderType.CRT, texture)
48
49 from data.constants import ShaderType
50 from data.shaders.protocol import SMPProtocol
51
52 class Grayscale:
53     def __init__(self, shader_manager: SMPProtocol):
54         self._shader_manager = shader_manager
55
56         shader_manager.create_framebuffer(ShaderType.GRAYSCALE)
57
58     def apply(self, texture):
59         self._shader_manager.render_to_fbo(ShaderType.GRAYSCALE, texture)
60
61 from data.constants import ShaderType
62 from data.shaders.protocol import SMPProtocol
63
64 HIGHLIGHT_THRESHOLD = 0.9
65
66 class _HighlightBrightness:
67     def __init__(self, shader_manager: SMPProtocol):
68         self._shader_manager = shader_manager

```

```

9
10         shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_BRIGHTNESS)
11
12     def apply(self, texture, intensity):
13         self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_BRIGHTNESS,
        texture, threshold=HIGHLIGHT_THRESHOLD, intensity=intensity)

1  from data.constants import ShaderType
2  from data.shaders.protocol import SMPProtocol
3
4  class _HighlightColour:
5      def __init__(self, shader_manager: SMPProtocol):
6          self._shader_manager = shader_manager
7
8          shader_manager.create_framebuffer(ShaderType._HIGHLIGHT_COLOUR)
9
10     def apply(self, texture, old_highlight, colour, intensity):
11         old_highlight.use(1)
12         self._shader_manager.render_to_fbo(ShaderType._HIGHLIGHT_COLOUR, texture,
        highlight=1, colour=colour, threshold=0.1, intensity=intensity)

1  from data.constants import ShaderType
2  from data.shaders.protocol import SMPProtocol
3  from data.shaders.classes.shadowmap import _Shadowmap
4
5  LIGHT_RESOLUTION = 256
6
7  class _Lightmap:
8      def __init__(self, shader_manager: SMPProtocol):
9          self._shader_manager = shader_manager
10
11         shader_manager.load_shader(ShaderType._SHADOWMAP)
12
13     def apply(self, texture, colour, softShadow, occlusion=None, falloff=0.0,
        clamp=(-180, 180)):
14         self._shader_manager.create_framebuffer(ShaderType._LIGHTMAP, size=texture
        .size)
15         self._shader_manager._ctx.enable(self._shader_manager._ctx.BLEND)
16
17         _Shadowmap(self._shader_manager).apply(texture, occlusion)
18         shadow_map = self._shader_manager.get_fbo_texture(ShaderType._SHADOWMAP)
19
20         self._shader_manager.render_to_fbo(ShaderType._LIGHTMAP, shadow_map,
        resolution=LIGHT_RESOLUTION, lightColour=colour, falloff=falloff, angleClamp=
        clamp, softShadow=softShadow)
21
22         self._shader_manager._ctx.disable(self._shader_manager._ctx.BLEND)

1  from data.constants import ShaderType
2  from data.shaders.protocol import SMPProtocol
3
4  class _Occlusion:
5      def __init__(self, shader_manager: SMPProtocol):
6          self._shader_manager = shader_manager
7
8     def apply(self, texture, occlusion_colour=(255, 0, 0)):
9         self._shader_manager.create_framebuffer(ShaderType._OCCLUSION, size=
        texture.size)
10         self._shader_manager.render_to_fbo(ShaderType._OCCLUSION, texture,
        checkColour=tuple(num / 255 for num in occlusion_colour))

```

```

1 from data.shaders.classes.lightmap import _Lightmap
2 from data.shaders.classes.blend import _Blend
3 from data.shaders.protocol import SMPProtocol
4 from data.shaders.classes.crop import _Crop
5 from data.constants import ShaderType
6
7 class Rays:
8     def __init__(self, shader_manager: SMPProtocol, lights):
9         self._shader_manager = shader_manager
10        self._lights = lights
11
12        # Load all necessary shaders
13        shader_manager.load_shader(ShaderType._LIGHTMAP)
14        shader_manager.load_shader(ShaderType._BLEND)
15        shader_manager.load_shader(ShaderType._CROP)
16        shader_manager.create_framebuffer(ShaderType.RAYS)
17
18    def apply(self, texture, occlusion=None, softShadow=0.3):
19        """
20        Applies the light rays effect to a given texture.
21
22        Args:
23            texture (modernogl.Texture): The texture to apply the effect to.
24            occlusion (pygame.Surface, optional): A Pygame mask surface to use as
the occlusion texture. Defaults to None.
25        """
26        final_texture = texture
27
28        # Iterate through array containing light information
29        for pos, radius, colour, *args in self._lights:
30            # Topleft of light source square
31            light_topleft = (pos[0] - (radius * texture.size[1] / texture.size[0])
, pos[1] - radius)
32            # Relative size of light compared to texture
33            relative_size = (radius * 2 * texture.size[1] / texture.size[0],
radius * 2)
34
35            # Crop texture to light source diameter, and to position light source
at the center
36            _Crop(self._shader_manager).apply(texture, relative_pos=light_topleft,
relative_size=relative_size)
37            cropped_texture = self._shader_manager.get_fbo_texture(ShaderType.
_CROP)
38
39            if occlusion:
40                # Calibrate Pygame mask surface and crop it
41                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)
43                occlusion_texture = self._shader_manager.get_fbo_texture(
ShaderType._CROP)
44            else:
45                occlusion_texture = None
46
47            # Apply lightmap shader, shadowmap and occlusion are included within
the _Lightmap class
48            _Lightmap(self._shader_manager).apply(cropped_texture, colour,
softShadow, occlusion_texture, *args)
49            light_map = self._shader_manager.get_fbo_texture(ShaderType._LIGHTMAP)
50
51            # Blend the final result with the original texture

```

```

52         _Blend(self._shader_manager).apply(final_texture, light_map,
light_topleft)
53         final_texture = self._shader_manager.get_fbo_texture(ShaderType._BLEND
)
54
55         self._shader_manager.render_to_fbo(ShaderType.RAYS, final_texture)

1 import moderngl
2 from data.constants import ShaderType
3 from data.shaders.protocol import SMPProtocol
4 from data.shaders.classes.occlusion import _Occlusion
5
6 LIGHT_RESOLUTION = 256
7
8 class _Shadowmap:
9     def __init__(self, shader_manager: SMPProtocol):
10         self._shader_manager = shader_manager
11
12         shader_manager.load_shader(ShaderType._OCCLUSION)
13
14     def apply(self, texture, occlusion_texture=None):
15         self._shader_manager.create_framebuffer(ShaderType._SHADOWMAP, size=(
texture.size[0], 1), filter=moderngl.LINEAR)
16
17         if occlusion_texture is None:
18             _Occlusion(self._shader_manager).apply(texture)
19             occlusion_texture = self._shader_manager.get_fbo_texture(ShaderType.
_OCCLUSION)
20
21         self._shader_manager.render_to_fbo(ShaderType._SHADOWMAP,
occlusion_texture, resolution=LIGHT_RESOLUTION)

1 from data.constants import ShaderType
2 from data.shaders.protocol import SMPProtocol
3 from random import randint
4
5 SHAKE_INTENSITY = 3
6
7 class Shake:
8     def __init__(self, shader_manager: SMPProtocol):
9         self._shader_manager = shader_manager
10
11         self._shader_manager.create_framebuffer(ShaderType.SHAKE)
12
13     def apply(self, texture, intensity=SHAKE_INTENSITY):
14         displacement = (randint(-intensity, intensity) / 1000, randint(-intensity,
intensity) / 1000)
15         self._shader_manager.render_to_fbo(ShaderType.SHAKE, texture, displacement
=displacement)

1 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
17
18 shader_pass_lookup = {
19     ShaderType.CHROMATIC_ABBREVIATION: ChromaticAbbreviation,
20     ShaderType.GRAYSCALE: Grayscale,
21     ShaderType.SHAKE: Shake,
22     ShaderType.BLOOM: Bloom,
23     ShaderType.BASE: Base,
24     ShaderType.RAYS: Rays,
25     ShaderType.CRT: CRT,
26
27     ShaderType._HIGHLIGHT_BRIGHTNESS: _HighlightBrightness,
28     ShaderType._HIGHLIGHT_COLOUR: _HighlightColour,
29     ShaderType._CALIBRATE: lambda *args: None,
30     ShaderType._OCCLUSION: _Occlusion,
31     ShaderType._SHADOWMAP: _Shadowmap,
32     ShaderType._LIGHTMAP: _Lightmap,
33     ShaderType._BLEND: _Blend,
34     ShaderType._BLUR: _Blur,
35     ShaderType._CROP: _Crop,
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
11
12 logger = initialise_logger(__name__)
13
14 class Browser(_State):
15     def __init__(self):
16         super().__init__()
17
18         self._selected_index = None
19         self._filter_column = 'number_of_ply'
20         self._filter_ascend = False
21         self._games_list = []
22         self._page_number = 1
23
24     def cleanup(self):
25         super().cleanup()
26
27         if self._selected_index is not None:
28             return self._games_list[self._selected_index]
29
30         return None
31
32     def startup(self, persist=None):
33         self.refresh_games_list() # BEFORE RESIZE TO FILL WIDGET BEFORE RESIZING
34         super().startup(BROWSER_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
35

```

```

36         self._filter_column = 'number_of_ply'
37         self._filter_ascend = False
38
39         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
40
41         BROWSER_WIDGETS['help'].kill()
42         BROWSER_WIDGETS['browser_strip'].kill()
43
44         self.draw()
45
46     def refresh_games_list(self):
47         column_map = {
48             'moves': 'number_of_ply',
49             'winner': 'winner',
50             'time': 'created_dt'
51         }
52
53         ascend_map = {
54             'asc': True,
55             'desc': False
56         }
57
58         filter_column = BROWSER_WIDGETS['filter_column_dropdown'].
get_selected_word()
59         filter_ascend = BROWSER_WIDGETS['filter_ascend_dropdown'].
get_selected_word()
60
61         self._selected_index = None
62
63         start_row = (self._page_number - 1) * GAMES_PER_PAGE + 1
64         end_row = (self._page_number) * GAMES_PER_PAGE
65         self._games_list = get_ordered_games(column_map[filter_column], ascend_map
[filter_ascend], start_row=start_row, end_row=end_row)
66
67         BROWSER_WIDGETS['browser_strip'].initialise_games_list(self._games_list)
68         BROWSER_WIDGETS['browser_strip'].set_surface_size(window.size)
69         BROWSER_WIDGETS['scroll_area'].set_image()
70
71     def get_event(self, event):
72         widget_event = self._widget_group.process_event(event)
73
74         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
75             BROWSER_WIDGETS['help'].kill()
76
77         if widget_event is None:
78             return
79
80         match widget_event.type:
81             case BrowserEventType.MENU_CLICK:
82                 self.next = 'menu'
83                 self.done = True
84
85             case BrowserEventType.BROWSER_STRIP_CLICK:
86                 self._selected_index = widget_event.selected_index
87
88             case BrowserEventType.COPY_CLICK:
89                 if self._selected_index is None:
90                     return
91                 logger.info('COPYING TO CLIPBOARD:', self._games_list[self.
_selected_index]['fen_string'])
92                 pyperclip.copy(self._games_list[self._selected_index]['fen_string']

```

```

    ])

93
94         case BrowserEventType.DELETE_CLICK:
95             if self._selected_index is None:
96                 return
97             delete_game(self._games_list[self._selected_index]['id'])
98             self.refresh_games_list()
99
100        case BrowserEventType.REVIEW_CLICK:
101            if self._selected_index is None:
102                return
103
104            self.next = 'review'
105            self.done = True
106
107        case BrowserEventType.FILTER_COLUMN_CLICK:
108            selected_word = BROWSER_WIDGETS['filter_column_dropdown'].
get_selected_word()
109
110            if selected_word is None:
111                return
112
113            self.refresh_games_list()
114
115        case BrowserEventType.FILTER_ASCEND_CLICK:
116            selected_word = BROWSER_WIDGETS['filter_ascend_dropdown'].
get_selected_word()
117
118            if selected_word is None:
119                return
120
121            self.refresh_games_list()
122
123        case BrowserEventType.PAGE_CLICK:
124            self._page_number = widget_event.data
125
126            self.refresh_games_list()
127
128        case BrowserEventType.HELP_CLICK:
129            self._widget_group.add(BROWSER_WIDGETS['help'])
130            self._widget_group.handle_resize(window.size)
131
132        def draw(self):
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
6
7  BROWSER_HEIGHT = 0.6
8
9  browser_strip = BrowserStrip(
10     relative_position=(0.0, 0.0),
11     relative_height=BROWSER_HEIGHT,
12     games_list=[]
13 )
14
15 number_of_pages = get_number_of_games() // GAMES_PER_PAGE + 1
16
17 carousel_widgets = {

```

```

18     i: Text(
19         relative_position=(0, 0),
20         relative_size=(0.3, 0.1),
21         text=f"PAGE {i} OF {number_of_pages}",
22         fill_colour=(0, 0, 0, 0),
23         fit_vertical=False,
24         border_width=0,
25     )
26     for i in range(1, number_of_pages + 1)
27 }
28
29 sort_by_container = Rectangle(
30     relative_size=(0.5, 0.1),
31     relative_position=(0.01, 0.77),
32     anchor_x='right',
33     visible=True
34 )
35
36 buttons_container = Rectangle(
37     relative_position=(0, 0.025),
38     relative_size=(0.5, 0.1),
39     scale_mode='height',
40     anchor_x='center'
41 )
42
43 top_right_container = Rectangle(
44     relative_position=(0, 0),
45     relative_size=(0.15, 0.075),
46     fixed_position=(5, 5),
47     anchor_x='right',
48     scale_mode='height'
49 )
50
51 BROWSER_WIDGETS = {
52     'help':
53     Icon(
54         relative_position=(0, 0),
55         relative_size=(1.02, 1.02),
56         icon=GRAPHICS['browser_help'],
57         anchor_x='center',
58         anchor_y='center',
59         border_width=0,
60         fill_colour=(0, 0, 0, 0)
61     ),
62     'default': [
63         buttons_container,
64         sort_by_container,
65         top_right_container,
66         ReactiveIconButton(
67             parent=top_right_container,
68             relative_position=(0, 0),
69             relative_size=(1, 1),
70             anchor_x='right',
71             scale_mode='height',
72             base_icon=GRAPHICS['home_base'],
73             hover_icon=GRAPHICS['home_hover'],
74             press_icon=GRAPHICS['home_press'],
75             event=CustomEvent(BrowserEventType.MENU_CLICK)
76         ),
77         ReactiveIconButton(
78             parent=top_right_container,
79             relative_position=(0, 0),

```

```

80         relative_size=(1, 1),
81         scale_mode='height',
82         base_icon=GRAPHICS['help_base'],
83         hover_icon=GRAPHICS['help_hover'],
84         press_icon=GRAPHICS['help_press'],
85         event=CustomEvent(BrowserEventType.HELP_CLICK)
86     ),
87     ReactiveIconButton(
88         parent=buttons_container,
89         relative_position=(0, 0),
90         relative_size=(1, 1),
91         scale_mode='height',
92         base_icon=GRAPHICS['copy_base'],
93         hover_icon=GRAPHICS['copy_hover'],
94         press_icon=GRAPHICS['copy_press'],
95         event=CustomEvent(BrowserEventType.COPY_CLICK),
96     ),
97     ReactiveIconButton(
98         parent=buttons_container,
99         relative_position=(0, 0),
100         relative_size=(1, 1),
101         scale_mode='height',
102         anchor_x='center',
103         base_icon=GRAPHICS['delete_base'],
104         hover_icon=GRAPHICS['delete_hover'],
105         press_icon=GRAPHICS['delete_press'],
106         event=CustomEvent(BrowserEventType.DELETE_CLICK),
107     ),
108     ReactiveIconButton(
109         parent=buttons_container,
110         relative_position=(0, 0),
111         relative_size=(1, 1),
112         scale_mode='height',
113         anchor_x='right',
114         base_icon=GRAPHICS['review_base'],
115         hover_icon=GRAPHICS['review_hover'],
116         press_icon=GRAPHICS['review_press'],
117         event=CustomEvent(BrowserEventType.REVIEW_CLICK),
118     ),
119     Text(
120         parent=sort_by_container,
121         relative_position=(0, 0),
122         relative_size=(0.3, 1),
123         fit_vertical=False,
124         text='SORT BY:',
125         border_width=0,
126         fill_colour=(0, 0, 0, 0)
127     )
128 ],
129 'browser_strip':
130     browser_strip,
131 'scroll_area':
132     ScrollArea(
133         relative_position=(0.0, 0.15),
134         relative_size=(1, BROWSER_HEIGHT),
135         vertical=False,
136         widget=browser_strip
137     ),
138 'filter_column_dropdown':
139     Dropdown(
140         parent=sort_by_container,
141         relative_position=(0.3, 0),

```

```

142         relative_height=0.75,
143         anchor_x='right',
144         word_list=['time', 'moves', 'winner'],
145         fill_colour=(255, 100, 100),
146         event=CustomEvent(BrowserEventType.FILTER_COLUMN_CLICK)
147     ),
148     'filter_ascend_dropdown':
149     Dropdown(
150         parent=sort_by_container,
151         relative_position=(0, 0),
152         relative_height=0.75,
153         anchor_x='right',
154         word_list=['desc', 'asc'],
155         fill_colour=(255, 100, 100),
156         event=CustomEvent(BrowserEventType.FILTER_ASCEND_CLICK)
157     ),
158     'page_carousel':
159     Carousel(
160         relative_position=(0.01, 0.77),
161         margin=5,
162         widgets_dict=carousel_widgets,
163         event=CustomEvent(BrowserEventType.PAGE_CLICK),
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
13
14 logger = initialise_logger(__name__)
15
16 class Config(_State):
17     def __init__(self):
18         super().__init__()
19
20         self._config = None
21         self._valid_fen = True
22         self._selected_preset = None
23
24     def cleanup(self):
25         super().cleanup()
26
27         window.clear_apply_arguments(ShaderType.BLOOM)
28
29         return self._config
30
31     def startup(self, persist=None):
32         super().startup(CONFIG_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
33         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
34             Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.9)
35
36         CONFIG_WIDGETS['invalid_fen_string'].kill()
37         CONFIG_WIDGETS['help'].kill()

```

```

37
38     self._config = default_config
39
40     if persist:
41         self._config['FEN_STRING'] = persist
42
43     self.set_fen_string(self._config['FEN_STRING'])
44     self.toggle_pvc(self._config['CPU_ENABLED'])
45     self.set_active_colour(self._config['COLOUR'])
46
47     CONFIG_WIDGETS['cpu_depth_carousel'].set_to_key(self._config['CPU_DEPTH'])
48     if self._config['CPU_ENABLED']:
49         self.create_depth_picker()
50     else:
51         self.remove_depth_picker()
52
53     self.draw()
54
55     def create_depth_picker(self):
56         # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.8))
57         # CONFIG_WIDGETS['start_button'].set_image()
58         CONFIG_WIDGETS['cpu_depth_carousel'].set_surface_size(window.size)
59         CONFIG_WIDGETS['cpu_depth_carousel'].set_image()
60         CONFIG_WIDGETS['cpu_depth_carousel'].set_geometry()
61         self._widget_group.add(CONFIG_WIDGETS['cpu_depth_carousel'])
62
63     def remove_depth_picker(self):
64         # CONFIG_WIDGETS['start_button'].update_relative_position((0.5, 0.7))
65         # CONFIG_WIDGETS['start_button'].set_image()
66
67         CONFIG_WIDGETS['cpu_depth_carousel'].kill()
68
69     def toggle_pvc(self, pvc_enabled):
70         if pvc_enabled:
71             CONFIG_WIDGETS['pvc_button'].set_locked(True)
72             CONFIG_WIDGETS['pvp_button'].set_locked(False)
73         else:
74             CONFIG_WIDGETS['pvp_button'].set_locked(True)
75             CONFIG_WIDGETS['pvc_button'].set_locked(False)
76
77         self._config['CPU_ENABLED'] = pvc_enabled
78
79         if self._config['CPU_ENABLED']:
80             self.create_depth_picker()
81         else:
82             self.remove_depth_picker()
83
84     def set_fen_string(self, new_fen_string):
85         CONFIG_WIDGETS['fen_string_input'].set_text(new_fen_string)
86         self._config['FEN_STRING'] = new_fen_string
87
88         self.set_preset_overlay(new_fen_string)
89
90         try:
91             CONFIG_WIDGETS['board_thumbnail'].initialise_board(new_fen_string)
92             CONFIG_WIDGETS['invalid_fen_string'].kill()
93
94             if new_fen_string[-1].lower() == 'r':
95                 self.set_active_colour(Colour.RED)
96             else:
97                 self.set_active_colour(Colour.BLUE)
98

```

```

99         self._valid_fen = True
100     except:
101         CONFIG_WIDGETS['board_thumbnail'].initialise_board('')
102         self._widget_group.add(CONFIG_WIDGETS['invalid_fen_string'])
103
104         window.set_effect(ShaderType.SHAKE)
105         animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE))
106
107     )
108
109     audio.play_sfx(SFX['error_1'])
110     audio.play_sfx(SFX['error_2'])
111
112     self._valid_fen = False
113
114 def get_event(self, event):
115     widget_event = self._widget_group.process_event(event)
116
117     if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
118         CONFIG_WIDGETS['help'].kill()
119
120     if widget_event is None:
121         return
122
123     match widget_event.type:
124     case ConfigEventType.GAME_CLICK:
125         if self._valid_fen:
126             self.next = 'game'
127             self.done = True
128
129     case ConfigEventType.MENU_CLICK:
130         self.next = 'menu'
131         self.done = True
132
133     case ConfigEventType.TIME_CLICK:
134         self._config['TIME_ENABLED'] = not(widget_event.data)
135         CONFIG_WIDGETS['timer_button'].set_next_icon()
136
137     case ConfigEventType.PVP_CLICK:
138         self.toggle_pvc(False)
139
140     case ConfigEventType.PVC_CLICK:
141         self.toggle_pvc(True)
142
143     case ConfigEventType.FEN_STRING_TYPE:
144         self.set_fen_string(widget_event.text)
145
146     case ConfigEventType.TIME_TYPE:
147         if widget_event.text == '':
148             self._config['TIME'] = 5
149         else:
150             self._config['TIME'] = float(widget_event.text)
151
152     case ConfigEventType.CPU_DEPTH_CLICK:
153         self._config['CPU_DEPTH'] = int(widget_event.data)
154
155     case ConfigEventType.PRESET_CLICK:
156         self.set_fen_string(widget_event.fen_string)
157
158     case ConfigEventType.SETUP_CLICK:
159         self.next = 'editor'
160         self.done = True

```



```

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

```

```

1 from data.constants import Colour
2
3 default_config = {
4     'CPU_ENABLED': False,
5     'CPU_DEPTH': 2,
6     'FEN_STRING': 'sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7
Pa2/2PdNaFaNa3Sa b',
7     'TIME_ENABLED': True,
8     'TIME': 5,
9     '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
7 from data.utils.asset_helpers import get_highlighted_icon
8 from data.managers.theme import theme
9
10 def float_validator(num_string):
11     try:
12         float(num_string)
13         return True
14     except:
15         return False
16
17 if default_config['CPU_ENABLED']:
18     pvp_icons = {False: GRAPHICS['swords'], True: GRAPHICS['swords']}
19     pvc_icons = {True: GRAPHICS['robot'], False: GRAPHICS['robot']}
20     pvc_locked = True
21     pvp_locked = False
22 else:
23     pvp_icons = {True: GRAPHICS['swords'], False: GRAPHICS['swords']}
24     pvc_icons = {False: GRAPHICS['robot'], True: GRAPHICS['robot']}
25     pvc_locked = False
26     pvp_locked = True
27
28 if default_config['TIME_ENABLED']:
29     time_enabled_icons = {True: GRAPHICS['timer'], False: get_highlighted_icon(
30         GRAPHICS['timer'])}
31 else:
32     time_enabled_icons = {False: get_highlighted_icon(GRAPHICS['timer']), True:
33         GRAPHICS['timer']}
34
35 if default_config['COLOUR'] == Colour.BLUE:
36     colour_icons = {Colour.BLUE: GRAPHICS['pharaoh_0_a'], Colour.RED: GRAPHICS['
37         pharaoh_1_a']}
38 else:
39     colour_icons = {Colour.RED: GRAPHICS['pharaoh_1_a'], Colour.BLUE: GRAPHICS['
40         pharaoh_0_a']}
41
42 preview_container = Rectangle(
43     relative_position=(-0.15, 0),
44     relative_size=(0.65, 0.9),
45     anchor_x='center',
46     anchor_y='center',
47 )
48
49 config_container = Rectangle(
50     relative_position=(0.325, 0),
51     relative_size=(0.3, 0.9),
52     anchor_x='center',
53     anchor_y='center',
54 )
55
56 to_move_container = Rectangle(
57     parent=config_container,
58     relative_size=(0.9, 0.15),
59     relative_position=(0, 0.1),
60     anchor_x='center'

```

```

57 )
58
59 board_thumbnail = BoardThumbnail(
60     parent=preview_container,
61     relative_position=(0, 0),
62     relative_width=0.7,
63     scale_mode='width',
64     anchor_x='right',
65 )
66
67 top_right_container = Rectangle(
68     relative_position=(0, 0),
69     relative_size=(0.15, 0.075),
70     fixed_position=(5, 5),
71     anchor_x='right',
72     scale_mode='height'
73 )
74
75 CONFIG_WIDGETS = {
76     'help':
77         Icon(
78             relative_position=(0, 0),
79             relative_size=(1.02, 1.02),
80             icon=GRAPHICS['config_help'],
81             anchor_x='center',
82             anchor_y='center',
83             border_width=0,
84             fill_colour=(0, 0, 0, 0)
85         ),
86     'default': [
87         preview_container,
88         config_container,
89         to_move_container,
90         top_right_container,
91         ReactiveIconButton(
92             parent=top_right_container,
93             relative_position=(0, 0),
94             relative_size=(1, 1),
95             anchor_x='right',
96             scale_mode='height',
97             base_icon=GRAPHICS['home_base'],
98             hover_icon=GRAPHICS['home_hover'],
99             press_icon=GRAPHICS['home_press'],
100             event=CustomEvent(ConfigEventType.MENU_CLICK)
101         ),
102         ReactiveIconButton(
103             parent=top_right_container,
104             relative_position=(0, 0),
105             relative_size=(1, 1),
106             scale_mode='height',
107             base_icon=GRAPHICS['help_base'],
108             hover_icon=GRAPHICS['help_hover'],
109             press_icon=GRAPHICS['help_press'],
110             event=CustomEvent(ConfigEventType.HELP_CLICK)
111         ),
112         TextInput(
113             parent=config_container,
114             relative_position=(0.3, 0.3),
115             relative_size=(0.65, 0.15),
116             fit_vertical=True,
117             placeholder='TIME CONTROL (DEFAULT 5)',
118             default=str(default_config['TIME']),

```

```

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

```

```

180         margin=10,
181         border_width=5,
182         border_radius=5,
183         icons_dict=time_enabled_icons,
184         event=CustomEvent(ConfigEventType.TIME_CLICK)
185     ),
186     'pvp_button':
187     MultipleIconButton(
188         parent=config_container,
189         relative_position=(-0.225, 0.5),
190         relative_size=(0.45, 0.15),
191         margin=15,
192         anchor_x='center',
193         icons_dict=pvp_icons,
194         stretch=False,
195         event=CustomEvent(ConfigEventType.PVP_CLICK)
196     ),
197     'pvc_button':
198     MultipleIconButton(
199         parent=config_container,
200         relative_position=(0.225, 0.5),
201         relative_size=(0.45, 0.15),
202         anchor_x='center',
203         margin=15,
204         icons_dict=pvc_icons,
205         stretch=False,
206         event=CustomEvent(ConfigEventType.PVC_CLICK)
207     ),
208     'invalid_fen_string':
209     Text(
210         parent=board_thumbnail,
211         relative_position=(0, 0),
212         relative_size=(0.9, 0.1),
213         fit_vertical=False,
214         anchor_x='center',
215         anchor_y='center',
216         text='INVALID FEN STRING!',
217         margin=10,
218         fill_colour=theme['fillError'],
219         text_colour=theme['textError'],
220     ),
221     'preset_1':
222     BoardThumbnailButton(
223         parent=preview_container,
224         relative_width=0.25,
225         relative_position=(0, 0),
226         scale_mode='width',
227         fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3
228 /7Pa2/2PdNaFaNa3Sa b",
229         event=CustomEvent(ConfigEventType.PRESET_CLICK)
230     ),
231     'preset_2':
232     BoardThumbnailButton(
233         parent=preview_container,
234         relative_width=0.25,
235         relative_position=(0, 0.35),
236         scale_mode='width',
237         fen_string="sc3ncfcncra2/10/3Pd2pa3/paPc2Pbra2pbPd/pbPd2Rapd2paPc/3Pc2pb3
238 /10/2RaNaFaNa3Sa b",
239         event=CustomEvent(ConfigEventType.PRESET_CLICK)
240     ),
241     'preset_3':

```

```

240 BoardThumbnailButton(
241     parent=preview_container,
242     relative_width=0.25,
243     relative_position=(0, 0.7),
244     scale_mode='width',
245     fen_string="sc3pcncpb3/5fc4/pa3pcncra3/pb1rd1Pd1Pb3/3pd1pb1Rd1Pd/3
RaNaPa3Pc/4Fa5/3PdNaPa3Sa b",
246     event=CustomEvent(ConfigEventType.PRESET_CLICK)
247 ),
248 'to_move_button':
249 MultipleIconButton(
250     parent=to_move_container,
251     scale_mode='height',
252     relative_position=(0, 0),
253     relative_size=(1, 1),
254     icons_dict=colour_icons,
255     anchor_x='left',
256     event=CustomEvent(ConfigEventType.COLOUR_CLICK)
257 ),
258 'to_move_text':
259 Text(
260     parent=to_move_container,
261     relative_position=(0, 0),
262     relative_size=(0.75, 1),
263     fit_vertical=False,
264     text='TO MOVE',
265     anchor_x='right'
266 ),
267 'cpu_depth_carousel':
268 Carousel(
269     parent=config_container,
270     relative_position=(0, 0.65),
271     event=CustomEvent(ConfigEventType.CPU_DEPTH_CLICK),
272     anchor_x='center',
273     border_width=0,
274     fill_colour=(0, 0, 0, 0),
275     widgets_dict={
276         2: Text(
277             parent=config_container,
278             relative_position=(0, 0),
279             relative_size=(0.8, 0.075),
280             text="EASY",
281             margin=0,
282             border_width=0,
283             fill_colour=(0, 0, 0, 0)
284         ),
285         3: Text(
286             parent=config_container,
287             relative_position=(0, 0),
288             relative_size=(0.8, 0.075),
289             text="MEDIUM",
290             margin=0,
291             border_width=0,
292             fill_colour=(0, 0, 0, 0)
293         ),
294         4: Text(
295             parent=config_container,
296             relative_position=(0, 0),
297             relative_size=(0.8, 0.075),
298             text="HARD",
299             margin=0,
300             border_width=0,

```

```

301         fill_colour=(0, 0, 0, 0)
302     ),
303 }
304 )
305 }

1 import pygame
2 import pyperclip
3 from data.constants import EditorEventType, Colour, RotationDirection, Piece,
    Rotation
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
8 from data.states.game.components.father import DragAndDrop
9 from data.utils.bitboard_helpers import coords_to_bitboard
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
15
16 logger = initialise_logger(__name__)
17
18 class Editor(_State):
19     def __init__(self):
20         super().__init__()
21
22         self._bitboards = None
23         self._piece_group = None
24         self._selected_coords = None
25         self._selected_tool = None
26         self._selected_tool_colour = None
27         self._initial_fen_string = None
28         self._starting_colour = None
29
30         self._drag_and_drop = None
31         self._overlay_draw = None
32
33     def cleanup(self):
34         super().cleanup()
35
36         self.deselect_tool()
37
38         return encode_fen_string(self._bitboards)
39
40     def startup(self, persist):
41         super().startup(EDITOR_WIDGETS)
42         EDITOR_WIDGETS['help'].kill()
43
44         self._drag_and_drop = DragAndDrop(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
45         self._overlay_draw = OverlayDraw(EDITOR_WIDGETS['chessboard'].position,
EDITOR_WIDGETS['chessboard'].size)
46         self._bitboards = BitboardCollection(persist['FEN_STRING'])
47         self._piece_group = PieceGroup()
48
49         self._selected_coords = None
50         self._selected_tool = None
51         self._selected_tool_colour = None
52         self._initial_fen_string = persist['FEN_STRING']
53         self._starting_colour = Colour.BLUE

```

```

54         self.refresh_pieces()
55         self.set_starting_colour(Colour.BLUE if persist['FEN_STRING'][-1].lower()
56 == 'b' else Colour.RED)
57         self.draw()
58
59     @property
60     def selected_coords(self):
61         return self._selected_coords
62
63     @selected_coords.setter
64     def selected_coords(self, new_coords):
65         self._overlay_draw.set_selected_coords(new_coords)
66         self._selected_coords = new_coords
67
68     def get_event(self, event):
69         widget_event = self._widget_group.process_event(event)
70
71         if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
72             EDITOR_WIDGETS['help'].kill()
73
74         if event.type == pygame.MOUSEBUTTONDOWN:
75             clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
76
77             if clicked_coords:
78                 self.selected_coords = clicked_coords
79
80                 if self._selected_tool is None:
81                     return
82
83                 if self._selected_tool == 'MOVE':
84                     self.set_dragged_piece(clicked_coords)
85
86                 elif self._selected_tool == 'ERASE':
87                     self.remove_piece()
88                 else:
89                     self.set_piece(self._selected_tool, self._selected_tool_colour
, Rotation.UP)
90
91             return
92
93         if event.type == pygame.MOUSEBUTTONUP:
94             clicked_coords = screen_pos_to_coords(event.pos, EDITOR_WIDGETS['
chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
95
96             if self._drag_and_drop.dragged_sprite:
97                 self.remove_dragged_piece(clicked_coords)
98                 return
99
100             if widget_event is None:
101                 if event.type == pygame.MOUSEBUTTONDOWN and self._widget_group.
on_widget(event.pos) is False:
102                     self.selected_coords = None
103
104             return
105
106         match widget_event.type:
107             case None:
108                 return
109
110             case EditorEventType.MENU_CLICK:

```



```

111         self.next = 'menu'
112         self.done = True
113
114         case EditorEventType.PICK_PIECE_CLICK:
115             if widget_event.piece == self._selected_tool and widget_event.
active_colour == self._selected_tool_colour:
116                 self.deselect_tool()
117             else:
118                 self.select_tool(widget_event.piece, widget_event.
active_colour)
119
120         case EditorEventType.ROTATE_PIECE_CLICK:
121             self.rotate_piece(widget_event.rotation_direction)
122
123         case EditorEventType.EMPTY_CLICK:
124             self._bitboards = BitboardCollection(fen_string='sc9
/10/10/10/10/10/10/9Sa b')
125             self.refresh_pieces()
126
127         case EditorEventType.RESET_CLICK:
128             self.reset_board()
129
130         case EditorEventType.COPY_CLICK:
131             logger.info(f'COPYING TO CLIPBOARD: {encode_fen_string(self.
_bitboards)}')
132             pyperclip.copy(encode_fen_string(self._bitboards))
133
134         case EditorEventType.BLUE_START_CLICK:
135             self.set_starting_colour(Colour.BLUE)
136
137         case EditorEventType.RED_START_CLICK:
138             self.set_starting_colour(Colour.RED)
139
140         case EditorEventType.START_CLICK:
141             self.next = 'config'
142             self.done = True
143
144         case EditorEventType.CONFIG_CLICK:
145             self.reset_board()
146             self.next = 'config'
147             self.done = True
148
149         case EditorEventType.ERASE_CLICK:
150             if self._selected_tool == 'ERASE':
151                 self.deselect_tool()
152             else:
153                 self.select_tool('ERASE', None)
154
155         case EditorEventType.MOVE_CLICK:
156             if self._selected_tool == 'MOVE':
157                 self.deselect_tool()
158             else:
159                 self.select_tool('MOVE', None)
160
161         case EditorEventType.HELP_CLICK:
162             self._widget_group.add(EDITOR_WIDGETS['help'])
163             self._widget_group.handle_resize(window.size)
164
165     def reset_board(self):
166         self._bitboards = BitboardCollection(self._initial_fen_string)
167         self.refresh_pieces()
168

```

```

169     def refresh_pieces(self):
170         self._piece_group.initialise_pieces(self._bitboards.convert_to_piece_list
171         (), EDITOR_WIDGETS['chessboard'].position, EDITOR_WIDGETS['chessboard'].size)
172
173     def set_starting_colour(self, new_colour):
174         if new_colour == Colour.BLUE:
175             EDITOR_WIDGETS['blue_start_button'].set_locked(True)
176             EDITOR_WIDGETS['red_start_button'].set_locked(False)
177         elif new_colour == Colour.RED:
178             EDITOR_WIDGETS['blue_start_button'].set_locked(False)
179             EDITOR_WIDGETS['red_start_button'].set_locked(True)
180
181         if new_colour != self._starting_colour:
182             EDITOR_WIDGETS['blue_start_button'].set_next_icon()
183             EDITOR_WIDGETS['red_start_button'].set_next_icon()
184
185         self._starting_colour = new_colour
186         self._bitboards.active_colour = new_colour
187
188     def set_dragged_piece(self, coords):
189         bitboard_under_mouse = coords_to_bitboard(coords)
190         dragged_piece = self._bitboards.get_piece_on(bitboard_under_mouse, Colour.
191         BLUE) or self._bitboards.get_piece_on(bitboard_under_mouse, Colour.RED)
192
193         if dragged_piece is None:
194             return
195
196         dragged_colour = self._bitboards.get_colour_on(bitboard_under_mouse)
197         dragged_rotation = self._bitboards.get_rotation_on(bitboard_under_mouse)
198
199         self._drag_and_drop.set_dragged_piece(dragged_piece, dragged_colour,
200         dragged_rotation)
201         self._overlay_draw.set_hover_limit(False)
202
203     def remove_dragged_piece(self, coords):
204         piece, colour, rotation = self._drag_and_drop.get_dragged_info()
205
206         if coords and coords != self._selected_coords and piece != Piece.SPHINX:
207             self.remove_piece()
208             self.selected_coords = coords
209             self.set_piece(piece, colour, rotation)
210             self.selected_coords = None
211
212         self._drag_and_drop.remove_dragged_piece()
213         self._overlay_draw.set_hover_limit(True)
214
215     def set_piece(self, piece, colour, rotation):
216         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
217         selected_coords == (9, 0):
218             return
219
220         self.remove_piece()
221
222         selected_bitboard = coords_to_bitboard(self.selected_coords)
223         self._bitboards.set_square(selected_bitboard, piece, colour)
224         self._bitboards.set_rotation(selected_bitboard, rotation)
225
226         self.refresh_pieces()
227
228     def remove_piece(self):
229         if self.selected_coords is None or self.selected_coords == (0, 7) or self.
230         selected_coords == (9, 0):

```

```

226         return
227
228     selected_bitboard = coords_to_bitboard(self.selected_coords)
229     self._bitboards.clear_square(selected_bitboard, Colour.BLUE)
230     self._bitboards.clear_square(selected_bitboard, Colour.RED)
231     self._bitboards.clear_rotation(selected_bitboard)
232
233     self.refresh_pieces()
234
235     def rotate_piece(self, rotation_direction):
236         if self.selected_coords is None or self.selected_coords == (0, 7) or self.selected_coords == (9, 0):
237             return
238
239         selected_bitboard = coords_to_bitboard(self.selected_coords)
240
241         if self._bitboards.get_piece_on(selected_bitboard, Colour.BLUE) is None
242         and self._bitboards.get_piece_on(selected_bitboard, Colour.RED) is None:
243             return
244
245         current_rotation = self._bitboards.get_rotation_on(selected_bitboard)
246
247         if rotation_direction == RotationDirection.CLOCKWISE:
248             self._bitboards.update_rotation(selected_bitboard, selected_bitboard,
249             current_rotation.get_clockwise())
250         elif rotation_direction == RotationDirection.ANTICLOCKWISE:
251             self._bitboards.update_rotation(selected_bitboard, selected_bitboard,
252             current_rotation.get_anticlockwise())
253
254         self.refresh_pieces()
255
256     def select_tool(self, piece, colour):
257         dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: '
258         red_piece_buttons' }
259
260         self.deselect_tool()
261
262         if piece == 'ERASE':
263             EDITOR_WIDGETS['erase_button'].set_locked(True)
264             EDITOR_WIDGETS['erase_button'].set_next_icon()
265         elif piece == 'MOVE':
266             EDITOR_WIDGETS['move_button'].set_locked(True)
267             EDITOR_WIDGETS['move_button'].set_next_icon()
268         else:
269             EDITOR_WIDGETS[dict_name_map[colour]][piece].set_locked(True)
270             EDITOR_WIDGETS[dict_name_map[colour]][piece].set_next_icon()
271
272         self._selected_tool = piece
273         self._selected_tool_colour = colour
274
275     def deselect_tool(self):
276         dict_name_map = { Colour.BLUE: 'blue_piece_buttons', Colour.RED: '
277         red_piece_buttons' }
278
279         if self._selected_tool:
280             if self._selected_tool == 'ERASE':
281                 EDITOR_WIDGETS['erase_button'].set_locked(False)
282                 EDITOR_WIDGETS['erase_button'].set_next_icon()
283             elif self._selected_tool == 'MOVE':
284                 EDITOR_WIDGETS['move_button'].set_locked(False)
285                 EDITOR_WIDGETS['move_button'].set_next_icon()
286             else:

```

```

282         EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self.
    _selected_tool].set_locked(False)
283         EDITOR_WIDGETS[dict_name_map[self._selected_tool_colour]][self.
    _selected_tool].set_next_icon()
284
285         self._selected_tool = None
286         self._selected_tool_colour = None
287
288     def handle_resize(self):
289         super().handle_resize()
290         self._piece_group.handle_resize(EDITOR_WIDGETS['chessboard'].position,
    EDITOR_WIDGETS['chessboard'].size)
291         self._drag_and_drop.handle_resize(EDITOR_WIDGETS['chessboard'].position,
    EDITOR_WIDGETS['chessboard'].size)
292         self._overlay_draw.handle_resize(EDITOR_WIDGETS['chessboard'].position,
    EDITOR_WIDGETS['chessboard'].size)
293
294     def draw(self):
295         self._widget_group.draw()
296         self._overlay_draw.draw(window.screen)
297         self._piece_group.draw(window.screen)
298         self._drag_and_drop.draw(window.screen)

```

  

```

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
4  from data.assets import GRAPHICS
5  from data.widgets import *
6
7  blue_pieces_container = Rectangle(
8      relative_position=(0.25, 0),
9      relative_size=(0.13, 0.65),
10     scale_mode='height',
11     anchor_y='center',
12     anchor_x='center'
13 )
14
15 red_pieces_container = Rectangle(
16     relative_position=(-0.25, 0),
17     relative_size=(0.13, 0.65),
18     scale_mode='height',
19     anchor_y='center',
20     anchor_x='center'
21 )
22
23 bottom_actions_container = Rectangle(
24     relative_position=(0, 0.05),
25     relative_size=(0.4, 0.1),
26     anchor_x='center',
27     anchor_y='bottom'
28 )
29
30 top_actions_container = Rectangle(
31     relative_position=(0, 0.05),
32     relative_size=(0.3, 0.1),
33     anchor_x='center',
34     scale_mode='height'
35 )
36
37 top_right_container = Rectangle(
38     relative_position=(0, 0),

```

```

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

```

```

100         base_icon=GRAPHICS['anticlockwise_arrow_base'],
101         hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
102         press_icon=GRAPHICS['anticlockwise_arrow_press'],
103         event=CustomEvent(EditorEventType.ROTATE_PIECE_CLICK,
rotation_direction=RotationDirection.ANTICLOCKWISE)
104     ),
105     ReactiveIconButton(
106         parent=top_actions_container,
107         relative_position=(0, 0),
108         relative_size=(1, 1),
109         scale_mode='height',
110         anchor_x='right',
111         base_icon=GRAPHICS['copy_base'],
112         hover_icon=GRAPHICS['copy_hover'],
113         press_icon=GRAPHICS['copy_press'],
114         event=CustomEvent(EditorEventType.COPY_CLICK),
115     ),
116     ReactiveIconButton(
117         parent=top_actions_container,
118         relative_position=(0, 0),
119         relative_size=(1, 1),
120         scale_mode='height',
121         base_icon=GRAPHICS['delete_base'],
122         hover_icon=GRAPHICS['delete_hover'],
123         press_icon=GRAPHICS['delete_press'],
124         event=CustomEvent(EditorEventType.EMPTY_CLICK),
125     ),
126     ReactiveIconButton(
127         parent=top_actions_container,
128         relative_position=(0, 0),
129         relative_size=(1, 1),
130         scale_mode='height',
131         anchor_x='center',
132         base_icon=GRAPHICS['discard_arrow_base'],
133         hover_icon=GRAPHICS['discard_arrow_hover'],
134         press_icon=GRAPHICS['discard_arrow_press'],
135         event=CustomEvent(EditorEventType.RESET_CLICK),
136     ),
137     ReactiveIconButton(
138         relative_position=(0, 0),
139         fixed_position=(10, 0),
140         relative_size=(0.1, 0.1),
141         anchor_x='right',
142         anchor_y='center',
143         scale_mode='height',
144         base_icon=GRAPHICS['play_arrow_base'],
145         hover_icon=GRAPHICS['play_arrow_hover'],
146         press_icon=GRAPHICS['play_arrow_press'],
147         event=CustomEvent(EditorEventType.START_CLICK),
148     ),
149     ReactiveIconButton(
150         relative_position=(0, 0),
151         fixed_position=(10, 0),
152         relative_size=(0.1, 0.1),
153         anchor_y='center',
154         scale_mode='height',
155         base_icon=GRAPHICS['return_arrow_base'],
156         hover_icon=GRAPHICS['return_arrow_hover'],
157         press_icon=GRAPHICS['return_arrow_press'],
158         event=CustomEvent(EditorEventType.CONFIG_CLICK),
159     )
160 ],

```

```

161     'blue_piece_buttons': {},
162     'red_piece_buttons': {},
163     'erase_button':
164     MultipleIconButton(
165         parent=red_pieces_container,
166         relative_position=(0, 0),
167         relative_size=(0.2, 0.2),
168         scale_mode='height',
169         margin=10,
170         icons_dict={True: GRAPHICS['eraser'], False: get_highlighted_icon(GRAPHICS
171     ['eraser'])},
172         event=CustomEvent(EditorEventType.ERASE_CLICK),
173     ),
174     'move_button':
175     MultipleIconButton(
176         parent=blue_pieces_container,
177         relative_position=(0, 0),
178         relative_size=(0.2, 0.2),
179         scale_mode='height',
180         box_colours=BLUE_BUTTON_COLOURS,
181         icons_dict={True: GRAPHICS['finger'], False: get_highlighted_icon(GRAPHICS
182     ['finger'])},
183         event=CustomEvent(EditorEventType.MOVE_CLICK),
184     ),
185     'chessboard':
186     Chessboard(
187         relative_position=(0, 0),
188         relative_width=0.4,
189         scale_mode='width',
190         anchor_x='center',
191         anchor_y='center'
192     ),
193     'blue_start_button':
194     MultipleIconButton(
195         parent=bottom_actions_container,
196         relative_position=(0, 0),
197         relative_size=(1, 1),
198         scale_mode='height',
199         anchor_x='right',
200         box_colours=BLUE_BUTTON_COLOURS,
201         icons_dict={False: get_highlighted_icon(GRAPHICS['pharoah_0_a']), True:
202     GRAPHICS['pharoah_0_a']},
203         event=CustomEvent(EditorEventType.BLUE_START_CLICK)
204     ),
205     'red_start_button':
206     MultipleIconButton(
207         parent=bottom_actions_container,
208         relative_position=(0, 0),
209         relative_size=(1, 1),
210         scale_mode='height',
211         icons_dict={True: GRAPHICS['pharoah_1_a'], False: get_highlighted_icon(
212     GRAPHICS['pharoah_1_a'])},
213         event=CustomEvent(EditorEventType.RED_START_CLICK)
214     )
215 }
216
217 for index, piece in enumerate([piece for piece in Piece if piece != Piece.SPHINX])
218 :
219     blue_icon = GRAPHICS[f'{piece.name.lower()}_0_a']
220     dimmed_blue_icon = get_highlighted_icon(blue_icon)
221
222     EDITOR_WIDGETS['blue_piece_buttons'][piece] = MultipleIconButton(

```

```

218         parent=blue_pieces_container,
219         relative_position=(0, (index + 1) / 5),
220         relative_size=(0.2, 0.2),
221         scale_mode='height',
222         box_colours=BLUE_BUTTON_COLOURS,
223         icons_dict={True: blue_icon, False: dimmed_blue_icon},
224         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
active_colour=Colour.BLUE)
225     )
226
227     red_icon = GRAPHICS[f'{piece.name.lower()}_1_a']
228
229     dimmed_red_icon = get_highlighted_icon(red_icon)
230
231     EDITOR_WIDGETS['red_piece_buttons'][piece] = MultipleIconButton(
232         parent=red_pieces_container,
233         relative_position=(0, (index + 1) / 5),
234         relative_size=(0.2, 0.2),
235         scale_mode='height',
236         icons_dict={True: red_icon, False: dimmed_red_icon},
237         event=CustomEvent(EditorEventType.PICK_PIECE_CLICK, piece=piece,
active_colour=Colour.RED)
238     )

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
16
17 logger = initialise_logger(__name__)
18
19 class Game(_State):
20     def __init__(self):
21         super().__init__()
22
23     def cleanup(self):
24         super().cleanup()
25
26         window.clear_apply_arguments(ShaderType.BLOOM)
27         window.clear_effect(ShaderType.RAYS)
28
29         game_entry = GameEntry(self.model.states, final_fen_string=self.model.
get_fen_string())
30         inserted_game = insert_into_games(game_entry.convert_to_row())
31
32         return inserted_game
33
34     def switch_to_menu(self):
35         self.next = 'menu'
36         self.done = True
37

```



```

38     def switch_to_review(self):
39         self.next = 'review'
40         self.done = True
41
42     def startup(self, persist):
43         music = MUSIC[['cpu_easy', 'cpu_medium', 'cpu_hard'][persist['CPU_DEPTH']
44 - 2]] if persist['CPU_ENABLED'] else MUSIC['pvp']
45         super().startup(music=music)
46
47         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_LASERS)
48         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
49         binded_startup = partial(self.startup, persist)
50
51         self.model = GameModel(persist)
52         self.view = GameView(self.model)
53         self.pause_view = PauseView(self.model)
54         self.win_view = WinView(self.model)
55         self.controller = GameController(self.model, self.view, self.win_view,
self.pause_view, self.switch_to_menu, self.switch_to_review, binded_startup)
56
57         self.view.draw()
58
59         audio.play_sfx(SFX['game_start_1'])
60         audio.play_sfx(SFX['game_start_2'])
61
62     def get_event(self, event):
63         self.controller.handle_event(event)
64
65     def handle_resize(self):
66         self.view.handle_resize()
67         self.win_view.handle_resize()
68         self.pause_view.handle_resize()
69
70     def draw(self):
71         self.view.draw()
72         self.win_view.draw()
73         self.pause_view.draw()
74
75     def update(self):
76         self.controller.check_cpu()
77         super().update()
78
79
80 1 from data.widgets import *
81 2 from data.components.custom_event import CustomEvent
82 3 from data.constants import GameEventType, RotationDirection, Colour
83 4 from data.assets import GRAPHICS
84
85 5
86 6 right_container = Rectangle(
87 7     relative_position=(0.05, 0),
88 8     relative_size=(0.2, 0.5),
89 9     anchor_y='center',
90 10    anchor_x='right',
91 11 )
92
93 12
94 13 rotate_container = Rectangle(
95 14     relative_position=(0, 0.05),
96 15     relative_size=(0.2, 0.1),
97 16     anchor_x='center',
98 17     anchor_y='bottom',
99 18 )

```

```

19
20 move_list = MoveList(
21     parent=right_container,
22     relative_position=(0, 0),
23     relative_width=1,
24     minimum_height=300,
25     move_list=[]
26 )
27
28 resign_button = TextButton(
29     parent=right_container,
30     relative_position=(0, 0),
31     relative_size=(0.5, 0.2),
32     fit_vertical=False,
33     anchor_y='bottom',
34     text="    Resign",
35     margin=5,
36     event=CustomEvent(GameEventType.RESIGN_CLICK)
37 )
38
39 draw_button = TextButton(
40     parent=right_container,
41     relative_position=(0, 0),
42     relative_size=(0.5, 0.2),
43     fit_vertical=False,
44     anchor_x='right',
45     anchor_y='bottom',
46     text="    Draw",
47     margin=5,
48     event=CustomEvent(GameEventType.DRAW_CLICK)
49 )
50
51 top_right_container = Rectangle(
52     relative_position=(0, 0),
53     relative_size=(0.225, 0.075),
54     fixed_position=(5, 5),
55     anchor_x='right',
56     scale_mode='height'
57 )
58
59 GAME_WIDGETS = {
60     'help':
61     Icon(
62         relative_position=(0, 0),
63         relative_size=(1.02, 1.02),
64         icon=GRAPHICS['game_help'],
65         anchor_x='center',
66         anchor_y='center',
67         border_width=0,
68         fill_colour=(0, 0, 0, 0)
69     ),
70     'tutorial':
71     Icon(
72         relative_position=(0, 0),
73         relative_size=(0.9, 0.9),
74         icon=GRAPHICS['game_tutorial'],
75         anchor_x='center',
76         anchor_y='center',
77     ),
78     'default': [
79         right_container,
80         rotate_container,

```

```

81     top_right_container,
82     ReactiveIconButton(
83         parent=top_right_container,
84         relative_position=(0, 0),
85         relative_size=(1, 1),
86         anchor_x='right',
87         scale_mode='height',
88         base_icon=GRAPHICS['home_base'],
89         hover_icon=GRAPHICS['home_hover'],
90         press_icon=GRAPHICS['home_press'],
91         event=CustomEvent(GameEventType.MENU_CLICK)
92     ),
93     ReactiveIconButton(
94         parent=top_right_container,
95         relative_position=(0, 0),
96         relative_size=(1, 1),
97         scale_mode='height',
98         base_icon=GRAPHICS['tutorial_base'],
99         hover_icon=GRAPHICS['tutorial_hover'],
100        press_icon=GRAPHICS['tutorial_press'],
101        event=CustomEvent(GameEventType.TUTORIAL_CLICK)
102    ),
103    ReactiveIconButton(
104        parent=top_right_container,
105        relative_position=(0.33, 0),
106        relative_size=(1, 1),
107        scale_mode='height',
108        base_icon=GRAPHICS['help_base'],
109        hover_icon=GRAPHICS['help_hover'],
110        press_icon=GRAPHICS['help_press'],
111        event=CustomEvent(GameEventType.HELP_CLICK)
112    ),
113    ReactiveIconButton(
114        parent=rotate_container,
115        relative_position=(0, 0),
116        relative_size=(1, 1),
117        scale_mode='height',
118        anchor_x='right',
119        base_icon=GRAPHICS['clockwise_arrow_base'],
120        hover_icon=GRAPHICS['clockwise_arrow_hover'],
121        press_icon=GRAPHICS['clockwise_arrow_press'],
122        event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
RotationDirection.CLOCKWISE)
123    ),
124    ReactiveIconButton(
125        parent=rotate_container,
126        relative_position=(0, 0),
127        relative_size=(1, 1),
128        scale_mode='height',
129        base_icon=GRAPHICS['anticlockwise_arrow_base'],
130        hover_icon=GRAPHICS['anticlockwise_arrow_hover'],
131        press_icon=GRAPHICS['anticlockwise_arrow_press'],
132        event=CustomEvent(GameEventType.ROTATE_PIECE, rotation_direction=
RotationDirection.ANTICLOCKWISE)
133    ),
134    resign_button,
135    draw_button,
136    Icon(
137        parent=resign_button,
138        relative_position=(0, 0),
139        relative_size=(0.75, 0.75),
140        fill_colour=(0, 0, 0, 0),

```

```

141         scale_mode='height',
142         anchor_y='center',
143         border_radius=0,
144         border_width=0,
145         margin=5,
146         icon=GRAPHICS['resign']
147     ),
148     Icon(
149         parent=draw_button,
150         relative_position=(0, 0),
151         relative_size=(0.75, 0.75),
152         fill_colour=(0, 0, 0, 0),
153         scale_mode='height',
154         anchor_y='center',
155         border_radius=0,
156         border_width=0,
157         margin=5,
158         icon=GRAPHICS['draw']
159     ),
160 ],
161 'scroll_area': # REMEMBER SCROLL AREA AFTER CONTAINER FOR RESIZING
162 ScrollArea(
163     parent=right_container,
164     relative_position=(0, 0),
165     relative_size=(1, 0.8),
166     vertical=True,
167     widget=move_list
168 ),
169 'move_list':
170     move_list,
171 'blue_timer':
172 Timer(
173     relative_position=(0.05, 0.05),
174     anchor_y='center',
175     relative_size=(0.1, 0.1),
176     active_colour=Colour.BLUE,
177     event=CustomEvent(GameEventType.TIMER_END),
178 ),
179 'red_timer':
180 Timer(
181     relative_position=(0.05, -0.05),
182     anchor_y='center',
183     relative_size=(0.1, 0.1),
184     active_colour=Colour.RED,
185     event=CustomEvent(GameEventType.TIMER_END),
186 ),
187 'status_text':
188 Text(
189     relative_position=(0, 0.05),
190     relative_size=(0.4, 0.1),
191     anchor_x='center',
192     fit_vertical=False,
193     margin=10,
194     text="g",
195     minimum_width=400
196 ),
197 'chessboard':
198 Chessboard(
199     relative_position=(0, 0),
200     anchor_x='center',
201     anchor_y='center',
202     scale_mode='width',

```

```

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

```

```

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

```

```

327         fill_colour=(0, 0, 0, 0),
328     ),
329     'by_resignation':
330     Icon(
331         parent=win_container,
332         relative_position=(0, 0.375),
333         relative_size=(0.8, 0.1),
334         anchor_x='center',
335         border_width=0,
336         margin=0,
337         icon=GRAPHICS['by_resignation'],
338         fill_colour=(0, 0, 0, 0),
339     ),
340     'by_draw':
341     Icon(
342         parent=win_container,
343         relative_position=(0, 0.375),
344         relative_size=(0.8, 0.1),
345         anchor_x='center',
346         border_width=0,
347         margin=0,
348         icon=GRAPHICS['by_draw'],
349         fill_colour=(0, 0, 0, 0),
350     ),
351     'by_timeout':
352     Icon(
353         parent=win_container,
354         relative_position=(0, 0.375),
355         relative_size=(0.8, 0.1),
356         anchor_x='center',
357         border_width=0,
358         margin=0,
359         icon=GRAPHICS['by_timeout'],
360         fill_colour=(0, 0, 0, 0),
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
6
7  logger = initialise_logger(__name__)
8
9  class BitboardCollection:
10     def __init__(self, fen_string):
11         self.piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char:
   EMPTY_BB for char in Piece}]
12         self.combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
13         self.combined_all_bitboard = EMPTY_BB
14         self.rotation_bitboards = [EMPTY_BB, EMPTY_BB]
15         self.active_colour = Colour.BLUE
16         self._hasher = ZobristHasher()
17
18         try:
19             if fen_string:
20                 self.piece_bitboards, self.combined_colour_bitboards, self.
   combined_all_bitboard, self.rotation_bitboards, self.active_colour =
   parse_fen_string(fen_string)
21                 self.initialise_hash()

```

```

22         except ValueError as error:
23             logger.error('Please input a valid FEN string:', error)
24             raise error
25
26     def __str__(self):
27         """
28         Returns a string representation of the bitboards.
29
30         Returns:
31         str: Bitboards formatted with piece type and colour shown.
32         """
33         characters = ''
34         for rank in reversed(Rank):
35             for file in File:
36                 bitboard = 1 << (rank * 10 + file)
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
41                 if piece is not None:
42                     characters += f'{piece.upper()} ' if colour == Colour.BLUE
else piece} '
43                 else:
44                     characters += '. '
45
46             characters += '\n\n'
47
48         return characters
49
50     def get_rotation_string(self):
51         """
52         Returns a string representation of the board rotations.
53
54         Returns:
55         str: Board formatted with only rotations shown.
56         """
57         characters = ''
58         for rank in reversed(Rank):
59
60             for file in File:
61                 mask = 1 << (rank * 10 + file)
62                 rotation = self.get_rotation_on(mask)
63                 has_piece = bb_helpers.is_occupied(self.combined_all_bitboard,
mask)
64
65                 if has_piece:
66                     characters += f'{rotation.upper()} '
67                 else:
68                     characters += '. '
69
70             characters += '\n\n'
71
72         return characters
73
74     def initialise_hash(self):
75         """
76         Initialises the Zobrist hash for the current board state.
77         """
78         for piece in Piece:
79             for colour in Colour:
80                 piece_bitboard = self.get_piece_bitboard(piece, colour)

```



```

81
82         for occupied_bitboard in bb_helpers.occupied_squares(
piece_bitboard):
83             self._hasher.apply_piece_hash(occupied_bitboard, piece, colour
)
84
85         for bitboard in bb_helpers.loop_all_squares():
86             rotation = self.get_rotation_on(bitboard)
87             self._hasher.apply_rotation_hash(bitboard, rotation)
88
89         if self.active_colour == Colour.RED:
90             self._hasher.apply_red_move_hash()
91
92     def flip_colour(self):
93         """
94         Flips the active colour and updates the Zobrist hash.
95         """
96         self.active_colour = self.active_colour.get_flipped_colour()
97
98         if self.active_colour == Colour.RED:
99             self._hasher.apply_red_move_hash()
100
101     def update_move(self, src, dest):
102         """
103         Updates the bitboards for a move.
104
105         Args:
106             src (int): The bitboard representation of the source square.
107             dest (int): The bitboard representation of the destination square.
108         """
109         piece = self.get_piece_on(src, self.active_colour)
110
111         self.clear_square(src, Colour.BLUE)
112         self.clear_square(dest, Colour.BLUE)
113         self.clear_square(src, Colour.RED)
114         self.clear_square(dest, Colour.RED)
115
116         self.set_square(dest, piece, self.active_colour)
117
118     def update_rotation(self, src, dest, new_rotation):
119         """
120         Updates the rotation bitboards for a move.
121
122         Args:
123             src (int): The bitboard representation of the source square.
124             dest (int): The bitboard representation of the destination square.
125             new_rotation (Rotation): The new rotation.
126         """
127         self.clear_rotation(src)
128         self.set_rotation(dest, new_rotation)
129
130     def clear_rotation(self, bitboard):
131         """
132         Clears the rotation for a given square.
133
134         Args:
135             bitboard (int): The bitboard representation of the square.
136         """
137         old_rotation = self.get_rotation_on(bitboard)
138         rotation_1, rotation_2 = self.rotation_bitboards
139         self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.clear_square(
rotation_1, bitboard)

```

```

140         self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.clear_square
        (rotation_2, bitboard)
141
142         self._hasher.apply_rotation_hash(bitboard, old_rotation)
143
144     def clear_square(self, bitboard, colour):
145         """
146         Clears a square piece and rotation for a given colour.
147
148         Args:
149             bitboard (int): The bitboard representation of the square.
150             colour (Colour): The colour to clear.
151         """
152         piece = self.get_piece_on(bitboard, colour)
153
154         if piece is None:
155             return
156
157         piece_bitboard = self.get_piece_bitboard(piece, colour)
158         colour_bitboard = self.combined_colour_bitboards[colour]
159         all_bitboard = self.combined_all_bitboard
160
161         self.piece_bitboards[colour][piece] = bb_helpers.clear_square(
        piece_bitboard, bitboard)
162         self.combined_colour_bitboards[colour] = bb_helpers.clear_square(
        colour_bitboard, bitboard)
163         self.combined_all_bitboard = bb_helpers.clear_square(all_bitboard,
        bitboard)
164
165         self._hasher.apply_piece_hash(bitboard, piece, colour)
166
167     def set_rotation(self, bitboard, rotation):
168         """
169         Sets the rotation for a given square.
170
171         Args:
172             bitboard (int): The bitboard representation of the square.
173             rotation (Rotation): The rotation to set.
174         """
175         rotation_1, rotation_2 = self.rotation_bitboards
176         self._hasher.apply_rotation_hash(bitboard, rotation)
177
178         match rotation:
179             case Rotation.UP:
180                 return
181             case Rotation.RIGHT:
182                 self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.
        set_square(rotation_1, bitboard)
183                 return
184             case Rotation.DOWN:
185                 self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.
        set_square(rotation_2, bitboard)
186                 return
187             case Rotation.LEFT:
188                 self.rotation_bitboards[RotationIndex.FIRSTBIT] = bb_helpers.
        set_square(rotation_1, bitboard)
189                 self.rotation_bitboards[RotationIndex.SECONDBIT] = bb_helpers.
        set_square(rotation_2, bitboard)
190                 return
191             case _:
192                 raise ValueError('Invalid rotation input (bitboard.py):', rotation
        )

```

```

193
194 def set_square(self, bitboard, piece, colour):
195     """
196     Sets a piece on a given square.
197
198     Args:
199         bitboard (int): The bitboard representation of the square.
200         piece (Piece): The piece to set.
201         colour (Colour): The colour of the piece.
202     """
203     piece_bitboard = self.get_piece_bitboard(piece, colour)
204     colour_bitboard = self.combined_colour_bitboards[colour]
205     all_bitboard = self.combined_all_bitboard
206
207     self.piece_bitboards[colour][piece] = bb_helpers.set_square(piece_bitboard
, bitboard)
208     self.combined_colour_bitboards[colour] = bb_helpers.set_square(
colour_bitboard, bitboard)
209     self.combined_all_bitboard = bb_helpers.set_square(all_bitboard, bitboard)
210
211     self._hasher.apply_piece_hash(bitboard, piece, colour)
212
213 def get_piece_bitboard(self, piece, colour):
214     """
215     Gets the bitboard for a piece type for a given colour.
216
217     Args:
218         piece (Piece): The piece bitboard to get.
219         colour (Colour): The colour of the piece.
220
221     Returns:
222         int: The bitboard representation for all squares occupied by that
piece type.
223     """
224     return self.piece_bitboards[colour][piece]
225
226 def get_piece_on(self, target_bitboard, colour):
227     """
228     Gets the piece on a given square for a given colour.
229
230     Args:
231         target_bitboard (int): The bitboard representation of the square.
232         colour (Colour): The colour of the piece.
233
234     Returns:
235         Piece: The piece on the square, or None if square is empty.
236     """
237     if not (bb_helpers.is_occupied(self.combined_colour_bitboards[colour],
target_bitboard)):
238         return None
239
240     return next(
241         (piece for piece in Piece if
242          bb_helpers.is_occupied(self.get_piece_bitboard(piece, colour),
target_bitboard)),
243         None)
244
245 def get_rotation_on(self, target_bitboard):
246     """
247     Gets the rotation on a given square.
248
249     Args:

```

```

250         target_bitboard (int): The bitboard representation of the square.
251
252     Returns:
253         Rotation: The rotation on the square.
254     """
255     rotationBits = [bb_helpers.is_occupied(self.rotation_bitboards[
RotationIndex.SECONDBIT], target_bitboard), bb_helpers.is_occupied(self.
rotation_bitboards[RotationIndex.FIRSTBIT], target_bitboard)]
256
257     match rotationBits:
258         case [False, False]:
259             return Rotation.UP
260         case [False, True]:
261             return Rotation.RIGHT
262         case [True, False]:
263             return Rotation.DOWN
264         case [True, True]:
265             return Rotation.LEFT
266
267 def get_colour_on(self, target_bitboard):
268     """
269     Gets the colour of the piece on a given square.
270
271     Args:
272         target_bitboard (int): The bitboard representation of the square.
273
274     Returns:
275         Colour: The colour of the piece on the square.
276     """
277     for piece in Piece:
278         if self.get_piece_bitboard(piece, Colour.BLUE) & target_bitboard !=
EMPTY_BB:
279             return Colour.BLUE
280         elif self.get_piece_bitboard(piece, Colour.RED) & target_bitboard !=
EMPTY_BB:
281             return Colour.RED
282
283 def get_piece_count(self, piece, colour):
284     """
285     Gets the count of a given piece type and colour.
286
287     Args:
288         piece (Piece): The piece to count.
289         colour (Colour): The colour of the piece.
290
291     Returns:
292         int: The number of that piece of that colour on the board.
293     """
294     return bb_helpers.pop_count(self.get_piece_bitboard(piece, colour))
295
296 def get_hash(self):
297     """
298     Gets the Zobrist hash of the current board state.
299
300     Returns:
301         int: The Zobrist hash.
302     """
303     return self._hasher.hash
304
305 def convert_to_piece_list(self):
306     """
307     Converts all bitboards to a list of pieces.

```

```

308
309     Returns:
310         list: Board represented as a 2D list of Piece and Rotation objects.
311     """
312     piece_list = []
313
314     for i in range(80):
315         if x := self.get_piece_on(1 << i, Colour.BLUE):
316             rotation = self.get_rotation_on(1 << i)
317             piece_list.append((x.upper(), rotation))
318         elif y := self.get_piece_on(1 << i, Colour.RED):
319             rotation = self.get_rotation_on(1 << i)
320             piece_list.append((y, rotation))
321         else:
322             piece_list.append(None)
323
324     return piece_list

```

```

1 from data.states.game.components.move import Move
2 from data.states.game.components.laser import Laser
3
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
8
9 class Board:
10     def __init__(self, fen_string="sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/
    pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa b"):
11         self.bitboards = BitboardCollection(fen_string)
12         self.hash_list = [self.bitboards.get_hash()]
13
14     def __str__(self):
15         """
16         Returns a string representation of the board.
17
18         Returns:
19             str: Board formatted as string.
20         """
21         characters = '8 '
22         pieces = defaultdict(int)
23
24         for rank_idx, rank in enumerate(reversed(Rank)):
25             for file_idx, file in enumerate(File):
26                 mask = 1 << (rank * 10 + file)
27                 blue_piece = self.bitboards.get_piece_on(mask, Colour.BLUE)
28                 red_piece = self.bitboards.get_piece_on(mask, Colour.RED)
29
30                 if blue_piece:
31                     pieces[blue_piece.value.upper()] += 1
32                     characters += f'{blue_piece.upper()} '
33                 elif red_piece:
34                     pieces[red_piece.value] += 1
35                     characters += f'{red_piece} '
36                 else:
37                     characters += ' '
38
39             characters += f'\n\n{7 - rank_idx} '
40         characters += 'A B C D E F G H I J\n\n'
41         characters += str(dict(pieces))

```

```

42         characters += f'\nCURRENT PLAYER TO MOVE: {self.bitboards.active_colour.
name}\n'
43     return characters
44
45     def get_piece_list(self):
46         """
47         Converts the board bitboards to a list of pieces.
48
49         Returns:
50             list: List of Pieces.
51         """
52         return self.bitboards.convert_to_piece_list()
53
54     def get_active_colour(self):
55         """
56         Gets the active colour.
57
58         Returns:
59             Colour: The active colour.
60         """
61         return self.bitboards.active_colour
62
63     def to_hash(self):
64         """
65         Gets the hash of the current board state.
66
67         Returns:
68             int: A Zobrist hash.
69         """
70         return self.bitboards.get_hash()
71
72     def check_win(self):
73         """
74         Checks for a Pharoah capture or threefold-repetition.
75
76         Returns:
77             Colour | Miscellaneous: The winning colour, or Miscellaneous.DRAW.
78         """
79         for colour in Colour:
80             if self.bitboards.get_piece_bitboard(Piece.PHAROAH, colour) ==
EMPTY_BB:
81                 return colour.get_flipped_colour()
82
83             if self.hash_list.count(self.hash_list[-1]) >= 3:
84                 return Miscellaneous.DRAW
85
86         return None
87
88     def apply_move(self, move, fire_laser=True, add_hash=False):
89         """
90         Applies a move to the board.
91
92         Args:
93             move (Move): The move to apply.
94             fire_laser (bool): Whether to fire the laser after the move.
95             add_hash (bool): Whether to add the board state hash to the hash list.
96
97         Returns:
98             Laser: The laser trajectory result.
99         """
100         piece_symbol = self.bitboards.get_piece_on(move.src, self.bitboards.
active_colour)

```

```

101
102     if piece_symbol is None:
103         raise ValueError('Invalid move - no piece found on source square')
104     elif piece_symbol == Piece.SPHINX:
105         raise ValueError('Invalid move - sphinx piece is immovable')
106
107     if move.move_type == MoveType.MOVE:
108         possible_moves = self.get_valid_squares(move.src)
109         if bb_helpers.is_occupied(move.dest, possible_moves) is False:
110             raise ValueError('Invalid move - destination square is occupied')
111
112         piece_rotation = self.bitboards.get_rotation_on(move.src)
113
114         self.bitboards.update_move(move.src, move.dest)
115         self.bitboards.update_rotation(move.src, move.dest, piece_rotation)
116
117     elif move.move_type == MoveType.ROTATE:
118         piece_symbol = self.bitboards.get_piece_on(move.src, self.bitboards.
active_colour)
119         piece_rotation = self.bitboards.get_rotation_on(move.src)
120
121         if move.rotation_direction == RotationDirection.CLOCKWISE:
122             new_rotation = piece_rotation.get_clockwise()
123         elif move.rotation_direction == RotationDirection.ANTICLOCKWISE:
124             new_rotation = piece_rotation.get_anticlockwise()
125
126         self.bitboards.update_rotation(move.src, move.src, new_rotation)
127
128     laser = None
129     if fire_laser:
130         laser = self.fire_laser(add_hash)
131
132     if add_hash:
133         self.hash_list.append(self.bitboards.get_hash())
134
135     self.bitboards.flip_colour()
136
137     return laser
138
139 def undo_move(self, move, laser_result):
140     """
141     Undoes a move on the board.
142
143     Args:
144         move (Move): The move to undo.
145         laser_result (Laser): The laser trajectory result.
146     """
147     self.bitboards.flip_colour()
148
149     if laser_result.hit_square_bitboard:
150         # Get info of destroyed piece, and add it to the board again
151         src = laser_result.hit_square_bitboard
152         piece = laser_result.piece_hit
153         colour = laser_result.piece_colour
154         rotation = laser_result.piece_rotation
155
156         self.bitboards.set_square(src, piece, colour)
157         self.bitboards.clear_rotation(src)
158         self.bitboards.set_rotation(src, rotation)
159
160     # Create new Move object that is the inverse of the passed move
161     if move.move_type == MoveType.MOVE:

```

```

162         reversed_move = Move.instance_from_bitboards(MoveType.MOVE, move.dest,
163         move.src)
164     elif move.move_type == MoveType.ROTATE:
165         reversed_move = Move.instance_from_bitboards(MoveType.ROTATE, move.src
166         , move.src, move.rotation_direction.get_opposite())
167
168     self.apply_move(reversed_move, fire_laser=False)
169     self.bitboards.flip_colour()
170
171     def remove_piece(self, square_bitboard):
172         """
173         Removes a piece from a given square.
174
175         Args:
176             square_bitboard (int): The bitboard representation of the square.
177         """
178         self.bitboards.clear_square(square_bitboard, Colour.BLUE)
179         self.bitboards.clear_square(square_bitboard, Colour.RED)
180         self.bitboards.clear_rotation(square_bitboard)
181
182     def get_valid_squares(self, src_bitboard, colour=None):
183         """
184         Gets valid squares for a piece to move to.
185
186         Args:
187             src_bitboard (int): The bitboard representation of the source square.
188             colour (Colour, optional): The active colour of the piece.
189
190         Returns:
191             int: The bitboard representation of valid squares.
192         """
193         target_top_left = (src_bitboard & A_FILE_MASK & EIGHT_RANK_MASK) << 9
194         target_top_middle = (src_bitboard & EIGHT_RANK_MASK) << 10
195         target_top_right = (src_bitboard & J_FILE_MASK & EIGHT_RANK_MASK) << 11
196         target_middle_right = (src_bitboard & J_FILE_MASK) << 1
197
198         target_bottom_right = (src_bitboard & J_FILE_MASK & ONE_RANK_MASK) >> 9
199         target_bottom_middle = (src_bitboard & ONE_RANK_MASK) >> 10
200         target_bottom_left = (src_bitboard & A_FILE_MASK & ONE_RANK_MASK) >> 11
201         target_middle_left = (src_bitboard & A_FILE_MASK) >> 1
202
203         possible_moves = target_top_left | target_top_middle | target_top_right |
204         target_middle_right | target_bottom_right | target_bottom_middle |
205         target_bottom_left | target_middle_left
206
207         if colour is not None:
208             valid_possible_moves = possible_moves & ~self.bitboards.
209             combined_colour_bitboards[colour]
210         else:
211             valid_possible_moves = possible_moves & ~self.bitboards.
212             combined_all_bitboard
213
214         return valid_possible_moves
215
216     def get_mobility(self, colour):
217         """
218         Gets all valid squares for a given colour.
219
220         Args:
221             colour (Colour): The colour of the pieces.
222
223         Returns:

```



```

218         int: The bitboard representation of all valid squares.
219     """
220     active_pieces = self.get_all_active_pieces(colour)
221     possible_moves = 0
222
223     for square in bb_helpers.occupied_squares(active_pieces):
224         possible_moves += bb_helpers.pop_count(self.get_valid_squares(square))
225
226     return possible_moves
227
228 def get_all_active_pieces(self, colour=None):
229     """
230     Gets all active pieces for the current player.
231
232     Args:
233         colour (Colour): Active colour of pieces to retrieve. Defaults to None
234     .
235
236     Returns:
237         int: The bitboard representation of all active pieces.
238     """
239     if colour is None:
240         colour = self.bitboards.active_colour
241
242     active_pieces = self.bitboards.combined_colour_bitboards[colour]
243     sphinx_bitboard = self.bitboards.get_piece_bitboard(Piece.SPHINX, colour)
244     return active_pieces ^ sphinx_bitboard
245
246 def fire_laser(self, remove_hash):
247     """
248     Fires the laser and removes hit pieces.
249
250     Args:
251         remove_hash (bool): Whether to clear the hash list if a piece is hit.
252
253     Returns:
254         Laser: The result of firing the laser.
255     """
256     laser = Laser(self.bitboards)
257
258     if laser.hit_square_bitboard:
259         self.remove_piece(laser.hit_square_bitboard)
260
261         if remove_hash:
262             self.hash_list = [] # Remove all hashes for threefold repetition,
263             as the position is impossible to be repeated after a piece is removed
264     return laser
265
266 def generate_square_moves(self, src):
267     """
268     Generates all valid moves for a piece on a given square.
269
270     Args:
271         src (int): The bitboard representation of the source square.
272
273     Yields:
274         Move: A valid move for the piece.
275     """
276     for dest in bb_helpers.occupied_squares(self.get_valid_squares(src)):
277         yield Move(MoveType.MOVE, src, dest)
278
279 def generate_all_moves(self, colour):

```

```

278         """
279         Generates all valid moves for a given colour.
280
281         Args:
282             colour (Colour): The colour of the pieces.
283
284         Yields:
285             Move: A valid move for the active colour.
286         """
287         sphinx_bitboard = self.bitboards.get_piece_bitboard(Piece.SPHINX, colour)
288         # Remove source squares for Sphinx pieces, as they cannot be moved
289         sphinx_masked_bitboard = self.bitboards.combined_colour_bitboards[colour]
290         ~ sphinx_bitboard
291
292         for square in bb_helpers.occupied_squares(sphinx_masked_bitboard):
293             # Generate movement moves
294             yield from self.generate_square_moves(square)
295
296             # Generate rotational moves
297             for rotation_direction in RotationDirection:
298                 yield Move(MoveType.ROTATE, square, rotation_direction=
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
5 from data.managers.animation import animation
6
7 class CaptureDraw:
8     def __init__(self, board_position, board_size):
9         self._board_position = board_position
10        self._square_size = board_size[0] / 10
11        self._particles_draw = ParticlesDraw()
12
13    def add_capture(self, piece, colour, rotation, piece_coords, sphinx_coords,
active_colour, particles=True, shake=True):
14        if particles:
15            self._particles_draw.add_captured_piece(
16                piece,
17                colour,
18                rotation,
19                coords_to_screen_pos(piece_coords, self._board_position, self.
_square_size),
20                self._square_size
21            )
22            self._particles_draw.add_sparks(
23                3,
24                (255, 0, 0) if active_colour == Colour.RED else (0, 0, 255),
25                coords_to_screen_pos(sphinx_coords, self._board_position, self.
_square_size)
26            )
27
28        if shake:
29            window.set_effect(ShaderType.SHAKE)
30            animation.set_timer(500, lambda: window.clear_effect(ShaderType.SHAKE))
31
32    def draw(self, screen):
33        self._particles_draw.draw(screen)
34
35    def update(self):

```

```

36         self._particles_draw.update()
37
38     def handle_resize(self, board_position, board_size):
39         self._board_position = board_position
40         self._square_size = board_size[0] / 10
41
42     1 import pygame
43     2 from data.constants import CursorMode
44     3 from data.states.game.components.piece_sprite import PieceSprite
45     4 from data.managers.cursor import cursor
46     5 from data.managers.audio import audio
47     6 from data.assets import SFX
48     7
49     8 DRAG_THRESHOLD = 500
50     9
51     10 class DragAndDrop:
52     11     def __init__(self, board_position, board_size, change_cursor=True):
53     12         self._board_position = board_position
54     13         self._board_size = board_size
55     14         self._change_cursor = change_cursor
56     15         self._ticks_since_drag = 0
57     16
58     17         self.dragged_sprite = None
59     18
60     19     def set_dragged_piece(self, piece, colour, rotation):
61     20         sprite = PieceSprite(piece=piece, colour=colour, rotation=rotation)
62     21         sprite.set_geometry((0, 0), self._board_size[0] / 10)
63     22         sprite.set_image()
64     23
65     24         self.dragged_sprite = sprite
66     25         self._ticks_since_drag = pygame.time.get_ticks()
67     26
68     27         if self._change_cursor:
69     28             cursor.set_mode(CursorMode.CLOSEDHAND)
70     29
71     30     def remove_dragged_piece(self):
72     31         self.dragged_sprite = None
73     32         time_dragged = pygame.time.get_ticks() - self._ticks_since_drag
74     33         self._ticks_since_drag = 0
75     34
76     35         if self._change_cursor:
77     36             cursor.set_mode(CursorMode.OPENHAND)
78     37
79     38         return time_dragged > DRAG_THRESHOLD
80     39
81     40     def get_dragged_info(self):
82     41         return self.dragged_sprite.type, self.dragged_sprite.colour, self.
83     42         dragged_sprite.rotation
84     43
85     44     def draw(self, screen):
86     45         if self.dragged_sprite is None:
87     46             return
88     47
89     48         self.dragged_sprite.rect.center = pygame.mouse.get_pos()
90     49         screen.blit(self.dragged_sprite.image, self.dragged_sprite.rect.topleft)
91     50
92     51     def handle_resize(self, board_position, board_size):
93     52         if self.dragged_sprite:
94     53             self.dragged_sprite.set_geometry(board_position, board_size[0] / 10)
95     54
96     55         self._board_position = board_position
97     56         self._board_size = board_size

```

```

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
3
4 def parse_fen_string(fen_string):
5     #sc3ncfcncpb2/2pc7/3Pd6/pa1Pc1rbra1pb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2
6     PdNaFaNa3Sa b
7     piece_bitboards = [{char: EMPTY_BB for char in Piece}, {char: EMPTY_BB for
8     char in Piece}]
9     rotation_bitboards = [EMPTY_BB, EMPTY_BB]
10    combined_colour_bitboards = [EMPTY_BB, EMPTY_BB]
11    combined_all_bitboard = 0
12    part_1, part_2 = fen_string.split(' ')
13
14    rank = 7
15    file = 0
16
17    piece_count = {char.lower(): 0 for char in Piece} | {char.upper(): 0 for char
18    in Piece}
19
20    for index, character in enumerate(part_1):
21        square = rank * 10 + file
22
23        if character.lower() in Piece:
24            piece_count[character] += 1
25            if character.isupper():
26                piece_bitboards[Colour.BLUE][character.lower()] |= 1 << square
27
28            else:
29                piece_bitboards[Colour.RED][character.lower()] |= 1 << square
30
31        rotation = part_1[index + 1]
32        match rotation:
33            case Rotation.UP:
34                pass
35            case Rotation.RIGHT:
36                rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
37            case Rotation.DOWN:
38                rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
39            case Rotation.LEFT:
40                rotation_bitboards[RotationIndex.SECONDBIT] |= 1 << square
41                rotation_bitboards[RotationIndex.FIRSTBIT] |= 1 << square
42            case _:
43                raise ValueError('Invalid FEN String - piece character not
44                followed by rotational character')
45
46        file += 1
47        elif character in '0123456789':
48            if character == '1' and fen_string[index + 1] == '0':
49                file += 10
50                continue
51
52        file += int(character)
53        elif character == '/':
54            rank = rank - 1
55            file = 0
56        elif character in Rotation:
57            continue
58        else:
59            raise ValueError('Invalid FEN String - invalid character found:',
60            character)

```

```

57     if piece_count['s'] != 1 or piece_count['S'] != 1:
58         raise ValueError('Invalid FEN string - invalid number of Sphinx pieces')
59     # COMMENTED OUT AS NO PHAROAH PIECES IS OKAY IF PARSING FEN STRING FOR
    FINISHED GAME BOARD THUMBNAIL
60     elif piece_count['f'] > 1 or piece_count['F'] > 1:
61         raise ValueError('Invalid FEN string - invalid number of Pharoah pieces')
62
63     if part_2 == 'b':
64         colour = Colour.BLUE
65     elif part_2 == 'r':
66         colour = Colour.RED
67     else:
68         raise ValueError('Invalid FEN string - invalid active colour')
69
70     for piece in Piece:
71         combined_colour_bitboards[Colour.BLUE] |= piece_bitboards[Colour.BLUE][
piece]
72         combined_colour_bitboards[Colour.RED] |= piece_bitboards[Colour.RED][piece
]
73
74     combined_all_bitboard = combined_colour_bitboards[Colour.BLUE] |
combined_colour_bitboards[Colour.RED]
75     return (piece_bitboards, combined_colour_bitboards, combined_all_bitboard,
rotation_bitboards, colour)
76
77 def encode_fen_string(bitboard_collection):
78     blue_bitboards = bitboard_collection.piece_bitboards[Colour.BLUE]
79     red_bitboards = bitboard_collection.piece_bitboards[Colour.RED]
80
81     fen_string_list = [''] * 80
82
83     for piece, bitboard in blue_bitboards.items():
84         for individual_bitboard in occupied_squares(bitboard):
85             index = bitboard_to_index(individual_bitboard)
86             rotation = bitboard_collection.get_rotation_on(individual_bitboard)
87             fen_string_list[index] = piece.upper() + rotation
88
89     for piece, bitboard in red_bitboards.items():
90         for individual_bitboard in occupied_squares(bitboard):
91             index = bitboard_to_index(individual_bitboard)
92             rotation = bitboard_collection.get_rotation_on(individual_bitboard)
93             fen_string_list[index] = piece.lower() + rotation
94
95     fen_string = ''
96     row_string = ''
97     empty_count = 0
98     for index, square in enumerate(fen_string_list):
99         if square == '':
100             empty_count += 1
101         else:
102             if empty_count > 0:
103                 row_string += str(empty_count)
104                 empty_count = 0
105
106             row_string += square
107
108     if index % 10 == 9:
109         if empty_count > 0:
110             fen_string = '/' + row_string + str(empty_count) + fen_string
111         else:
112             fen_string = '/' + row_string + fen_string
113

```

```

114         row_string = ''
115         empty_count = 0
116
117     fen_string = fen_string[1:]
118
119     if bitboard_collection.active_colour == Colour.BLUE:
120         colour = 'b'
121     else:
122         colour = 'r'
123
124     return fen_string + ' ' + colour

```

```

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
4
5  class Laser:
6      def __init__(self, bitboards):
7          self._bitboards = bitboards
8          self.hit_square_bitboard, self.piece_hit, self.laser_path, self.
   path_bitboard, self.pieces_on_trajectory = self.calculate_trajectory()
9
10         if (self.hit_square_bitboard != EMPTY_BB):
11             self.piece_rotation = self._bitboards.get_rotation_on(self.
   hit_square_bitboard)
12             self.piece_colour = self._bitboards.get_colour_on(self.
   hit_square_bitboard)
13
14     def calculate_trajectory(self):
15         current_square = self._bitboards.get_piece_bitboard(Piece.SPHINX, self.
   _bitboards.active_colour)
16         previous_direction = self._bitboards.get_rotation_on(current_square)
17         trajectory_bitboard = 0b0
18         trajectory_list = []
19         square_animation_states = []
20         pieces_on_trajectory = []
21
22         while current_square:
23             current_piece = self._bitboards.get_piece_on(current_square, Colour.
   BLUE) or self._bitboards.get_piece_on(current_square, Colour.RED)
24             current_rotation = self._bitboards.get_rotation_on(current_square)
25
26             next_square, direction, piece_hit = self.calculate_next_square(
   current_square, current_piece, current_rotation, previous_direction)
27
28             trajectory_bitboard |= current_square
29             trajectory_list.append(bb_helpers.bitboard_to_coords(current_square))
30             square_animation_states.append(direction)
31
32             if previous_direction != direction:
33                 pieces_on_trajectory.append(current_square)
34
35             if next_square == EMPTY_BB:
36                 hit_square_bitboard = 0b0
37
38                 if piece_hit:
39                     hit_square_bitboard = current_square
40
41             return hit_square_bitboard, piece_hit, list(zip(trajectory_list,
   square_animation_states)), trajectory_bitboard, pieces_on_trajectory
42

```

```

43         current_square = next_square
44         previous_direction = direction
45
46     def calculate_next_square(self, square, piece, rotation, previous_direction):
47         match piece:
48             case Piece.SPHINX:
49                 if previous_direction != rotation:
50                     return EMPTY_BB, previous_direction, None
51
52                 next_square = self.next_square_bitboard(square, rotation)
53                 return next_square, previous_direction, Piece.SPHINX
54
55             case Piece.PYRAMID:
56                 if previous_direction in [rotation, rotation.get_clockwise()]:
57                     return EMPTY_BB, previous_direction, Piece.PYRAMID
58
59                 if previous_direction == rotation.get_anticlockwise():
60                     new_direction = previous_direction.get_clockwise()
61                 else:
62                     new_direction = previous_direction.get_anticlockwise()
63
64                 next_square = self.next_square_bitboard(square, new_direction)
65
66                 return next_square, new_direction, None
67
68             case Piece.ANUBIS:
69                 if previous_direction == rotation.get_clockwise().get_clockwise():
70                     return EMPTY_BB, previous_direction, None
71
72                 return EMPTY_BB, previous_direction, Piece.ANUBIS
73
74             case Piece.SCARAB:
75                 if previous_direction in [rotation.get_clockwise(), rotation.
76 get_anticlockwise()]:
77                     new_direction = previous_direction.get_anticlockwise()
78                 else:
79                     new_direction = previous_direction.get_clockwise()
80
81                 next_square = self.next_square_bitboard(square, new_direction)
82
83                 return next_square, new_direction, None
84
85             case Piece.PHAROAH:
86                 return EMPTY_BB, previous_direction, Piece.PHAROAH
87
88             case None:
89                 next_square = self.next_square_bitboard(square, previous_direction
90 )
91
92                 return next_square, previous_direction, None
93
94     def next_square_bitboard(self, src_bitboard, previous_direction):
95         match previous_direction:
96             case Rotation.UP:
97                 masked_src_bitboard = src_bitboard & EIGHT_RANK_MASK
98                 return masked_src_bitboard << 10
99             case Rotation.RIGHT:
100                 masked_src_bitboard = src_bitboard & J_FILE_MASK
101                 return masked_src_bitboard << 1
102             case Rotation.DOWN:
103                 masked_src_bitboard = src_bitboard & ONE_RANK_MASK
104                 return masked_src_bitboard >> 10

```

```

103         case Rotation.LEFT:
104             masked_src_bitboard = src_bitboard & A_FILE_MASK
105             return masked_src_bitboard >> 1

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
9
10 type_to_image = {
11     LaserType.END: ['laser_end_1', 'laser_end_2'],
12     LaserType.STRAIGHT: ['laser_straight_1', 'laser_straight_2'],
13     LaserType.CORNER: ['laser_corner_1', 'laser_corner_2']
14 }
15
16 GLOW_SCALE_FACTOR = 1.5
17
18 class LaserDraw:
19     def __init__(self, board_position, board_size):
20         self._board_position = board_position
21         self._square_size = board_size[0] / 10
22         self._laser_lists = []
23
24     @property
25     def firing(self):
26         return len(self._laser_lists) > 0
27
28     def add_laser(self, laser_result, laser_colour):
29         """
30         Adds a laser to the board.
31
32         Args:
33             laser_result (Laser): Laser class instance containing laser trajectory
34             info.
35             laser_colour (Colour.RED | Colour.BLUE): Active colour of laser.
36         """
37         laser_path = laser_result.laser_path.copy()
38         laser_types = [LaserType.END]
39         # List of angles in degree to rotate the laser image surface when drawn
40         laser_rotation = [laser_path[0][1]]
41         laser_lights = []
42
43         # Iterates through every square laser passes through
44         for i in range(1, len(laser_path)):
45             previous_direction = laser_path[i-1][1]
46             current_coords, current_direction = laser_path[i]
47
48             if current_direction == previous_direction:
49                 laser_types.append(LaserType.STRAIGHT)
50                 laser_rotation.append(current_direction)
51             elif current_direction == previous_direction.get_clockwise():
52                 laser_types.append(LaserType.CORNER)
53                 laser_rotation.append(current_direction)
54             elif current_direction == previous_direction.get_anticlockwise():
55                 laser_types.append(LaserType.CORNER)
56                 laser_rotation.append(current_direction.get_anticlockwise())
57
58         # Adds a shader ray effect on the first and last square of the laser

```



```

trajectory
58     if i in [1, len(laser_path) - 1]:
59         abs_position = coords_to_screen_pos(current_coords, self.
_board_position, self._square_size)
60         laserLights.append([
61             (abs_position[0] / window.size[0], abs_position[1] / window.
size[1]),
62             0.35,
63             (0, 0, 255) if laser_colour == Colour.BLUE else (255, 0, 0),
64             ])
65
66     # Sets end laser draw type if laser hits a piece
67     if laser_result.hit_square_bitboard != EMPTY_BB:
68         laser_types[-1] = LaserType.END
69         laser_path[-1] = (laser_path[-1][0], laser_path[-2][1].get_opposite())
70         laser_rotation[-1] = laser_path[-2][1].get_opposite()
71
72         audio.play_sfx(SFX['piece_destroy'])
73
74     laser_path = [(coords, rotation, type) for (coords, dir), rotation, type
in zip(laser_path, laser_rotation, laser_types)]
75     self._laser_lists.append((laser_path, laser_colour))
76
77     window.clear_effect(ShaderType.RAYS)
78     window.set_effect(ShaderType.RAYS, lights=laserLights)
79     animation.set_timer(1000, self.remove_laser)
80
81     audio.play_sfx(SFX['laser_1'])
82     audio.play_sfx(SFX['laser_2'])
83
84     def remove_laser(self):
85         """
86         Removes a laser from the board.
87         """
88         self._laser_lists.pop(0)
89
90         if len(self._laser_lists) == 0:
91             window.clear_effect(ShaderType.RAYS)
92
93     def draw_laser(self, screen, laser_list, glow=True):
94         """
95         Draws every laser on the screen.
96
97         Args:
98             screen (pygame.Surface): The screen to draw on.
99             laser_list (list): The list of laser segments to draw.
100             glow (bool, optional): Whether to draw a glow effect. Defaults to True
.
101         """
102         laser_path, laser_colour = laser_list
103         laser_list = []
104         glow_list = []
105
106         for coords, rotation, type in laser_path:
107             square_x, square_y = coords_to_screen_pos(coords, self._board_position
, self._square_size)
108
109             image = GRAPHICS[type_to_image[type]][laser_colour]
110             rotated_image = pygame.transform.rotate(image, rotation.to_angle())
111             scaled_image = pygame.transform.scale(rotated_image, (self.
_square_size + 1, self._square_size + 1)) # +1 to prevent rounding creating
black lines

```

```

112         laser_list.append((scaled_image, (square_x, square_y)))
113
114         # Scales up the laser image surface as a glow surface
115         scaled_glow = pygame.transform.scale(rotated_image, (self._square_size
116 * GLOW_SCALE_FACTOR, self._square_size * GLOW_SCALE_FACTOR))
117         offset = self._square_size * ((GLOW_SCALE_FACTOR - 1) / 2)
118         glow_list.append((scaled_glow, (square_x - offset, square_y - offset))
119 )
120
121 # Scaled glow surfaces drawn on top with the RGB_ADD blend mode
122 if glow:
123     screen.fblits(glow_list, pygame.BLEND_RGB_ADD)
124
125 screen.blits(laser_list)
126
127 def draw(self, screen):
128     """
129     Draws all lasers on the screen.
130
131     Args:
132         screen (pygame.Surface): The screen to draw on.
133     """
134     for laser_list in self._laser_lists:
135         self.draw_laser(screen, laser_list)
136
137 def handle_resize(self, board_position, board_size):
138     """
139     Handles resizing of the board.
140
141     Args:
142         board_position (tuple[int, int]): The new position of the board.
143         board_size (tuple[int, int]): The new size of the board.
144     """
145     self._board_position = board_position
146     self._square_size = board_size[0] / 10
147
148 from data.constants import MoveType, Colour, RotationDirection
149 from data.utils.bitboard_helpers import notation_to_bitboard, coords_to_bitboard,
150     bitboard_to_coords, bitboard_to_notation, print_bitboard
151 import re
152 from data.managers.logs import initialise_logger
153
154 logger = initialise_logger(__name__)
155
156 class Move():
157     def __init__(self, move_type, src, dest=None, rotation_direction=None):
158         self.move_type = move_type
159         self.src = src
160         self.dest = dest
161         self.rotation_direction = rotation_direction
162
163     def to_notation(self, colour, piece, hit_square_bitboard):
164         hit_square = ''
165         if colour == Colour.BLUE:
166             piece = piece.upper()
167
168         if hit_square_bitboard:
169             hit_square = 'x' + bitboard_to_notation(hit_square_bitboard)
170
171         if self.move_type == MoveType.MOVE:
172             return 'M' + piece + bitboard_to_notation(self.src) +
173                 bitboard_to_notation(self.dest) + hit_square

```

```

25         else:
26             return 'R' + piece + bitboard_to_notation(self.src) + self.
rotation_direction + hit_square
27
28     def __str__(self):
29         rotate_text = ''
30         coords_1 = '(' + chr(bitboard_to_coords(self.src)[0] + 65) + ',' + str(
bitboard_to_coords(self.src)[1] + 1) + ')'
31
32         if self.move_type == MoveType.ROTATE:
33             rotate_text = ' ' + self.rotation_direction.name
34             return f'{self.move_type.name}{rotate_text}: ON {coords_1}'
35
36         elif self.move_type == MoveType.MOVE:
37             coords_2 = '(' + chr(bitboard_to_coords(self.dest)[0] + 65) + ',' + str(
bitboard_to_coords(self.dest)[1] + 1) + ')'
38             return f'{self.move_type.name}{rotate_text}: FROM {coords_1} TO {
coords_2}'
39
40         # (Rotation: {self.rotation_direction})
41
42     @classmethod
43     def instance_from_notation(move_cls, notation):
44         try:
45             notation = notation.split('x')[0]
46             move_type = notation[0].lower()
47
48             moves = notation[2:]
49             letters = re.findall(r'[A-Za-z]+', moves)
50             numbers = re.findall(r'\d+', moves)
51
52             if move_type == MoveType.MOVE:
53                 src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
54                 dest_bitboard = notation_to_bitboard(letters[1] + numbers[1])
55
56                 return move_cls(move_type, src_bitboard, dest_bitboard)
57
58             elif move_type == MoveType.ROTATE:
59                 src_bitboard = notation_to_bitboard(letters[0] + numbers[0])
60                 rotation_direction = RotationDirection(letters[1])
61
62                 return move_cls(move_type, src_bitboard, src_bitboard,
rotation_direction)
63         else:
64             raise ValueError('(Move.instance_from_notation) Invalid move type:
', move_type)
65
66         except Exception as error:
67             logger.info('(Move.instance_from_notation) Error occured while parsing
:', error)
68             raise error
69
70     @classmethod
71     def instance_from_input(move_cls, move_type, src, dest=None, rotation=None):
72         try:
73             if move_type == MoveType.MOVE:
74                 src_bitboard = notation_to_bitboard(src)
75                 dest_bitboard = notation_to_bitboard(dest)
76
77             elif move_type == MoveType.ROTATE:
78                 src_bitboard = notation_to_bitboard(src)
79                 dest_bitboard = src_bitboard

```

```

80
81         return move_cls(move_type, src_bitboard, dest_bitboard, rotation)
82     except Exception as error:
83         logger.info('Error (Move.instance_from):', error)
84         raise error
85
86     @classmethod
87     def instance_from_coords(move_cls, move_type, src_coords, dest_coords=None,
88                             rotation_direction=None):
89         try:
90             src_bitboard = coords_to_bitboard(src_coords)
91             dest_bitboard = coords_to_bitboard(dest_coords)
92
93             return move_cls(move_type, src_bitboard, dest_bitboard,
94                             rotation_direction)
95         except Exception as error:
96             logger.info('Error (Move.instance_from_coords):', error)
97             raise error
98
99     @classmethod
100     def instance_from_bitboards(move_cls, move_type, src_bitboard, dest_bitboard=
101                                None, rotation_direction=None):
102         try:
103             return move_cls(move_type, src_bitboard, dest_bitboard,
104                             rotation_direction)
105         except Exception as error:
106             logger.info('Error (Move.instance_from_bitboards):', error)
107             raise error
108
109
110 1 import pygame
111 2 from data.constants import OVERLAY_COLOUR_LIGHT, OVERLAY_COLOUR_DARK
112 3 from data.utils.board_helpers import coords_to_screen_pos, screen_pos_to_coords,
113   create_square_overlay, create_circle_overlay
114
115 4
116 5 class OverlayDraw:
117 6     def __init__(self, board_position, board_size, limit_hover=True):
118 7         self._board_position = board_position
119 8         self._board_size = board_size
120
121 9
122 10         self._hovered_coords = None
123 11         self._selected_coords = None
124 12         self._available_coords = None
125
126 13
127 14         self._limit_hover = limit_hover
128
129 15
130 16         self._selected_overlay = None
131 17         self._hovered_overlay = None
132 18         self._available_overlay = None
133
134 19
135 20         self.initialise_overlay_surfaces()
136
137 21
138 22     @property
139 23     def square_size(self):
140 24         return self._board_size[0] / 10
141
142 25
143 26     def initialise_overlay_surfaces(self):
144 27         self._selected_overlay = create_square_overlay(self.square_size,
145 OVERLAY_COLOUR_DARK)
146 28         self._hovered_overlay = create_square_overlay(self.square_size,
147 OVERLAY_COLOUR_LIGHT)
148 29         self._available_overlay = create_circle_overlay(self.square_size,
149 OVERLAY_COLOUR_LIGHT)

```

```

30
31     def set_hovered_coords(self, mouse_pos):
32         self._hovered_coords = screen_pos_to_coords(mouse_pos, self.
33             _board_position, self._board_size)
34
35     def set_selected_coords(self, coords):
36         self._selected_coords = coords
37
38     def set_available_coords(self, coords_list):
39         self._available_coords = coords_list
40
41     def set_hover_limit(self, new_limit):
42         self._limit_hover = new_limit
43
44     def draw(self, screen):
45         self.set_hovered_coords(pygame.mouse.get_pos())
46
47         if self._selected_coords:
48             screen.blit(self._selected_overlay, coords_to_screen_pos(self.
49                 _selected_coords, self._board_position, self.square_size))
50
51         if self._available_coords:
52             for coords in self._available_coords:
53                 screen.blit(self._available_overlay, coords_to_screen_pos(coords,
54                     self._board_position, self.square_size))
55
56         if self._hovered_coords:
57             if self._hovered_coords is None:
58                 return
59
60             if self._limit_hover and ((self._available_coords is None) or (self.
61                 _hovered_coords not in self._available_coords)):
62                 return
63
64             screen.blit(self._hovered_overlay, coords_to_screen_pos(self.
65                 _hovered_coords, self._board_position, self.square_size))
66
67     def handle_resize(self, board_position, board_size):
68         self._board_position = board_position
69         self._board_size = board_size
70
71         self.initialise_overlay_surfaces()
72
73
74 1 import pygame
75 2 from random import randint
76 3 from data.utils.asset_helpers import get_perimeter_sample, get_vector,
77   get_angle_between_vectors, get_next_corner
78 4 from data.states.game.components.piece_sprite import PieceSprite
79 5
80 6 class ParticlesDraw:
81 7     def __init__(self, gravity=0.2, rotation=180, shrink=0.5, opacity=150):
82 8         self._particles = []
83 9         self._glow_particles = []
84 10
85 11         self._gravity = gravity
86 12         self._rotation = rotation
87 13         self._shrink = shrink
88 14         self._opacity = opacity
89 15
90 16     def fragment_image(self, image, number):
91 17         image_size = image.get_rect().size
92 18         """

```

```

19     1. Takes an image surface and samples random points on the perimeter.
20     2. Iterates through points, and depending on the nature of two consecutive
    points, finds a corner between them.
21     3. Draws a polygon with the points as the vertices to mask out the area
    not in the fragment.
22
23     Args:
24         image (pygame.Surface): Image to fragment.
25         number (int): The number of fragments to create.
26
27     Returns:
28         list[pygame.Surface]: List of image surfaces with fragment of original
    surface drawn on top.
29     """
30     center = image.get_rect().center
31     points_list = get_perimeter_sample(image_size, number)
32     fragment_list = []
33
34     points_list.append(points_list[0])
35
36     # Iterate through points_list, using the current point and the next one
37     for i in range(len(points_list) - 1):
38         vertex_1 = points_list[i]
39         vertex_2 = points_list[i + 1]
40         vector_1 = get_vector(center, vertex_1)
41         vector_2 = get_vector(center, vertex_2)
42         angle = get_angle_between_vectors(vector_1, vector_2)
43
44         cropped_image = pygame.Surface(image_size, pygame.SRCALPHA)
45         cropped_image.fill((0, 0, 0, 0))
46         cropped_image.blit(image, (0, 0))
47
48         corners_to_draw = None
49
50         if vertex_1[0] == vertex_2[0] or vertex_1[1] == vertex_2[1]: # Points
    on the same side
51             corners_to_draw = 4
52
53             elif abs(vertex_1[0] - vertex_2[0]) == image_size[0] or abs(vertex_1
    [1] - vertex_2[1]) == image_size[1]: # Points on opposite sides
54                 corners_to_draw = 2
55
56             elif angle < 180: # Points on adjacent sides
57                 corners_to_draw = 3
58
59             else:
60                 corners_to_draw = 1
61
62             corners_list = []
63             for j in range(corners_to_draw):
64                 if len(corners_list) == 0:
65                     corners_list.append(get_next_corner(vertex_2, image_size))
66                 else:
67                     corners_list.append(get_next_corner(corners_list[-1],
    image_size))
68
69             pygame.draw.polygon(cropped_image, (0, 0, 0, 0), (center, vertex_2, *
    corners_list, vertex_1))
70
71             fragment_list.append(cropped_image)
72
73     return fragment_list

```

```

74
75 def add_captured_piece(self, piece, colour, rotation, position, size):
76     """
77     Adds a captured piece to fragment into particles.
78
79     Args:
80         piece (Piece): The piece type.
81         colour (Colour): The active colour of the piece.
82         rotation (int): The rotation of the piece.
83         position (tuple[int, int]): The position where particles originate
84 from.
85         size (tuple[int, int]): The size of the piece.
86     """
87     piece_sprite = PieceSprite(piece, colour, rotation)
88     piece_sprite.set_geometry((0, 0), size)
89     piece_sprite.set_image()
90
91     particles = self.fragment_image(piece_sprite.image, 5)
92
93     for particle in particles:
94         self.add_particle(particle, position)
95
96 def add_sparks(self, radius, colour, position):
97     """
98     Adds laser spark particles.
99
100    Args:
101        radius (int): The radius of the sparks.
102        colour (Colour): The active colour of the sparks.
103        position (tuple[int, int]): The position where particles originate
104 from.
105    """
106    for i in range(randint(10, 15)):
107        velocity = [randint(-15, 15) / 10, randint(-20, 0) / 10]
108        random_colour = [min(max(val + randint(-20, 20), 0), 255) for val in
109            colour]
110        self._particles.append([None, [radius, random_colour], [*position],
111            velocity, 0])
112
113 def add_particle(self, image, position):
114     """
115     Adds a particle.
116
117     Args:
118         image (pygame.Surface): The image of the particle.
119         position (tuple): The position of the particle.
120     """
121     velocity = [randint(-15, 15) / 10, randint(-20, 0) / 10]
122
123     # Each particle is stored with its attributes: [surface, copy of surface,
124     position, velocity, lifespan]
125     self._particles.append([image, image.copy(), [*position], velocity, 0])
126
127 def update(self):
128     """
129     Updates each particle and its attributes.
130     """
131     for i in range(len(self._particles) - 1, -1, -1):
132         particle = self._particles[i]
133
134         #update position
135         particle[2][0] += particle[3][0]

```

```

131         particle[2][1] += particle[3][1]
132
133     #update lifespan
134     self._particles[i][4] += 0.01
135
136     if self._particles[i][4] >= 1:
137         self._particles.pop(i)
138         continue
139
140     if isinstance(particle[1], pygame.Surface): # Particle is a piece
141         # Update velocity
142         particle[3][1] += self._gravity
143
144         # Update size
145         image_size = particle[1].get_rect().size
146         end_size = ((1 - self._shrink) * image_size[0], (1 - self._shrink)
147 * image_size[1])
148         target_size = (image_size[0] - particle[4] * (image_size[0] -
149 end_size[0]), image_size[1] - particle[4] * (image_size[1] - end_size[1]))
150
151         # Update rotation
152         rotation = (self._rotation if particle[3][0] <= 0 else -self.
153 _rotation) * particle[4]
154
155         updated_image = pygame.transform.scale(pygame.transform.rotate(
156 particle[1], rotation), target_size)
157
158     elif isinstance(particle[1], list): # Particle is a spark
159         # Update size
160         end_radius = (1 - self._shrink) * particle[1][0]
161         target_radius = particle[1][0] - particle[4] * (particle[1][0] -
162 end_radius)
163
164         updated_image = pygame.Surface((target_radius * 2, target_radius *
165 2), pygame.SRCALPHA)
166         pygame.draw.circle(updated_image, particle[1][1], (target_radius,
167 target_radius), target_radius)
168
169         # Update opacity
170         alpha = 255 - particle[4] * (255 - self._opacity)
171
172         updated_image.fill((255, 255, 255, alpha), None, pygame.
173 BLEND_RGBA_MULT)
174
175         particle[0] = updated_image
176
177 def draw(self, screen):
178     """
179     Draws the particles, indexing the surface and position attributes for each
180     particle.
181
182     Args:
183         screen (pygame.Surface): The screen to draw on.
184     """
185     screen.blits([
186         (particle[0], particle[2]) for particle in self._particles
187     ])
188
189 1 import pygame
190 2 from data.constants import EMPTY_BB, Colour, Piece
191 3 from data.states.game.components.piece_sprite import PieceSprite
192 4 from data.utils.board_helpers import coords_to_screen_pos

```



```

5 from data.utils import bitboard_helpers as bb_helpers
6
7 class PieceGroup(pygame.sprite.Group):
8     def __init__(self):
9         # self.square_list = []
10        # self.valid_square_list_positions = []
11        super().__init__()
12
13    def initialise_pieces(self, piece_list, board_position, board_size):
14        self.empty()
15
16        for index, piece_and_rotation in enumerate(piece_list):
17            x = index % 10
18            y = index // 10
19
20            if piece_and_rotation:
21                if piece_and_rotation[0].isupper():
22                    colour = Colour.BLUE
23                else:
24                    colour = Colour.RED
25
26                piece = PieceSprite(piece=Piece(piece_and_rotation[0].lower()),
27                                     colour=colour, rotation=piece_and_rotation[1])
28                piece.set_coords((x, y))
29                piece.set_geometry(board_position, board_size[0] / 10)
30                piece.set_image()
31                self.add(piece)
32
33    def set_geometry(self, board_position, board_size):
34        for sprite in self.sprites():
35            sprite.set_geometry(board_position, board_size[0] / 10)
36
37    def handle_resize(self, board_position, board_size):
38        self.set_geometry(board_position, board_size)
39
40        for sprite in self.sprites():
41            sprite.set_image()
42
43    def remove_piece(self, coords):
44        for sprite in self.sprites():
45            if sprite.coords == coords:
46                sprite.kill()
47
48    # def handle_resize_end(self):
49    #     for sprite in self.sprites():
50    #         sprite.handle_resize_end()
51
52    # def clear_square(self, src_bitboard):
53    #     list_position = bb_helpers.bitboard_to_index(src_bitboard)
54    #     self.square_list[list_position].clear_piece()
55
56    # def update_squares_move(self, src, dest, new_piece_symbol, new_colour,
57    #                           rotation):
58    #     self.square_list[src].clear_piece()
59    #     self.square_list[dest].clear_piece()
60    #     self.square_list[dest].set_piece(piece_symbol=new_piece_symbol, colour=
61    # new_colour, rotation=rotation)
62
63    # def update_squares_rotate(self, src, piece_symbol, colour, new_rotation):
64    #     self.square_list[src].clear_piece()
65    #     self.square_list[src].set_piece(piece_symbol=piece_symbol, colour=colour
66    # , rotation=new_rotation)

```

```

63
64     # def add_valid_square_overlays(self, valid_bitboard):
65     #     if valid_bitboard == EMPTY_BB:
66     #         return
67
68     #     list_positions = self.bitboard_to_list_positions(valid_bitboard)
69     #     self.valid_square_list_positions = list_positions
70
71     #     for square_position in list_positions:
72     #         square = self.square_list[square_position]
73     #         square.selected = True
74
75     # def remove_valid_square_overlays(self):
76     #     for square_position in self.valid_square_list_positions:
77     #         square = self.square_list[square_position]
78     #         square.selected = False
79     #         square.remove_overlay()
80
81     #     self.valid_square_list_positions = []
82
83     # def draw_valid_square_overlays(self):
84     #     for square_position in self.valid_square_list_positions:
85     #         square = self.square_list[square_position]
86     #         square.draw_overlay()
87
88     # def bitboard_to_list_positions(self, bitboard):
89     #     list_positions = []
90
91     #     for square in bb_helpers.occupied_squares(bitboard):
92     #         list_positions.append(bb_helpers.bitboard_to_index(square))
93
94     #     return list_positions

```

```

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
6
7  class EmptyPiece(pygame.sprite.Sprite):
8      def __init__(self):
9          super().__init__()
10
11          self.image = pygame.Surface((1, 1))
12          self.rect = self.image.get_rect()
13          self.rect.topleft = (0, 0)
14
15      def set_image(self, type):
16          pass
17
18      def set_rect(self):
19          pass
20
21      def set_geometry(self, anchor_position, size):
22          pass
23
24  class PieceSprite(pygame.sprite.Sprite):
25      def __init__(self, piece, colour, rotation):
26          super().__init__()
27          self.colour = colour
28          self.rotation = rotation
29

```

```

30         self.type = piece
31         self.coords = None
32         self.size = None
33
34     @property
35     def image_name(self):
36         return Piece(self.type).name.lower() + '_' + str(self.colour) + '_' + self
           .rotation
37
38     def set_image(self):
39         self.image = scale_and_cache(GRAPHICS[self.image_name], (self.size, self.
           size))
40
41     def set_geometry(self, new_position, square_size):
42         self.size = square_size
43         self.rect = pygame.FRect((0, 0, square_size, square_size))
44
45         if self.coords:
46             self.rect.topleft = coords_to_screen_pos(self.coords, new_position,
           square_size)
47         else:
48             self.rect.topleft = new_position
49
50     def set_coords(self, new_coords):
51         self.coords = new_coords
52
53
54 1 from data.constants import Piece
55 2
56 3 FLIP = [
57 4     70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
58 5     60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
59 6     50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60 7     40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
61 8     6, 31, 32, 33, 34, 35, 36, 37, 38, 39,
62 9     4, 21, 22, 23, 24, 25, 26, 27, 28, 29,
63 10    2, 11, 12, 13, 14, 3, 16, 17, 18, 19,
64 11    0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
65 12 ]
66 13
67 14 PSQT = {
68 15     Piece.PYRAMID: [
69 16         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
70 17         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
71 18         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
72 19         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
73 20         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
74 21         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
75 22         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
76 23         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
77 24     ],
78 25     Piece.ANUBIS: [
79 26         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
80 27         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
81 28         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
82 29         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
83 30         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
84 31         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
85 32         4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
86 33         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
87 34     ],
88 35     Piece.SCARAB: [
89 36         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

```

37         0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
38         0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
39         0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
40         0, 0, 1, 2, 3, 3, 2, 1, 0, 0,
41         0, 0, 1, 2, 2, 2, 2, 1, 0, 0,
42         0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
43         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
44     ],
45     Piece.PHAROAH: [
46         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
47         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
48         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
49         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
50         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
51         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
52         0, 0, 0, 2, 2, 2, 2, 0, 0, 0,
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
5
6  logger = initialise_logger(__name__)
7  # sc3ncfcncpb2/2pc7/3Pd6/pa1Pclrbraipb1Pd/pb1Pd1RaRb1pa1Pc/6pb3/7Pa2/2PdNaFaNa3Sa
8  # scfaRa7/RaRaRaFa6/RaRaRa7/10/10/10/10/9Sa b
9  # scfa8/10/10/10/10/10/10/8FaSa b
10
11 def compare(cls1, cls2, depth, rounds):
12     wins = [0, 0]
13
14     board = Board()
15     def callback(move):
16         board.apply_move(move, add_hash=True)
17
18     cpu1 = cls1(callback=callback, max_depth=depth, verbose='compact')
19     cpu2 = cls2(callback=callback, max_depth=depth, verbose='compact')
20
21     for i in range(rounds):
22         board = Board(fen_string="scfa8/10/10/10/10/10/10/8FaSa b")
23         ply = 0
24
25         if i % 2 == 0:
26             players = { Colour.BLUE: cpu1, Colour.RED: cpu2, Miscellaneous.DRAW: '
DRAW' }
27         else:
28             players = { Colour.BLUE: cpu2, Colour.RED: cpu1, Miscellaneous.DRAW: '
DRAW' }
29
30         while (winner := board.check_win()) is None:
31             players[board.get_active_colour()].find_move(board, None)
32             ply += 1
33             logger.debug('PLY:', ply)
34
35         if winner == Miscellaneous.DRAW:
36             wins[0] += 0.5
37             wins[1] += 0.5
38         else:
39             if players[winner] == cpu1:

```

```

40         wins[0] += 1
41     else:
42         wins[1] += 1
43
44     logger.debug(f'ROUND {i + 1} | WINNER: {players[winner]} | PLY: {ply}')
45
46     logger.debug(f'{cpu1} SCORE: {wins[0]} | {cpu2} SCORE: {wins[1]}')
47
48     compare(TTNegamaxCPU, TTNegamaxCPU, 2, 1)
49
50
51 import time
52 from pprint import PrettyPrinter
53 from data.constants import Colour, Score, Miscellaneous
54 from data.states.game.cpu.evaluator import Evaluator
55 from data.managers.logs import initialise_logger
56
57 logger = initialise_logger(__name__)
58 printer = PrettyPrinter(indent=2, sort_dicts=False)
59
60 class BaseCPU:
61     def __init__(self, callback, verbose=True):
62         self._evaluator = Evaluator(verbose=False)
63         self._verbose = verbose
64         self._callback = callback
65         self._stats = {}
66
67     def initialise_stats(self):
68         self._stats = {
69             'nodes': 0,
70             'leaf_nodes': 0,
71             'draws': 0,
72             'mates': 0,
73             'ms_per_node': 0,
74             'time_taken': time.time()
75         }
76
77     def print_stats(self, score, move):
78         """
79         Prints statistics after traversing tree.
80
81         Args:
82             score (int): Final score obtained after traversal.
83             move (Move): Best move obtained after traversal.
84         """
85         if self._verbose is False:
86             return
87
88         self._stats['time_taken'] = round(1000 * (time.time() - self._stats['
89 time_taken']), 3)
90         self._stats['ms_per_node'] = round(self._stats['time_taken'] / self._stats
91 ['nodes'], 3)
92
93         # Prints stats across multiple lines
94         if self._verbose is True:
95             logger.info(f'\n\n'
96                         f'{self.__str__()} Search Results:\n'
97                         f'{printer.pformat(self._stats)}\n'
98                         f'Best score: {score}    Best move: {move}\n'
99                         )
100
101         # Prints stats in a compacted format
102         elif self._verbose.lower() == 'compact':

```

```

51         logger.info(self._stats)
52         logger.info(f'Best score: {score}    Best move: {move}')
53
54     def find_move(self, board, stop_event=None):
55         raise NotImplementedError
56
57     def search(self, board, depth, stop_event, absolute=False, **kwargs):
58         if stop_event and stop_event.is_set():
59             raise Exception(f'Thread killed - stopping minimax function ({self.
__str__}.search)')
60
61         self._stats['nodes'] += 1
62
63         if (winner := board.check_win()) is not None:
64             self._stats['leaf_nodes'] += 1
65             return self.process_win(winner, depth, absolute)
66
67         if depth == 0:
68             self._stats['leaf_nodes'] += 1
69             return self._evaluator.evaluate(board, absolute), None
70
71     def process_win(self, winner, depth, absolute):
72         self._stats['leaf_nodes'] += 1
73
74         if winner == Miscellaneous.DRAW:
75             self._stats['draws'] += 1
76             return 0, None
77         elif winner == Colour.BLUE or absolute:
78             self._stats['mates'] += 1
79             return Score.CHECKMATE + depth, None
80         elif winner == Colour.RED:
81             self._stats['mates'] += 1
82             return -Score.CHECKMATE - depth, None
83
84     def __str__(self):
85         return self.__class__.__name__

```

  

```

1  import threading
2  import time
3  from data.managers.logs import initialise_logger
4
5  logger = initialise_logger(__name__)
6
7  class CPUThread(threading.Thread):
8      def __init__(self, cpu, verbose=False):
9          super().__init__()
10         self._stop_event = threading.Event()
11         self._running = True
12         self._verbose = verbose
13         self.daemon = True
14
15         self._board = None
16         self._cpu = cpu
17
18     def kill_thread(self):
19         """
20         Kills the CPU and terminates the thread by stopping the run loop.
21         """
22         self.stop_cpu()
23         self._running = False
24
25     def stop_cpu(self):

```

```

26         """
27         Kills the CPU's move search.
28         """
29         self._stop_event.set()
30         self._board = None
31
32     def start_cpu(self, board):
33         """
34         Starts the CPU's move search.
35
36         Args:
37             board (Board): The current board state.
38         """
39         self._stop_event.clear()
40         self._board = board
41
42     def run(self):
43         """
44         Periodically checks if the board variable is set.
45         If it is, then starts CPU search.
46         """
47         while self._running:
48             if self._board and self._cpu:
49                 self._cpu.find_move(self._board, self._stop_event)
50                 self.stop_cpu()
51             else:
52                 time.sleep(1)
53                 if self._verbose:
54                     logger.debug(f'(CPUThread.run) Thread {threading.get_native_id
55                                 ({} idling...}')
56
57 1 from data.utils.bitboard_helpers import pop_count, occupied_squares,
58   bitboard_to_index
59 2 from data.states.game.components.psqt import PSQT, FLIP
60 3 from data.managers.logs import initialise_logger
61 4 from data.constants import Colour, Piece, Score
62
63 5
64 6 logger = initialise_logger(__name__)
65
66 7
67 8 class Evaluator:
68 9     def __init__(self, verbose=True):
69 10         self._verbose = verbose
70
71 11
72 12     def evaluate(self, board, absolute=False):
73 13         """
74 14         Evaluates and returns a numerical score for the board state.
75
76 15
77 16         Args:
78 17             board (Board): The current board state.
79 18             absolute (bool): Whether to always return the absolute score from the
80 19                             active colour's perspective (for NegaMax).
81
82 20
83 21         Returns:
84 22             int: Score representing advantage/disadvantage for the player.
85 23         """
86 24         blue_score = (
87 25             self.evaluate_material(board, Colour.BLUE),
88 26             self.evaluate_position(board, Colour.BLUE),
89 27             self.evaluate_mobility(board, Colour.BLUE),
90 28             self.evaluate_pharoah_safety(board, Colour.BLUE)
91 29         )

```

```

30     red_score = (
31         self.evaluate_material(board, Colour.RED),
32         self.evaluate_position(board, Colour.RED),
33         self.evaluate_mobility(board, Colour.RED),
34         self.evaluate_pharoah_safety(board, Colour.RED)
35     )
36
37     if self._verbose:
38         logger.info(f'Material: {blue_score[0]} | {red_score[0]}')
39         logger.info(f'Position: {blue_score[1]} | {red_score[1]}')
40         logger.info(f'Mobility: {blue_score[2]} | {red_score[2]}')
41         logger.info(f'Safety: {blue_score[3]} | {red_score[3]}')
42         logger.info(f'Overall score: {sum(blue_score) - sum(red_score)}')
43
44     if absolute and board.get_active_colour() == Colour.RED:
45         return sum(red_score) - sum(blue_score)
46     else:
47         return sum(blue_score) - sum(red_score)
48
49 def evaluate_material(self, board, colour):
50     """
51     Evaluates the material score for a given colour.
52
53     Args:
54         board (Board): The current board state.
55         colour (Colour): The colour to evaluate.
56
57     Returns:
58         int: Sum of all piece scores.
59     """
60     return (
61         Score.SPHINX * board.bitboards.get_piece_count(Piece.SPHINX, colour) +
62         Score.PYRAMID * board.bitboards.get_piece_count(Piece.PYRAMID, colour)
63     +
64         Score.ANUBIS * board.bitboards.get_piece_count(Piece.ANUBIS, colour) +
65         Score.SCARAB * board.bitboards.get_piece_count(Piece.SCARAB, colour)
66     )
67
68 def evaluate_position(self, board, colour):
69     """
70     Evaluates the positional score for a given colour.
71
72     Args:
73         board (Board): The current board state.
74         colour (Colour): The colour to evaluate.
75
76     Returns:
77         int: Score representing positional advantage/disadvantage.
78     """
79     score = 0
80
81     for piece in Piece:
82         if piece == Piece.SPHINX:
83             continue
84
85         piece_bitboard = board.bitboards.get_piece_bitboard(piece, colour)
86
87         for bitboard in occupied_squares(piece_bitboard):
88             index = bitboard_to_index(bitboard)
89             # Flip PSQT if using from blue player's perspective
90             index = FLIP[index] if colour == Colour.BLUE else index

```



```

91         score += PSQT[piece][index] * Score.POSITION
92
93     return score
94
95     def evaluate_mobility(self, board, colour):
96         """
97         Evaluates the mobility score for a given colour.
98
99         Args:
100             board (Board): The current board state.
101             colour (Colour): The colour to evaluate.
102
103         Returns:
104             int: Score on numerical representation of mobility.
105         """
106         number_of_moves = board.get_mobility(colour)
107         return number_of_moves * Score.MOVE
108
109     def evaluate_pharoah_safety(self, board, colour):
110         """
111         Evaluates the safety of the Pharoah for a given colour.
112
113         Args:
114             board (Board): The current board state.
115             colour (Colour): The colour to evaluate.
116
117         Returns:
118             int: Score representing mobility of the Pharoah.
119         """
120         pharoah_bitboard = board.bitboards.get_piece_bitboard(Piece.PHAROAH,
121             colour)
122
123         if pharoah_bitboard:
124             pharoah_available_moves = pop_count(board.get_valid_squares(
125                 pharoah_bitboard, colour))
126             return (8 - pharoah_available_moves) * Score.PHAROAH_SAFETY
127         else:
128             return 0
129
130
131 1 from data.states.game.cpu.evaluator import Evaluator
132 2 from data.constants import Colour
133 3 from data.utils.bitboard_helpers import print_bitboard, pop_count
134 4
135 5 class SimpleEvaluator:
136 6     def __init__(self):
137 7         self._evaluator = Evaluator(verbose=False)
138 8         self._cache = {}
139 9
140 10     def evaluate(self, board):
141 11         if (hashed := board.to_hash()) in self._cache:
142 12             return self._cache[hashed]
143 13
144 14         score = self._evaluator.evaluate_material(board, board.get_active_colour()
145 15     )
146 16         self._cache[hashed] = score
147 17
148 18         return score
149 19
150 19 class MoveOrderer:
151 20     def __init__(self):
152 21         self._evaluator = SimpleEvaluator()
153 22

```

```

23     # def get_eval(self, board, move):
24     #     laser_result = board.apply_move(move)
25     #     score = self._evaluator.evaluate(board)
26     #     board.undo_move(move, laser_result)
27     #     return score
28
29     # def score_moves(self, board, moves):
30     #     for i in range(len(moves)):
31     #         score = self.get_eval(board, moves[i])
32     #         moves[i] = (moves[i], score)
33
34     #     return moves
35
36     def best_move_to_front(self, moves, start_idx, hint):
37         for i in range(start_idx + 1, len(moves)):
38             if moves[i].src in hint:
39                 moves[i], moves[start_idx] = moves[start_idx], moves[i]
40             return
41
42     def get_moves(self, board, hint=None):
43         colour = board.get_active_colour()
44         moves = list(board.generate_all_moves(colour))
45
46         for i in range(len(moves)):
47             if hint:
48                 self.best_move_to_front(moves, i, hint)
49
50         yield moves[i]
51
52
53 1 from data.constants import Score, Colour
54 2 from data.states.game.cpu.transposition_table import TranspositionTable
55 3 from data.states.game.cpu.base import BaseCPU
56 4 from pprint import pprint
57
58 5
59 6 class MinimaxCPU(BaseCPU):
60 7     def __init__(self, max_depth, callback, verbose):
61 8         super().__init__(callback, verbose)
62 9         self._max_depth = max_depth
63
64 10
65 11     def find_move(self, board, stop_event):
66 12         # No bit_length bug as None type returned, so Move __str__ called on
67 13         # NoneType I think (just deal with None being returned)
68 14         try:
69 15             best_move = self.search(board, self._max_depth, -Score.INFINITE, Score
70 16             .INFINITE, stop_event)
71
72 17             if self._verbose:
73 18                 print('\nCPU Search Results:')
74 19                 pprint(self._stats)
75 20                 print('Best move:', best_move, '\n')
76
77 21                 self._callback(self._best_move)
78 22         except Exception as error:
79 23             print('(MinimaxBase.find_move) Error has occurred:')
80 24             raise error
81
82 25
83 26     def search(self, board, depth, alpha, beta, stop_event):
84 27         if stop_event.is_set():
85 28             raise Exception('Thread killed - stopping minimax function (CPU.
86 29             minimax)')
87
88 30         # cached_move, cached_score = self._transposition_table.get_entry(hash_key

```

```

31     =board.bitboards.get_hash(), depth=depth, alpha=alpha, beta=beta)
32     # if cached_move or cached_score:
33     #     if depth == self._max_depth:
34     #         self._best_move = cached_move
35     #     return cached_score
36
37     if depth == 0:
38         return self.evaluate(board)
39
40     is_maximiser = board.get_active_colour() == Colour.BLUE
41
42     if is_maximiser:
43         score = -Score.INFINITE
44
45         for move in board.generate_all_moves(board.get_active_colour()):
46             before, before_score = board.bitboards.get_rotation_string(), self
47             .evaluate(board)
48
49             laser_result = board.apply_move(move)
50             new_score = self.minimax(board, depth - 1, alpha, beta, False,
51             stop_event)
52
53             if new_score >= score:
54                 score = new_score
55
56             if depth == self._max_depth:
57                 self._best_move = move
58
59             board.undo_move(move, laser_result)
60
61             alpha = max(alpha, score)
62             if depth == self._max_depth: # https://stackoverflow.com/questions/31429974/alphabeta-pruning-alpha-equals-or-greater-than-beta-why-equals
63                 if beta < alpha:
64                     break
65             else:
66                 if beta <= alpha:
67                     break
68
69             after, after_score = board.bitboards.get_rotation_string(), self.
70             evaluate(board)
71             if (before != after or before_score != after_score):
72                 print('shit\n\n')
73
74             return score
75
76     else:
77         score = Score.INFINITE
78
79         for move in board.generate_all_moves(board.get_active_colour()):
80             bef, before_score = board.bitboards.get_rotation_string(), self.
81             evaluate(board)
82
83             laser_result = board.apply_move(move)
84             new_score = self.minimax(board, depth - 1, alpha, beta, False,
85             stop_event)
86
87             if new_score <= score:
88                 score = new_score
89
90             if depth == self._max_depth:
91                 self._best_move = move

```

```

86         board.undo_move(move, laser_result)
87
88         beta = min(beta, score)
89         if depth == self._max_depth:
90             if beta < alpha:
91                 break
92         else:
93             if beta <= alpha:
94                 break
95
96         after, after_score = board.bitboards.get_rotation_string(), self.
evaluate(board)
97         if (bef != after or before_score != after_score):
98             print('shit\n\n')
99             raise ValueError
100
101     return score
102
103
104 from data.constants import TranspositionFlag
105
106 class TranspositionEntry:
107     def __init__(self, score, move, flag, hash_key, depth):
108         self.score = score
109         self.move = move
110         self.flag = flag
111         self.hash_key = hash_key
112         self.depth = depth
113
114 class TranspositionTable:
115     def __init__(self, max_entries=100000):
116         self._max_entries = max_entries
117         self._table = dict()
118
119     def calculate_entry_index(self, hash_key):
120         """
121         Gets the dictionary key for a given Zobrist hash.
122
123         Args:
124             hash_key (int): A Zobrist hash.
125
126         Returns:
127             int: Key for the given hash.
128         """
129         # return hash_key % self._max_entries
130         return hash_key
131
132     def insert_entry(self, score, move, hash_key, depth, alpha, beta):
133         """
134         Inserts an entry into the transposition table.
135
136         Args:
137             score (int): The evaluation score.
138             move (Move): The best move found.
139             hash_key (int): The Zobrist hash key.
140             depth (int): The depth of the search.
141             alpha (int): The upper bound value.
142             beta (int): The lower bound value.
143
144         Raises:
145             Exception: Invalid depth or score.
146         """

```

```

44         if depth == 0 or alpha < score < beta:
45             flag = TranspositionFlag.EXACT
46             score = score
47         elif score <= alpha:
48             flag = TranspositionFlag.UPPER
49             score = alpha
50         elif score >= beta:
51             flag = TranspositionFlag.LOWER
52             score = beta
53         else:
54             raise Exception('(TranspositionTable.insert_entry)')
55
56         self._table[self.calculate_entry_index(hash_key)] = TranspositionEntry(
57             score, move, flag, hash_key, depth)
58
59         if len(self._table) > self._max_entries:
60             # Removes the longest-existing entry to free up space for more up-to-
61             # date entries
62             # Expression to remove leftmost item taken from https://docs.python.
63             # org/3/library/collections.html#ordereddict-objects
64             (k := next(iter(self._table)), self._table.pop(k))
65
66     def get_entry(self, hash_key, depth, alpha, beta):
67         """
68         Gets an entry from the transposition table.
69
70         Args:
71             hash_key (int): The Zobrist hash key.
72             depth (int): The depth of the search.
73             alpha (int): The alpha value for pruning.
74             beta (int): The beta value for pruning.
75
76         Returns:
77             tuple[int, Move] | tuple[None, None]: The evaluation score and the
78             best move found, if entry exists.
79         """
80         index = self.calculate_entry_index(hash_key)
81
82         if index not in self._table:
83             return None, None
84
85         entry = self._table[index]
86
87         if entry.hash_key == hash_key and entry.depth >= depth:
88             if entry.flag == TranspositionFlag.EXACT:
89                 return entry.score, entry.move
90
91             if entry.flag == TranspositionFlag.LOWER and entry.score >= beta:
92                 return entry.score, entry.move
93
94             if entry.flag == TranspositionFlag.UPPER and entry.score <= alpha:
95                 return entry.score, entry.move
96
97         return None, None
98
99     1 from random import randint
100    2 from data.utils.bitboard_helpers import bitboard_to_index
101    3 from data.constants import Piece, Colour, Rotation
102    4
103    5 # Initialise random values for each piece type on every square
104    6 # (5 x 2 colours) pieces + 4 rotations, for 80 squares
105    7 zobrist_table = [[randint(0, 2 ** 64) for i in range(14)] for j in range(80)]

```

```

8 # Hash for when the red player's move
9 red_move_hash = randint(0, 2 ** 64)
10
11 # Maps piece to the correct random value
12 piece_lookup = {
13     Colour.BLUE: {
14         piece: i for i, piece in enumerate(Piece)
15     },
16     Colour.RED: {
17         piece: i + 5 for i, piece in enumerate(Piece)
18     },
19 }
20
21 # Maps rotation to the correct random value
22 rotation_lookup = {
23     rotation: i + 10 for i, rotation in enumerate(Rotation)
24 }
25
26 class ZobristHasher:
27     def __init__(self):
28         self.hash = 0
29
30     def get_piece_hash(self, index, piece, colour):
31         """
32         Gets the random value for the piece type on the given square.
33
34         Args:
35             index (int): The index of the square.
36             piece (Piece): The piece on the square.
37             colour (Colour): The colour of the piece.
38
39         Returns:
40             int: A 64-bit value.
41         """
42         piece_index = piece_lookup[colour][piece]
43         return zobrist_table[index][piece_index]
44
45     def get_rotation_hash(self, index, rotation):
46         """
47         Gets the random value for theon the given square.
48
49         Args:
50             index (int): The index of the square.
51             rotation (Rotation): The rotation on the square.
52             colour (Colour): The colour of the piece.
53
54         Returns:
55             int: A 64-bit value.
56         """
57         rotation_index = rotation_lookup[rotation]
58         return zobrist_table[index][rotation_index]
59
60     def apply_piece_hash(self, bitboard, piece, colour):
61         """
62         Updates the Zobrist hash with a new piece.
63
64         Args:
65             bitboard (int): The bitboard representation of the square.
66             piece (Piece): The piece on the square.
67             colour (Colour): The colour of the piece.
68         """
69         index = bitboard_to_index(bitboard)

```

```

70     piece_hash = self.get_piece_hash(index, piece, colour)
71     self.hash ^= piece_hash
72
73     def apply_rotation_hash(self, bitboard, rotation):
74         """Updates the Zobrist hash with a new rotation.
75
76         Args:
77             bitboard (int): The bitboard representation of the square.
78             rotation (Rotation): The rotation on the square.
79         """
80         index = bitboard_to_index(bitboard)
81         rotation_hash = self.get_rotation_hash(index, rotation)
82         self.hash ^= rotation_hash
83
84     def apply_red_move_hash(self):
85         """
86         Applies the Zobrist hash for the red player's move.
87         """
88         self.hash ^= red_move_hash
89
90
91 1
92 2 from data.states.game.cpu.move_orderer import MoveOrderer
93 3 from data.states.game.cpu.base import BaseCPU
94 4 from data.constants import Score, Colour
95 5 from random import choice
96 6 from data.utils.bitboard_helpers import print_bitboard
97 7 orderer = MoveOrderer()
98 8
99 9 class ABMinimaxCPU(BaseCPU):
100 10     def __init__(self, max_depth, callback, verbose=True):
101 11         super().__init__(callback, verbose)
102 12         self._max_depth = max_depth
103 13
104 14     def initialise_stats(self):
105 15         """
106 16         Initialises the number of prunes to the statistics dictionary to be logged
107 17         .
108 18         """
109 19         super().initialise_stats()
110 20         self._stats['beta_prunes'] = 0
111 21         self._stats['alpha_prunes'] = 0
112 22
113 23     def find_move(self, board, stop_event):
114 24         """
115 25         Finds the best move for the current board state.
116 26
117 27         Args:
118 28             board (Board): The current board state.
119 29             stop_event (threading.Event): Event used to kill search from an
120 30             external class.
121 31         """
122 32         self.initialise_stats()
123 33         best_score, best_move = self.search(board, self._max_depth, -Score.
124 34         INFINITE, Score.INFINITE, stop_event)
125 35
126 36         if self._verbose:
127 37             self.print_stats(best_score, best_move)
128 38
129 39         self._callback(best_move)
130 40
131 41     def search(self, board, depth, alpha, beta, stop_event, hint=None):
132 42         """

```

```

40     Recursively DFS through minimax tree while pruning branches using the
    alpha and beta bounds.
41
42     Args:
43         board (Board): The current board state.
44         depth (int): The current search depth.
45         alpha (int): The upper bound value.
46         beta (int): The lower bound value.
47         stop_event (threading.Event): Event used to kill search from an
    external class.
48
49     Returns:
50         tuple[int, Move]: The best score and the best move found.
51     """
52     if (base_case := super().search(board, depth, stop_event)):
53         return base_case
54
55     best_move = None
56
57     # Blue is the maximising player
58     if board.get_active_colour() == Colour.BLUE:
59         max_score = -Score.INFINITE
60
61         for move in orderer.get_moves(board, hint):
62             laser_result = board.apply_move(move)
63             new_score = self.search(board, depth - 1, alpha, beta, stop_event,
    laser_result.pieces_on_trajectory)[0]
64
65             if new_score > max_score:
66                 max_score = new_score
67                 best_move = move
68
69             board.undo_move(move, laser_result)
70
71             alpha = max(alpha, max_score)
72
73             if beta <= alpha:
74                 self._stats['alpha_prunes'] += 1
75                 break
76
77         return max_score, best_move
78
79     else:
80         min_score = Score.INFINITE
81
82         for move in orderer.get_moves(board, hint):
83             laser_result = board.apply_move(move)
84             new_score = self.search(board, depth - 1, alpha, beta, stop_event,
    laser_result.pieces_on_trajectory)[0]
85
86             if new_score < min_score:
87                 min_score = new_score
88                 best_move = move
89
90             board.undo_move(move, laser_result)
91
92             beta = min(beta, min_score)
93             if beta <= alpha:
94                 self._stats['beta_prunes'] += 1
95                 break
96
97         return min_score, best_move

```



```

98
99 class ABNegamaxCPU(BaseCPU):
100     def __init__(self, max_depth, callback, verbose=True):
101         super().__init__(callback, verbose)
102         self._max_depth = max_depth
103
104     def initialise_stats(self):
105         """Initialises the statistics for the search."""
106         super().initialise_stats()
107         self._stats['beta_prunes'] = 0
108
109     def find_move(self, board, stop_event):
110         """Finds the best move for the current board state.
111
112         Args:
113             board (Board): The current board state.
114             stop_event (threading.Event): The event to signal stopping the search.
115         """
116         self.initialise_stats()
117         best_score, best_move = self.search(board, self._max_depth, -Score.
INFINITE, Score.INFINITE, stop_event)
118
119         if self._verbose:
120             self.print_stats(best_score, best_move)
121
122         self._callback(best_move)
123
124     def search(self, board, depth, alpha, beta, stop_event):
125         """Searches for the best move using the Alpha-Beta Negamax algorithm.
126
127         Args:
128             board (Board): The current board state.
129             depth (int): The current depth in the game tree.
130             alpha (int): The alpha value for pruning.
131             beta (int): The beta value for pruning.
132             stop_event (threading.Event): The event to signal stopping the search.
133
134         Returns:
135             tuple: The best score and the best move found.
136         """
137         if (base_case := super().search(board, depth, stop_event, absolute=True)):
138             return base_case
139
140         best_move = None
141         best_score = alpha
142
143         for move in board.generate_all_moves(board.get_active_colour()):
144             laser_result = board.apply_move(move)
145
146             new_score = self.search(board, depth - 1, -beta, -best_score,
stop_event)[0]
147             new_score = -new_score
148
149             if new_score > best_score:
150                 best_score = new_score
151                 best_move = move
152             elif new_score == best_score:
153                 best_move = choice([best_move, move])
154
155             board.undo_move(move, laser_result)
156
157             if best_score >= beta:

```

```

158         self._stats['beta_prunes'] += 1
159         break
160
161     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
4
5 class IterativeDeepeningMixin:
6     def find_move(self, board, stop_event):
7         best_move = None
8
9         for depth in range(1, self._max_depth + 1):
10             self.initialise_stats()
11             self._stats['ID_depth'] = depth
12
13             best_score, best_move = self.search(board, depth, -Score.INFINITE,
Score.INFINITE, stop_event)
14
15             if self._verbose:
16                 self.print_stats(best_score, best_move)
17
18             self._callback(best_move)
19
20 class IDMinimaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABMinimaxCPU)
:
21     def initialise_stats(self):
22         super().initialise_stats()
23         self._stats['cache_hits'] = 0
24
25     def print_stats(self, score, move):
26         self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
self._stats['nodes'], 3)
27         self._stats['cache_entries'] = len(self._table._table)
28         super().print_stats(score, move)
29
30 class IDNegamaxCPU(TranspositionTableMixin, IterativeDeepeningMixin, ABNegamaxCPU)
:
31     def initialise_stats(self):
32         super().initialise_stats()
33         self._stats['cache_hits'] = 0
34
35     def print_stats(self, score, move):
36         self._stats['cache_hits_percentage'] = self._stats['cache_hits'] / self.
_stats['nodes']
37         self._stats['cache_entries'] = len(self._table._table)
38         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
4
5 class MinimaxCPU(BaseCPU):
6     def __init__(self, max_depth, callback, verbose=False):
7         super().__init__(callback, verbose)
8         self._max_depth = max_depth
9
10    def find_move(self, board, stop_event):
11        """
12        Finds the best move for the current board state.

```

```

13
14     Args:
15         board (Board): The current board state.
16         stop_event (threading.Event): Event used to kill search from an
external class.
17     """
18     self.initialise_stats()
19     best_score, best_move = self.search(board, self._max_depth, stop_event)
20
21     if self._verbose:
22         self.print_stats(best_score, best_move)
23
24     self._callback(best_move)
25
26 def search(self, board, depth, stop_event):
27     """
28     Recursively DFS through minimax tree with evaluation score.
29
30     Args:
31         board (Board): The current board state.
32         depth (int): The current search depth.
33         stop_event (threading.Event): Event used to kill search from an
external class.
34     Returns:
35         tuple[int, Move]: The best score and the best move found.
36     """
37     if (base_case := super().search(board, depth, stop_event)):
38         return base_case
39
40     best_move = None
41
42     # Blue is the maximising player
43     if board.get_active_colour() == Colour.BLUE:
44         max_score = -Score.INFINITE
45
46         for move in board.generate_all_moves(Colour.BLUE):
47             laser_result = board.apply_move(move)
48
49             new_score = self.search(board, depth - 1, stop_event)[0]
50
51             # if depth < self._max_depth:
52             #     print('DEPTH', depth, new_score, move)
53
54             if new_score > max_score:
55                 max_score = new_score
56                 best_move = move
57
58                 if new_score == (Score.CHECKMATE + self._max_depth):
59                     board.undo_move(move, laser_result)
60                     return max_score, best_move
61
62             elif new_score == max_score:
63                 # If evaluated scores are equal, pick a random move
64                 best_move = choice([best_move, move])
65
66             board.undo_move(move, laser_result)
67
68         return max_score, best_move
69
70 else:
71     min_score = Score.INFINITE
72

```

```

73
74         for move in board.generate_all_moves(Colour.RED):
75             laser_result = board.apply_move(move)
76             # print('DEPTH', depth, move)
77             new_score = self.search(board, depth - 1, stop_event)[0]
78
79             if new_score < min_score:
80                 # print('setting new', new_score, move)
81                 min_score = new_score
82                 best_move = move
83
84             if new_score == (-Score.CHECKMATE - self._max_depth):
85                 board.undo_move(move, laser_result)
86                 return min_score, best_move
87
88             elif new_score == min_score:
89                 best_move = choice([best_move, move])
90
91             board.undo_move(move, laser_result)
92
93         return min_score, best_move
94
95 from data.constants import Score, Colour, Miscellaneous, MoveType
96 from data.states.game.cpu.base import BaseCPU
97 from data.utils.bitboard_helpers import print_bitboard, is_occupied
98 from random import choice, randint
99 from copy import deepcopy
100
101 class NegamaxCPU(BaseCPU):
102     def __init__(self, max_depth, callback, verbose=False):
103         super().__init__(callback, verbose)
104         self._max_depth = max_depth
105
106     def find_move(self, board, stop_event):
107         self.initialise_stats()
108         best_score, best_move = self.search(board, self._max_depth, stop_event)
109
110         if self._verbose:
111             self.print_stats(best_score, best_move)
112
113         self._callback(best_move)
114
115     def search(self, board, depth, stop_event, moves=None):
116         if (base_case := super().search(board, depth, stop_event, absolute=True)):
117             return base_case
118
119         best_move = None
120         best_score = -Score.INFINITE
121
122         for move in board.generate_all_moves(board.get_active_colour()):
123             laser_result = board.apply_move(move)
124
125             new_score = self.search(board, depth - 1, stop_event)[0]
126             new_score = -new_score
127
128             if new_score > best_score:
129                 best_score = new_score
130                 best_move = move
131             elif new_score == best_score:
132                 best_move = choice([best_move, move])
133
134             board.undo_move(move, laser_result)

```

```

41
42         return best_score, best_move

1 from data.states.game.cpu.base import BaseCPU
2 from data.constants import Colour, Score
3
4 class SimpleCPU(BaseCPU):
5     def __init__(self, callback, verbose=True):
6         super().__init__(callback, verbose)
7
8     def find_move(self, board, stop_event=None):
9         self.initialise_stats()
10        best_score, best_move = self.search(board, stop_event)
11
12        if self._verbose:
13            self.print_stats(best_score, best_move)
14
15        self._callback(best_move)
16
17    def search(self, board, stop_event):
18        if stop_event and stop_event.is_set():
19            raise Exception('Thread killed - stopping simple function (SimpleCPU.
search)')
20
21        active_colour = board.bitboards.active_colour
22        best_score = -Score.INFINITE if active_colour == Colour.BLUE else Score.
INFINITE
23        best_move = None
24
25        for move in board.generate_all_moves(active_colour):
26            laser_result = board.apply_move(move)
27
28            self._stats['nodes'] += 1
29
30            if winner := board.check_win() is not None:
31                self.process_win(winner)
32            else:
33                self._stats['leaf_nodes'] += 1
34
35            score = self._evaluator.evaluate(board)
36
37            if (active_colour == Colour.BLUE and score > best_score) or (
active_colour == Colour.RED and score < best_score):
38                best_move = move
39                best_score = score
40
41            board.undo_move(move, laser_result)
42
43        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
3
4 class TranspositionTableMixin:
5     def __init__(self, *args, **kwargs):
6         super().__init__(*args, **kwargs)
7         self._table = TranspositionTable()
8
9     def find_move(self, *args, **kwargs):
10        self._table = TranspositionTable()
11        super().find_move(*args, **kwargs)
12

```

```

13     def search(self, board, depth, alpha, beta, stop_event, hint=None):
14         """
15         Searches transposition table for a cached move before running a full
16         search if necessary.
17         Caches the searched result.
18
19         Args:
20             board (Board): The current board state.
21             depth (int): The current search depth.
22             alpha (int): The upper bound value.
23             beta (int): The lower bound value.
24             stop_event (threading.Event): Event used to kill search from an
25             external class.
26
27         Returns:
28             tuple[int, Move]: The best score and the best move found.
29         """
30         hash = board.to_hash()
31         score, move = self._table.get_entry(hash, depth, alpha, beta)
32
33         if score is not None:
34             self._stats['cache_hits'] += 1
35             self._stats['nodes'] += 1
36
37             return score, move
38         else:
39             # If board hash entry not found in cache, run a full search
40             score, move = super().search(board, depth, alpha, beta, stop_event,
41             hint)
42             self._table.insert_entry(score, move, hash, depth, alpha, beta)
43
44             return score, move
45
46     class TTMinimaxCPU(TranspositionTableMixin, ABMinimaxCPU):
47         def initialise_stats(self):
48             """
49             Initialises cache statistics to be logged.
50             """
51             super().initialise_stats()
52             self._stats['cache_hits'] = 0
53
54         def print_stats(self, score, move):
55             """
56             Logs the statistics for the search.
57
58             Args:
59                 score (int): The best score found.
60                 move (Move): The best move found.
61             """
62             # Calculate number of cached entries retrieved as a percentage of all
63             nodes
64             self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
65             self._stats['nodes'], 3)
66             self._stats['cache_entries'] = len(self._table._table)
67             super().print_stats(score, move)
68
69     class TTNegamaxCPU(TranspositionTableMixin, ABNegamaxCPU):
70         """Negamax CPU engine with transposition table support."""
71
72         def initialise_stats(self):
73             """Initialises the statistics for the search."""
74             super().initialise_stats()

```

```

70         self._stats['cache_hits'] = 0
71
72     def print_stats(self, score, move):
73         """Prints the statistics for the search.
74
75         Args:
76             score (int): The best score found.
77             move (Move): The best move found.
78         """
79         self._stats['cache_hits_percentage'] = round(self._stats['cache_hits'] /
80 self._stats['nodes'], 3)
81         self._stats['cache_entries'] = len(self._table._table)
82         super().print_stats(score, move)
83
84
85 1 from data.states.game.cpu.engines.simple import SimpleCPU
86 2 from data.states.game.cpu.engines.negamax import NegamaxCPU
87 3 from data.states.game.cpu.engines.minimax import MinimaxCPU
88 4 from data.states.game.cpu.engines.alpha_beta import ABMinimaxCPU, ABNegamaxCPU
89 5 from data.states.game.cpu.engines.iterative_deepening import IDMinimaxCPU,
90   IDNegamaxCPU
91 6 from data.states.game.cpu.engines.transposition_table import TTMinimaxCPU,
92   TTNegamaxCPU
93
94 1 import pygame
95 2 from data.constants import GameEventType, MoveType, StatusText, Miscellaneous
96 3 from data.utils import bitboard_helpers as bb_helpers
97 4 from data.states.game.components.move import Move
98 5 from data.managers.logs import initialise_logger
99
100 6
101 7 logger = initialise_logger(__name__)
102 8
103 9 class GameController:
104 10     def __init__(self, model, view, win_view, pause_view, to_menu, to_review,
105   to_new_game):
106 11         self._model = model
107 12         self._view = view
108 13         self._win_view = win_view
109 14         self._pause_view = pause_view
110 15
111 16         self._to_menu = to_menu
112 17         self._to_review = to_review
113 18         self._to_new_game = to_new_game
114 19
115 20         self._view.initialise_timers()
116 21         self._win_view.set_win_type('CAPTURE')
117 22
118 23     def cleanup(self, next):
119 24         """
120 25         Handles game quit, either leaving to main menu or restarting a new game.
121 26
122 27         Args:
123 28             next (str): New state to switch to.
124 29         """
125 30         self._model.kill_thread()
126 31
127 32         if next == 'menu':
128 33             self._to_menu()
129 34         elif next == 'game':
130 35             self._to_new_game()
131 36         elif next == 'review':
132 37             self._to_review()
133 38

```

```

39     def make_move(self, move):
40         """
41         Handles player move.
42
43         Args:
44             move (Move): Move to make.
45         """
46         self._model.make_move(move)
47         self._view.set_overlay_coords([], None)
48
49         if self._model.states['CPU_ENABLED']:
50             self._model.make_cpu_move()
51
52     def handle_pause_event(self, event):
53         """
54         Processes events when game is paused.
55
56         Args:
57             event (GameEventType): Event to process.
58
59         Raises:
60             Exception: If event type is unrecognised.
61         """
62         game_event = self._pause_view.convert_mouse_pos(event)
63
64         if game_event is None:
65             return
66
67         match game_event.type:
68             case GameEventType.PAUSE_CLICK:
69                 self._model.toggle_paused()
70
71             case GameEventType.MENU_CLICK:
72                 self.cleanup('menu')
73
74             case _:
75                 raise Exception('Unhandled event type (GameController.handle_event
)')
76
77     def handle_winner_event(self, event):
78         """
79         Processes events when game is over.
80
81         Args:
82             event (GameEventType): Event to process.
83
84         Raises:
85             Exception: If event type is unrecognised.
86         """
87         game_event = self._win_view.convert_mouse_pos(event)
88
89         if game_event is None:
90             return
91
92         match game_event.type:
93             case GameEventType.MENU_CLICK:
94                 self.cleanup('menu')
95                 return
96
97             case GameEventType.GAME_CLICK:
98                 self.cleanup('game')
99                 return

```



```

100
101         case GameEventType.REVIEW_CLICK:
102             self.cleanup('review')
103
104         case _:
105             raise Exception('Unhandled event type (GameController.handle_event
106 )')
107
108 def handle_game_widget_event(self, event):
109     """
110     Processes events for game GUI widgets.
111
112     Args:
113         event (GameEventType): Event to process.
114
115     Raises:
116         Exception: If event type is unrecognised.
117
118     Returns:
119         CustomEvent | None: A widget event.
120     """
121     widget_event = self._view.process_widget_event(event)
122
123     if widget_event is None:
124         return None
125
126     match widget_event.type:
127         case GameEventType.ROTATE_PIECE:
128             src_coords = self._view.get_selected_coords()
129
130             if src_coords is None:
131                 logger.info('None square selected')
132                 return
133
134             move = Move.instance_from_coords(MoveType.ROTATE, src_coords,
135 src_coords, rotation_direction=widget_event.rotation_direction)
136             self.make_move(move)
137
138         case GameEventType.RESIGN_CLICK:
139             self._model.set_winner(self._model.states['ACTIVE_COLOUR'].
140 get_flipped_colour())
141             self._view.handle_game_end(play_sfx=False)
142             self._win_view.set_win_type('RESIGN')
143
144         case GameEventType.DRAW_CLICK:
145             self._model.set_winner(Miscellaneous.DRAW)
146             self._view.handle_game_end(play_sfx=False)
147             self._win_view.set_win_type('DRAW')
148
149         case GameEventType.TIMER_END:
150             if self._model.states['TIME_ENABLED']:
151                 self._model.set_winner(widget_event.active_colour.
152 get_flipped_colour())
153                 self._win_view.set_win_type('TIME')
154                 self._view.handle_game_end(play_sfx=False)
155
156         case GameEventType.MENU_CLICK:
157             self.cleanup('menu')
158
159         case GameEventType.HELP_CLICK:
160             self._view.add_help_screen()

```

```

158         case GameEventType.TUTORIAL_CLICK:
159             self._view.add_tutorial_screen()
160
161         case _:
162             raise Exception('Unhandled event type (GameController.handle_event
)')
163
164         return widget_event.type
165
166     def check_cpu(self):
167         """
168         Checks if CPU calculations are finished every frame.
169         """
170         if self._model.states['CPU_ENABLED'] and self._model.states['AWAITING_CPU'
] is False:
171             self._model.check_cpu()
172
173     def handle_game_event(self, event):
174         """
175         Processes Pygame events for main game.
176
177         Args:
178             event (pygame.Event): If event type is unrecognised.
179
180         Raises:
181             Exception: If event type is unrecognised.
182         """
183         # Pass event for widgets to process
184         widget_event = self.handle_game_widget_event(event)
185
186         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
KEYDOWN]:
187             if event.type != pygame.KEYDOWN:
188                 game_event = self._view.convert_mouse_pos(event)
189             else:
190                 game_event = None
191
192             if game_event is None:
193                 if widget_event is None:
194                     if event.type in [pygame.MOUSEBUTTONUP, pygame.KEYDOWN]:
195                         # If user releases mouse click not on a widget
196                         self._view.remove_help_screen()
197                         self._view.remove_tutorial_screen()
198                     if event.type == pygame.MOUSEBUTTONUP:
199                         # If user releases mouse click on neither a widget or
board
200
201                         self._view.set_overlay_coords(None, None)
202
203                 return
204
205             match game_event.type:
206                 case GameEventType.BOARD_CLICK:
207                     if self._model.states['AWAITING_CPU']:
208                         return
209
210                     clicked_coords = game_event.coords
211                     clicked_bitboard = bb_helpers.coords_to_bitboard(
clicked_coords)
212
213                     selected_coords = self._view.get_selected_coords()
214
215                     if selected_coords:
216                         if clicked_coords == selected_coords:

```

```

215         # If clicking on an already selected square, start
dragging piece on that square
216         self._view.set_dragged_piece(*self._model.
get_piece_info(clicked_bitboard))
217         return
218
219         selected_bitboard = bb_helpers.coords_to_bitboard(
selected_coords)
220         available_bitboard = self._model.get_available_moves(
selected_bitboard)
221
222         if bb_helpers.is_occupied(clicked_bitboard,
available_bitboard):
223             # If the newly clicked square is not the same as the
old one, and is an empty surrounding square, make a move
224             move = Move.instance_from_coords(MoveType.MOVE,
selected_coords, clicked_coords)
225             self.make_move(move)
226         else:
227             # If the newly clicked square is not the same as the
old one, but is an invalid square, unselect the currently selected square
228             self._view.set_overlay_coords(None, None)
229
230             # Select hovered square if it is same as active colour
231             elif self._model.is_selectable(clicked_bitboard):
232                 available_bitboard = self._model.get_available_moves(
clicked_bitboard)
233                 self._view.set_overlay_coords(bb_helpers.
bitboard_to_coords_list(available_bitboard), clicked_coords)
234                 self._view.set_dragged_piece(*self._model.get_piece_info(
clicked_bitboard))
235
236         case GameEventType.PIECE_DROP:
237             hovered_coords = game_event.coords
238
239             # if piece is dropped onto the board
240             if hovered_coords:
241                 hovered_bitboard = bb_helpers.coords_to_bitboard(
hovered_coords)
242                 selected_coords = self._view.get_selected_coords()
243                 selected_bitboard = bb_helpers.coords_to_bitboard(
selected_coords)
244                 available_bitboard = self._model.get_available_moves(
selected_bitboard)
245
246                 if bb_helpers.is_occupied(hovered_bitboard,
available_bitboard):
247                     # Make a move if mouse is hovered over an empty
surrounding square
248                     move = Move.instance_from_coords(MoveType.MOVE,
selected_coords, hovered_coords)
249                     self.make_move(move)
250
251                     if game_event.remove_overlay:
252                         self._view.set_overlay_coords(None, None)
253
254                     self._view.remove_dragged_piece()
255
256         case _:
257             raise Exception('Unhandled event type (GameController.
handle_event)', game_event.type)
258

```

```

259     def handle_event(self, event):
260         """
261         Passe a Pygame event to the correct handling function according to the
262         game state.
263
264         Args:
265             event (pygame.Event): Event to process.
266         """
267         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
268             MOUSEMOTION, pygame.KEYDOWN]:
269             if self._model.states['PAUSED']:
270                 self.handle_pause_event(event)
271             elif self._model.states['WINNER'] is not None:
272                 self.handle_winner_event(event)
273             else:
274                 self.handle_game_event(event)
275
276         if event.type == pygame.KEYDOWN:
277             if event.key == pygame.K_ESCAPE:
278                 self._model.toggle_paused()
279             elif event.key == pygame.K_l:
280                 logger.info('\nSTOPPING CPU')
281                 self._model._cpu_thread.stop_cpu() #temp
282
283
284 1 from data.states.game.components.fen_parser import encode_fen_string
285 2 from data.constants import Colour, GameEventType, EMPTY_BB
286 3 from data.states.game.widget_dict import GAME_WIDGETS
287 4 from data.states.game.cpu.cpu_thread import CPUThread
288 5 from data.states.game.cpu.engines import ABMinimaxCPU
289 6 from data.components.custom_event import CustomEvent
290 7 from data.utils.bitboard_helpers import is_occupied
291 8 from data.states.game.components.board import Board
292 9 from data.utils import input_helpers as ip_helpers
293 10 from data.states.game.components.move import Move
294 11 from data.managers.logs import initialise_logger
295 12
296 13 logger = initialise_logger(__name__)
297 14
298 15 class GameModel:
299     def __init__(self, game_config):
300         self._listeners = {
301             'game': [],
302             'win': [],
303             'pause': [],
304         }
305         self._board = Board(fen_string=game_config['FEN_STRING'])
306
307         self.states = {
308             'CPU_ENABLED': game_config['CPU_ENABLED'],
309             'CPU_DEPTH': game_config['CPU_DEPTH'],
310             'AWAITING_CPU': False,
311             'WINNER': None,
312             'PAUSED': False,
313             'ACTIVE_COLOUR': game_config['COLOUR'],
314             'TIME_ENABLED': game_config['TIME_ENABLED'],
315             'TIME': game_config['TIME'],
316             'START_FEN_STRING': game_config['FEN_STRING'],
317             'MOVES': [],
318             'ZOBORIST_KEYS': []
319         }
320
321         self._cpu = ABMinimaxCPU(self.states['CPU_DEPTH'], self.cpu_callback,

```

```

verbose=False)
39     self._cpu_thread = CPUThread(self._cpu)
40     self._cpu_thread.start()
41     self._cpu_move = None
42
43     logger.info(f'Initialising CPU depth of {self.states['CPU_DEPTH']}')
44
45     def register_listener(self, listener, parent_class):
46         """
47         Registers listener method of another MVC class.
48
49         Args:
50             listener (callable): Listener callback function.
51             parent_class (str): Class name.
52         """
53         self._listeners[parent_class].append(listener)
54
55     def alert_listeners(self, event):
56         """
57         Alerts all registered classes of an event by calling their listener
58         function.
59
60         Args:
61             event (GameEventType): Event to pass as argument.
62
63         Raises:
64             Exception: If an unrecognised event tries to be passed onto listeners.
65         """
66         for parent_class, listeners in self._listeners.items():
67             match event.type:
68                 case GameEventType.UPDATE_PIECES:
69                     if parent_class in 'game':
70                         for listener in listeners: listener(event)
71
72                 case GameEventType.SET_LASER:
73                     if parent_class == 'game':
74                         for listener in listeners: listener(event)
75
76                 case GameEventType.PAUSE_CLICK:
77                     if parent_class in ['pause', 'game']:
78                         for listener in listeners:
79                             listener(event)
80
81                 case _:
82                     raise Exception('Unhandled event type (GameModel.
83                     alert_listeners)')
84
85     def set_winner(self, colour=None):
86         """
87         Sets winner.
88
89         Args:
90             colour (Colour, optional): Describes winnner colour, or draw. Defaults
91             to None.
92         """
93         self.states['WINNER'] = colour
94
95     def toggle_paused(self):
96         """
97         Toggles pause screen, and alerts pause view.
98         """
99         self.states['PAUSED'] = not self.states['PAUSED']

```

```

97         game_event = CustomEvent.create_event(GameEventType.PAUSE_CLICK)
98         self.alert_listeners(game_event)
99
100     def get_terminal_move(self):
101         """
102         Debugging method for inputting a move from the terminal.
103
104         Returns:
105             Move: Parsed move.
106         """
107         while True:
108             try:
109                 move_type = ip_helpers.parse_move_type(input('Input move type (m/r
110 ): '))
111                 src_square = ip_helpers.parse_notation(input("From: "))
112                 dest_square = ip_helpers.parse_notation(input("To: "))
113                 rotation = ip_helpers.parse_rotation(input("Enter rotation (a/b/c/
114 d): "))
115                 return Move.instance_from_notation(move_type, src_square,
116 dest_square, rotation)
117             except ValueError as error:
118                 logger.warning('Input error (Board.get_move): ' + str(error))
119
120     def make_move(self, move):
121         """
122         Takes a Move object and applies it to the board.
123
124         Args:
125             move (Move): Move to apply.
126         """
127         colour = self._board.bitboards.get_colour_on(move.src)
128         piece = self._board.bitboards.get_piece_on(move.src, colour)
129         # Apply move and get results of laser trajectory
130         laser_result = self._board.apply_move(move, add_hash=True)
131
132         self.alert_listeners(CustomEvent.create_event(GameEventType.SET_LASER,
133 laser_result=laser_result))
134
135         # Sets new active colour and checks for a win
136         self.states['ACTIVE_COLOUR'] = self._board.get_active_colour()
137         self.set_winner(self._board.check_win())
138
139         move_notation = move.to_notation(colour, piece, laser_result,
140 hit_square_bitboard)
141
142         self.alert_listeners(CustomEvent.create_event(GameEventType.UPDATE_PIECES,
143 move_notation=move_notation))
144
145         # Adds move to move history list for review screen
146         self.states['MOVES'].append({
147             'time': {
148                 Colour.BLUE: GAME_WIDGETS['blue_timer'].get_time(),
149                 Colour.RED: GAME_WIDGETS['red_timer'].get_time()
150             },
151             'move': move_notation,
152             'laserResult': laser_result
153         })
154
155     def make_cpu_move(self):
156         """
157         Starts CPU calculations on the separate thread.
158         """

```

```

153         self.states['AWAITING_CPU'] = True
154         self._cpu_thread.start_cpu(self.get_board())
155
156     def cpu_callback(self, move):
157         """
158         Callback function passed to CPU thread. Called when CPU stops processing.
159
160         Args:
161             move (Move): Move that CPU found.
162         """
163         if self.states['WINNER'] is None:
164             # CPU move passed back to main thread by reassigning variable
165             self._cpu_move = move
166             self.states['AWAITING_CPU'] = False
167
168     def check_cpu(self):
169         """
170         Constantly checks if CPU calculations are finished, so that make_move can
171         be run on the main thread.
172         """
173         if self._cpu_move is not None:
174             self.make_move(self._cpu_move)
175             self._cpu_move = None
176
177     def kill_thread(self):
178         """
179         Interrupt and kill CPU thread.
180         """
181         self._cpu_thread.kill_thread()
182         self.states['AWAITING_CPU'] = False
183
184     def is_selectable(self, bitboard):
185         """
186         Checks if square is occupied by a piece of the current active colour.
187
188         Args:
189             bitboard (int): Bitboard representing single square.
190
191         Returns:
192             bool: True if square is occupied by a piece of the current active
193             colour. False if not.
194         """
195         return is_occupied(self._board.bitboards.combined_colour_bitboards[self.
196             states['ACTIVE_COLOUR']], bitboard)
197
198     def get_available_moves(self, bitboard):
199         """
200         Gets all surrounding empty squares. Used for drawing overlay.
201
202         Args:
203             bitboard (int): Bitboard representing single center square.
204
205         Returns:
206             int: Bitboard representing all empty surrounding squares.
207         """
208         if (bitboard & self._board.get_all_active_pieces()) != EMPTY_BB:
209             return self._board.get_valid_squares(bitboard)
210
211         return EMPTY_BB
212
213     def get_piece_list(self):
214         """

```

```

212         Returns:
213             list[Piece, ...]: Array of all pieces on the board.
214         """
215         return self._board.get_piece_list()
216
217     def get_piece_info(self, bitboard):
218         """
219         Args:
220             bitboard (int): Square containing piece.
221
222         Returns:
223             tuple[Colour, Rotation, Piece]: Piece information.
224         """
225         colour = self._board.bitboards.get_colour_on(bitboard)
226         rotation = self._board.bitboards.get_rotation_on(bitboard)
227         piece = self._board.bitboards.get_piece_on(bitboard, colour)
228         return (piece, colour, rotation)
229
230     def get_fen_string(self):
231         return encode_fen_string(self._board.bitboards)
232
233     def get_board(self):
234         return self._board

```

```

1  import pygame
2  from data.constants import GameEventType, Colour, StatusText, Miscellaneous,
   ShaderType
3  from data.states.game.components.overlay_draw import 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:
18     def __init__(self, model):
19         self._model = model
20         self._hide_pieces = False
21         self._selected_coords = None
22         self._event_to_func_map = {
23             GameEventType.UPDATE_PIECES: self.handle_update_pieces,
24             GameEventType.SET_LASER: self.handle_set_laser,
25             GameEventType.PAUSE_CLICK: self.handle_pause,
26         }
27
28         # Register model event handling with process_model_event()
29         self._model.register_listener(self.process_model_event, 'game')
30
31         # Initialise WidgetGroup with map of widgets
32         self._widget_group = WidgetGroup(GAME_WIDGETS)
33         self._widget_group.handle_resize(window.size)
34         self.initialise_widgets()
35
36         self._laser_draw = LaserDraw(self.board_position, self.board_size)
37         self._overlay_draw = OverlayDraw(self.board_position, self.board_size)

```



```

38         self._drag_and_drop = DragAndDrop(self.board_position, self.board_size)
39         self._capture_draw = CaptureDraw(self.board_position, self.board_size)
40         self._piece_group = PieceGroup()
41         self.handle_update_pieces()
42
43         self.set_status_text(StatusText.PLAYER_MOVE)
44
45     @property
46     def board_position(self):
47         return GAME_WIDGETS['chessboard'].position
48
49     @property
50     def board_size(self):
51         return GAME_WIDGETS['chessboard'].size
52
53     @property
54     def square_size(self):
55         return self.board_size[0] / 10
56
57     def initialise_widgets(self):
58         """
59         Run methods on widgets stored in GAME_WIDGETS dictionary to reset them.
60         """
61         GAME_WIDGETS['move_list'].reset_move_list()
62         GAME_WIDGETS['move_list'].kill()
63         GAME_WIDGETS['help'].kill()
64         GAME_WIDGETS['tutorial'].kill()
65
66         GAME_WIDGETS['scroll_area'].set_image()
67
68         GAME_WIDGETS['chessboard'].refresh_board()
69
70         GAME_WIDGETS['blue_piece_display'].reset_piece_list()
71         GAME_WIDGETS['red_piece_display'].reset_piece_list()
72
73     def set_status_text(self, status):
74         """
75         Sets text on status text widget.
76
77         Args:
78             status (StatusText): The game stage for which text should be displayed
79         """
80         match status:
81             case StatusText.PLAYER_MOVE:
82                 GAME_WIDGETS['status_text'].set_text(f"{self._model.states['ACTIVE_COLOUR'].name}'s turn to move")
83             case StatusText.CPU_MOVE:
84                 GAME_WIDGETS['status_text'].set_text(f"CPU calculating a crazy move...")
85             case StatusText.WIN:
86                 if self._model.states['WINNER'] == Miscellaneous.DRAW:
87                     GAME_WIDGETS['status_text'].set_text(f"Game is a draw! Boring...")
88                 else:
89                     GAME_WIDGETS['status_text'].set_text(f"{self._model.states['WINNER'].name} won!")
90             case StatusText.DRAW:
91                 GAME_WIDGETS['status_text'].set_text(f"Game is a draw! Boring...")
92
93     def handle_resize(self):
94         """

```

```

95     Handles resizing of the window.
96     """
97     self._overlay_draw.handle_resize(self.board_position, self.board_size)
98     self._capture_draw.handle_resize(self.board_position, self.board_size)
99     self._piece_group.handle_resize(self.board_position, self.board_size)
100     self._laser_draw.handle_resize(self.board_position, self.board_size)
101     self._laser_draw.handle_resize(self.board_position, self.board_size)
102     self._widget_group.handle_resize(window.size)
103
104     if self._laser_draw.firing:
105         self.update_laser_mask()
106
107 def handle_update_pieces(self, event=None):
108     """
109     Callback function to update pieces after move.
110
111     Args:
112         event (GameEventType, optional): If updating pieces after player move,
113         event contains move information. Defaults to None.
114         toggle_timers (bool, optional): Toggle timers on and off for new
115         active colour. Defaults to True.
116     """
117     piece_list = self._model.get_piece_list()
118     self._piece_group.initialise_pieces(piece_list, self.board_position, self.
board_size)
119
120     if event:
121         GAME_WIDGETS['move_list'].append_to_move_list(event.move_notation)
122         GAME_WIDGETS['scroll_area'].set_image()
123         audio.play_sfx(SFX['piece_move'])
124
125     if self._model.states['ACTIVE_COLOUR'] == Colour.BLUE:
126         self.set_status_text(StatusText.PLAYER_MOVE)
127     elif self._model.states['CPU_ENABLED'] is False:
128         self.set_status_text(StatusText.PLAYER_MOVE)
129     else:
130         self.set_status_text(StatusText.CPU_MOVE)
131
132     if self._model.states['TIME_ENABLED']:
133         self.toggle_timer(self._model.states['ACTIVE_COLOUR'], True)
134         self.toggle_timer(self._model.states['ACTIVE_COLOUR'],
get_flipped_colour(), False)
135
136     if self._model.states['WINNER'] is not None:
137         self.handle_game_end()
138
139 def handle_game_end(self, play_sfx=True):
140     self.toggle_timer(self._model.states['ACTIVE_COLOUR'], False)
141     self.toggle_timer(self._model.states['ACTIVE_COLOUR'].get_flipped_colour()
, False)
142
143     if self._model.states['WINNER'] == Miscellaneous.DRAW:
144         self.set_status_text(StatusText.DRAW)
145     else:
146         self.set_status_text(StatusText.WIN)
147
148     if play_sfx:
149         audio.play_sfx(SFX['sphinx_destroy_1'])
150         audio.play_sfx(SFX['sphinx_destroy_2'])
151         audio.play_sfx(SFX['sphinx_destroy_3'])
152
153 def handle_set_laser(self, event):

```

```

152         """
153         Callback function to draw laser after move.
154
155         Args:
156             event (GameEventType): Contains laser trajectory information.
157         """
158         laser_result = event.laser_result
159
160         # If laser has hit a piece
161         if laser_result.hit_square_bitboard:
162             coords_to_remove = bitboard_to_coords(laser_result.hit_square_bitboard
163 )
164             self._piece_group.remove_piece(coords_to_remove)
165
166             if laser_result.piece_colour == Colour.BLUE:
167                 GAME_WIDGETS['red_piece_display'].add_piece(laser_result.piece_hit
168 )
169                 elif laser_result.piece_colour == Colour.RED:
170                     GAME_WIDGETS['blue_piece_display'].add_piece(laser_result.
171 piece_hit)
172
173             # Draw piece capture GFX
174             self._capture_draw.add_capture(
175                 laser_result.piece_hit,
176                 laser_result.piece_colour,
177                 laser_result.piece_rotation,
178                 coords_to_remove,
179                 laser_result.laser_path[0][0],
180                 self._model.states['ACTIVE_COLOUR']
181             )
182
183             self._laser_draw.add_laser(laser_result, self._model.states['ACTIVE_COLOUR
184 '])
185             self.update_laser_mask()
186
187     def handle_pause(self, event=None):
188         """
189         Callback function for pausing timer.
190
191         Args:
192             event (None): Event argument not used.
193         """
194         is_active = not(self._model.states['PAUSED'])
195         self.toggle_timer(self._model.states['ACTIVE_COLOUR'], is_active)
196
197     def initialise_timers(self):
198         """
199         Initialises both timers with the correct amount of time and starts the
200 timer for the active colour.
201         """
202         if self._model.states['TIME_ENABLED']:
203             GAME_WIDGETS['blue_timer'].set_time(self._model.states['TIME'] * 60 *
204 1000)
205             GAME_WIDGETS['red_timer'].set_time(self._model.states['TIME'] * 60 *
206 1000)
207         else:
208             GAME_WIDGETS['blue_timer'].kill()
209             GAME_WIDGETS['red_timer'].kill()
210
211         self.toggle_timer(self._model.states['ACTIVE_COLOUR'], True)
212
213     def toggle_timer(self, colour, is_active):

```

```

207     """
208     Stops or resumes timer.
209
210     Args:
211         colour (Colour): Timer to toggle.
212         is_active (bool): Whether to pause or resume timer.
213     """
214     if colour == Colour.BLUE:
215         GAME_WIDGETS['blue_timer'].set_active(is_active)
216     elif colour == Colour.RED:
217         GAME_WIDGETS['red_timer'].set_active(is_active)
218
219     def update_laser_mask(self):
220         """
221         Uses pygame.mask to create a mask for the pieces.
222         Used for occluding the ray shader.
223         """
224         temp_surface = pygame.Surface(window.size, pygame.SRCALPHA)
225         self._piece_group.draw(temp_surface)
226         mask = pygame.mask.from_surface(temp_surface, threshold=127)
227         mask_surface = mask.to_surface(unsetcolor=(0, 0, 0, 255), setcolor=(255,
228 0, 0, 255))
229
230         window.set_apply_arguments(ShaderType.RAYS, occlusion=mask_surface)
231
232     def draw(self):
233         """
234         Draws GUI and pieces onto the screen.
235         """
236         self._widget_group.update()
237         self._capture_draw.update()
238
239         self._widget_group.draw()
240         self._overlay_draw.draw(window.screen)
241
242         if self._hide_pieces is False:
243             self._piece_group.draw(window.screen)
244
245         self._laser_draw.draw(window.screen)
246         self._drag_and_drop.draw(window.screen)
247         self._capture_draw.draw(window.screen)
248
249     def process_model_event(self, event):
250         """
251         Registered listener function for handling GameModel events.
252         Each event is mapped to a callback function, and the appropriate one is run
253         .
254
255         Args:
256             event (GameEventType): Game event to process.
257
258         Raises:
259             KeyError: If an unrecognised event type is passed as the argument.
260         """
261         try:
262             self._event_to_func_map.get(event.type)(event)
263         except:
264             raise KeyError('Event type not recognized in Game View (GameView.
265 process_model_event):', event.type)
266
267     def set_overlay_coords(self, available_coords_list, selected_coord):
268         """

```

```

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

```

```

322         """
323         Draw help overlay when player clicks on the help button.
324         """
325         self._widget_group.add(GAME_WIDGETS['help'])
326         self._widget_group.handle_resize(window.size)
327
328     def add_tutorial_screen(self):
329         """
330         Draw tutorial overlay when player clicks on the tutorial button.
331         """
332         self._widget_group.add(GAME_WIDGETS['tutorial'])
333         self._widget_group.handle_resize(window.size)
334         self._hide_pieces = True
335
336     def remove_help_screen(self):
337         GAME_WIDGETS['help'].kill()
338
339     def remove_tutorial_screen(self):
340         GAME_WIDGETS['tutorial'].kill()
341         self._hide_pieces = False
342
343     def process_widget_event(self, event):
344         """
345         Passes Pygame event to WidgetGroup to allow individual widgets to process
346         events.
347
348         Args:
349             event (pygame.Event): Event to process.
350
351         Returns:
352             CustomEvent | None: A widget event.
353         """
354         return self._widget_group.process_event(event)
355
356
357 1 import pygame
358 2 from data.states.game.widget_dict import PAUSE_WIDGETS
359 3 from data.constants import GameEventType, PAUSE_COLOUR
360 4 from data.components.widget_group import WidgetGroup
361 5 from data.managers.window import window
362 6 from data.managers.audio import audio
363 7
364 8 class PauseView:
365 9     def __init__(self, model):
366 10         self._model = model
367 11
368 12         self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
369 13         self._screen_overlay.fill(PAUSE_COLOUR)
370 14
371 15         self._widget_group = WidgetGroup(PAUSE_WIDGETS)
372 16         self._widget_group.handle_resize(window.size)
373 17
374 18         self._model.register_listener(self.process_model_event, 'pause')
375 19
376 20         self._event_to_func_map = {
377 21             GameEventType.PAUSE_CLICK: self.handle_pause_click
378 22         }
379 23
380 24         self.states = {
381 25             'PAUSED': False
382 26         }
383 27
384 28     def handle_pause_click(self, event):

```

```

29         self.states['PAUSED'] = not self.states['PAUSED']
30
31     if self.states['PAUSED']:
32         audio.pause_sfx()
33     else:
34         audio.unpause_sfx()
35
36     def handle_resize(self):
37         self._screen_overlay = pygame.Surface(window.size, pygame.SRCALPHA)
38         self._screen_overlay.fill(PAUSE_COLOUR)
39         self._widget_group.handle_resize(window.size)
40
41     def draw(self):
42         if self.states['PAUSED']:
43             window.screen.blit(self._screen_overlay, (0, 0))
44             self._widget_group.draw()
45
46     def process_model_event(self, event):
47         try:
48             self._event_to_func_map.get(event.type)(event)
49         except:
50             raise KeyError('Event type not recognized in Paused View (PauseView.
process_model_event)', event)
51
52     def convert_mouse_pos(self, event):
53         return self._widget_group.process_event(event)
54
55 from data.constants import Colour, Miscellaneous, CursorMode
56 from data.components.widget_group import WidgetGroup
57 from data.states.game.widget_dict import WIN_WIDGETS
58 from data.managers.window import window
59 from data.managers.cursor import cursor
60
61 class WinView:
62     def __init__(self, model):
63         self._model = model
64
65         self._widget_group = WidgetGroup(WIN_WIDGETS)
66         self._widget_group.handle_resize(window.size)
67
68     def handle_resize(self):
69         self._widget_group.handle_resize(window.size)
70
71     def draw(self):
72         if self._model.states['WINNER'] is not None:
73             if cursor.get_mode() != CursorMode.ARROW:
74                 cursor.set_mode(CursorMode.ARROW)
75
76             if self._model.states['WINNER'] == Colour.BLUE:
77                 WIN_WIDGETS['red_won'].kill()
78                 WIN_WIDGETS['draw_won'].kill()
79             elif self._model.states['WINNER'] == Colour.RED:
80                 WIN_WIDGETS['blue_won'].kill()
81                 WIN_WIDGETS['draw_won'].kill()
82             elif self._model.states['WINNER'] == Miscellaneous.DRAW:
83                 WIN_WIDGETS['red_won'].kill()
84                 WIN_WIDGETS['blue_won'].kill()
85
86             self._widget_group.draw()
87
88     def set_win_type(self, win_type):
89         WIN_WIDGETS['by_draw'].kill()

```

```

36     WIN_WIDGETS['by_timeout'].kill()
37     WIN_WIDGETS['by_resignation'].kill()
38     WIN_WIDGETS['by_checkmate'].kill()
39
40     match win_type:
41         case 'CAPTURE':
42             self._widget_group.add(WIN_WIDGETS['by_checkmate'])
43         case 'DRAW':
44             self._widget_group.add(WIN_WIDGETS['by_draw'])
45         case 'RESIGN':
46             self._widget_group.add(WIN_WIDGETS['by_resignation'])
47         case 'TIME':
48             self._widget_group.add(WIN_WIDGETS['by_timeout'])
49
50     def convert_mouse_pos(self, event):
51         return self._widget_group.process_event(event)

1 import pygame
2 import sys
3 from random import randint
4 from data.utils.asset_helpers import get_rotational_angle
5 from data.states.menu.widget_dict import 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
14
15 logger = initialise_logger(__file__)
16
17 class Menu(_State):
18     def __init__(self):
19         super().__init__()
20         self._fire_laser = False
21         self._bloom_mask = None
22         self._laser_mask = None
23
24     def cleanup(self):
25         super().cleanup()
26
27         window.clear_apply_arguments(ShaderType.BLOOM)
28         window.clear_apply_arguments(ShaderType.SHAKE)
29         window.clear_effect(ShaderType.CHROMATIC_ABBREVIATION)
30
31         return None
32
33     def startup(self, persist=None):
34         super().startup(MENU_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
35         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
36         window.set_effect(ShaderType.CHROMATIC_ABBREVIATION)
37
38         MENU_WIDGETS['credits'].kill()
39
40         self._fire_laser = False
41         self._bloom_mask = None
42         self._laser_mask = None
43
44         self.draw()

```



```

45         self.update_masks()
46
47     @property
48     def sphinx_center(self):
49         return (window.size[0] - self.sphinx_size[0] / 2, window.size[1] - self.
sphinx_size[1] / 2)
50
51     @property
52     def sphinx_size(self):
53         return (min(window.size) * 0.1, min(window.size) * 0.1)
54
55     @property
56     def sphinx_rotation(self):
57         mouse_pos = (pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1] + 0.01)
58         return -get_rotational_angle(mouse_pos, self.sphinx_center)
59
60     def get_event(self, event):
61         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
62             MENU_WIDGETS['credits'].kill()
63
64         if event.type == pygame.MOUSEBUTTONDOWN:
65             self._fire_laser = True
66             audio.play_sfx(SFX['menu_laser_windup'])
67             audio.play_sfx(SFX['menu_laser_loop'], loop=True)
68             animation.set_timer(SFX['menu_laser_loop'].get_length() * 1000 / 2,
lambda: audio.play_sfx(SFX['menu_laser_loop'], loop=True) if self._fire_laser
else ...) # OVERLAP TWO LOOPS TO HIDE TRANSITION
69
70         elif event.type == pygame.MOUSEBUTTONDOWN:
71             self._fire_laser = False
72
73         window.clear_effect(ShaderType.RAYS)
74         animation.set_timer(300, lambda: window.clear_effect(ShaderType.SHAKE)
)
75
76         audio.stop_sfx(1000)
77
78         widget_event = self._widget_group.process_event(event)
79
80         if widget_event is None:
81             return
82
83         match widget_event.type:
84             case None:
85                 return
86
87             case MenuEventType.CONFIG_CLICK:
88                 self.next = 'config'
89                 self.done = True
90             case MenuEventType.SETTINGS_CLICK:
91                 self.next = 'settings'
92                 self.done = True
93             case MenuEventType.BROWSER_CLICK:
94                 self.next = 'browser'
95                 self.done = True
96             case MenuEventType.QUIT_CLICK:
97                 pygame.quit()
98                 sys.exit()
99                 logger.info('quitting...')
100             case MenuEventType.CREDITS_CLICK:
101                 self._widget_group.add(MENU_WIDGETS['credits'])
102
103     def draw_sphinx(self):

```

```

103         sphinx_surface = scale_and_cache(GRAPHICS['sphinx_0_b'], self.sphinx_size)
104         sphinx_surface = pygame.transform.rotate(sphinx_surface, self.
sphinx_rotation)
105         sphinx_rect = pygame.Rect(0, 0, *self.sphinx_size)
106         sphinx_rect.center = self.sphinx_center
107
108         window.screen.blit(sphinx_surface, sphinx_rect)
109
110     def update_masks(self):
111         self.draw()
112
113         widget_mask = window.screen.copy()
114         laser_mask = pygame.mask.from_surface(widget_mask)
115         laser_mask = laser_mask.to_surface(setcolor=(255, 0, 0, 255), unsetcolor
=(0, 0, 0, 255))
116         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)
)
117         pygame.draw.rect(widget_mask, (0, 0, 0, 255), (window.screen.width - 50,
0, 50, 50))
118
119         self._bloom_mask = widget_mask
120         self._laser_mask = laser_mask
121
122     def draw(self):
123         self._widget_group.draw()
124         self.draw_sphinx()
125
126         if self._fire_laser:
127             window.set_apply_arguments(ShaderType.RAYS, occlusion=self._laser_mask
, softShadow=0.1)
128
129             window.set_apply_arguments(ShaderType.BLOOM, highlight_surface=self.
_bloom_mask, surface_intensity=0.3, brightness_intensity=0.6)
130
131     def update(self, **kwargs):
132         random_offset = lambda: randint(-5, 5) / 40
133         if self._fire_laser:
134             window.clear_effect(ShaderType.RAYS)
135             window.set_effect(ShaderType.RAYS, lights=[[
136                 (self.sphinx_center[0] / window.size[0], self.sphinx_center[1] /
window.size[1]),
137                 2.2,
138                 (190, 190, 255),
139                 0.99,
140                 (self.sphinx_rotation - 2 + random_offset(), self.sphinx_rotation
+ 2 + random_offset())
141             ]])
142
143             window.set_effect(ShaderType.SHAKE)
144             window.set_apply_arguments(ShaderType.SHAKE, intensity=1)
145             pygame.mouse.set_pos(pygame.mouse.get_pos()[0] + random_offset(),
pygame.mouse.get_pos()[1] + random_offset())
146
147             super().update(**kwargs)
148
149     def handle_resize(self):
150         super().handle_resize()
151         self.update_masks()

```

```

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 *
6
7 top_right_container = Rectangle(
8     relative_position=(0, 0),
9     relative_size=(0.15, 0.075),
10    fixed_position=(5, 5),
11    anchor_x='right',
12    scale_mode='height'
13 )
14
15 MENU_WIDGETS = {
16     'credits':
17         Icon(
18             relative_position=(0, 0),
19             relative_size=(0.7, 0.7),
20             icon=GRAPHICS['credits'],
21             anchor_x='center',
22             anchor_y='center',
23             margin=50
24         ),
25     'default': [
26         top_right_container,
27         ReactiveIconButton(
28             parent=top_right_container,
29             relative_position=(0, 0),
30             relative_size=(1, 1),
31             anchor_x='right',
32             scale_mode='height',
33             base_icon=GRAPHICS['quit_base'],
34             hover_icon=GRAPHICS['quit_hover'],
35             press_icon=GRAPHICS['quit_press'],
36             event=CustomEvent(MenuEventType.QUIT_CLICK)
37         ),
38         ReactiveIconButton(
39             parent=top_right_container,
40             relative_position=(0, 0),
41             relative_size=(1, 1),
42             scale_mode='height',
43             base_icon=GRAPHICS['credits_base'],
44             hover_icon=GRAPHICS['credits_hover'],
45             press_icon=GRAPHICS['credits_press'],
46             event=CustomEvent(MenuEventType.CREDITS_CLICK)
47         ),
48         ReactiveIconButton(
49             relative_position=(0.05, -0.2),
50             relative_size=(0, 0.15),
51             anchor_y='center',
52             base_icon=GRAPHICS['play_text_base'],
53             hover_icon=GRAPHICS['play_text_hover'],
54             press_icon=GRAPHICS['play_text_press'],
55             event=CustomEvent(MenuEventType.CONFIG_CLICK)
56         ),
57         ReactiveIconButton(
58             relative_position=(0.05, 0),
59             relative_size=(0, 0.15),
60             anchor_y='center',
61             base_icon=GRAPHICS['review_text_base'],
62             hover_icon=GRAPHICS['review_text_hover'],
63             press_icon=GRAPHICS['review_text_press'],
64             event=CustomEvent(MenuEventType.BROWSER_CLICK)

```

```

65     ),
66     ReactiveIconButton(
67         relative_position=(0.05, 0.2),
68         relative_size=(0, 0.15),
69         anchor_y='center',
70         base_icon=GRAPHICS['settings_text_base'],
71         hover_icon=GRAPHICS['settings_text_hover'],
72         press_icon=GRAPHICS['settings_text_press'],
73         event=CustomEvent(MenuEventType.SETTINGS_CLICK)
74     ),
75     Icon(
76         relative_position=(0.0, 0.1),
77         relative_size=(0.3, 0.2),
78         anchor_x='center',
79         fill_colour=theme['fillSecondary'],
80         icon=GRAPHICS['title_screen_art'],
81         stretch=False
82     ),
83 ]
84 }
85
86 # Widgets used for testing light rays effect
87 TEST_WIDGETS = {
88     'default': [
89         Rectangle(
90             relative_position=(0.4, 0.2),
91             relative_size=(0.1, 0.1),
92             scale_mode='height',
93             visible=True,
94             border_width=0,
95             fill_colour=(255, 0, 0),
96             border_radius=1000
97         ),
98         Rectangle(
99             relative_position=(0.5, 0.7),
100             relative_size=(0.1, 0.1),
101             scale_mode='height',
102             visible=True,
103             border_width=0,
104             fill_colour=(255, 0, 0),
105             border_radius=1000
106         ),
107         Rectangle(
108             relative_position=(0.6, 0.6),
109             relative_size=(0.2, 0.2),
110             scale_mode='height',
111             visible=True,
112             border_width=0,
113             fill_colour=(255, 0, 0),
114             border_radius=1000
115         ),
116         Rectangle(
117             relative_position=(0.4, 0.6),
118             relative_size=(0.1, 0.1),
119             scale_mode='height',
120             visible=True,
121             border_width=0,
122             fill_colour=(255, 0, 0),
123             border_radius=1000
124         ),
125         Rectangle(
126             relative_position=(0.6, 0.4),

```

```

127         relative_size=(0.1, 0.1),
128         scale_mode='height',
129         visible=True,
130         border_width=0,
131         fill_colour=(255, 0, 0),
132         border_radius=1000
133     ),
134     Rectangle(
135         relative_position=(0.3, 0.4),
136         relative_size=(0.1, 0.1),
137         scale_mode='height',
138         visible=True,
139         border_width=0,
140         fill_colour=(255, 0, 0),
141         border_radius=1000
142     ),
143     Rectangle(
144         relative_position=(0.475, 0.15),
145         relative_size=(0.2, 0.2),
146         scale_mode='height',
147         visible=True,
148         border_width=0,
149         fill_colour=(255, 0, 0),
150         border_radius=1000
151     ),
152     Rectangle(
153         relative_position=(0.6, 0.2),
154         relative_size=(0.1, 0.1),
155         scale_mode='height',
156         visible=True,
157         border_width=0,
158         fill_colour=(255, 0, 0),
159         border_radius=1000
160     )
161 ]
162 }

1 import pygame
2 from collections import deque
3 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
7 from data.utils.bitboard_helpers import bitboard_to_coords
8 from data.states.review.widget_dict import REVIEW_WIDGETS
9 from data.utils.browser_helpers import get_winner_string
10 from data.states.game.components.board import 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
16
17 logger = initialise_logger(__name__)
18
19 class Review(_State):
20     def __init__(self):
21         super().__init__()
22
23         self._moves = deque()
24         self._popped_moves = deque()
25         self._game_info = {}

```

```

26         self._board = None
27         self._piece_group = None
28         self._laser_draw = None
29         self._capture_draw = None
30
31     def cleanup(self):
32         """
33         Cleanup function. Clears shader effects.
34         """
35         super().cleanup()
36
37         window.clear_apply_arguments(ShaderType.BLOOM)
38         window.clear_effect(ShaderType.RAYS)
39
40         return None
41
42     def startup(self, persist):
43         """
44         Startup function. Initialises all objects, widgets and game data.
45
46         Args:
47             persist (dict): Dict containing game entry data.
48         """
49         super().startup(REVIEW_WIDGETS, MUSIC['review'])
50
51         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
52 BACKGROUND_WAVES)
53         window.set_apply_arguments(ShaderType.BLOOM, highlight_colours=[(pygame.
54 Color('0x95e0cc')).rgb, pygame.Color('0xf14e52').rgb], colour_intensity=0.8)
55         REVIEW_WIDGETS['help'].kill()
56
57         self._moves = deque(GameEntry.parse_moves(persist.pop('moves', '')))
58         self._popped_moves = deque()
59         self._game_info = persist
60
61         self._board = Board(self._game_info['start_fen_string'])
62         self._piece_group = PieceGroup()
63         self._laser_draw = LaserDraw(self.board_position, self.board_size)
64         self._capture_draw = CaptureDraw(self.board_position, self.board_size)
65
66         self.initialise_widgets()
67         self.simulate_all_moves()
68         self.refresh_pieces()
69         self.refresh_widgets()
70
71         self.draw()
72
73     @property
74     def board_position(self):
75         return REVIEW_WIDGETS['chessboard'].position
76
77     @property
78     def board_size(self):
79         return REVIEW_WIDGETS['chessboard'].size
80
81     @property
82     def square_size(self):
83         return self.board_size[0] / 10
84
85     def initialise_widgets(self):
86         """

```

```

86         Initializes the widgets for a new game.
87         """
88         REVIEW_WIDGETS['move_list'].reset_move_list()
89         REVIEW_WIDGETS['move_list'].kill()
90         REVIEW_WIDGETS['scroll_area'].set_image()
91
92         REVIEW_WIDGETS['winner_text'].set_text(f'WINNER: {get_winner_string(self.
_game_info["winner"])}')
93         REVIEW_WIDGETS['blue_piece_display'].reset_piece_list()
94         REVIEW_WIDGETS['red_piece_display'].reset_piece_list()
95
96         if self._game_info['time_enabled']:
97             REVIEW_WIDGETS['timer_disabled_text'].kill()
98         else:
99             REVIEW_WIDGETS['blue_timer'].kill()
100             REVIEW_WIDGETS['red_timer'].kill()
101
102     def refresh_widgets(self):
103         """
104         Refreshes the widgets after every move.
105         """
106         REVIEW_WIDGETS['move_number_text'].set_text(f'MOVE NO: {(len(self._moves))
/ 2:.1f} / {(len(self._moves) + len(self._popped_moves)) / 2:.1f}')
107         REVIEW_WIDGETS['move_colour_text'].set_text(f'{self.calculate_colour().
name} TO MOVE')
108
109         if self._game_info['time_enabled']:
110             if len(self._moves) == 0:
111                 REVIEW_WIDGETS['blue_timer'].set_time(float(self._game_info['time'
]) * 60 * 1000)
112                 REVIEW_WIDGETS['red_timer'].set_time(float(self._game_info['time'
]) * 60 * 1000)
113             else:
114                 REVIEW_WIDGETS['blue_timer'].set_time(float(self._moves[-1]['
blue_time']) * 60 * 1000)
115                 REVIEW_WIDGETS['red_timer'].set_time(float(self._moves[-1]['
red_time']) * 60 * 1000)
116
117         REVIEW_WIDGETS['scroll_area'].set_image()
118
119     def refresh_pieces(self):
120         """
121         Refreshes the pieces on the board.
122         """
123         self._piece_group.initialise_pieces(self._board.get_piece_list(), self.
board_position, self.board_size)
124
125     def simulate_all_moves(self):
126         """
127         Simulates all moves at the start of every game to obtain laser results and
fill up piece display and move list widgets.
128         """
129         for index, move_dict in enumerate(self._moves):
130             laser_result = self._board.apply_move(move_dict['move'], fire_laser=
True)
131             self._moves[index]['laser_result'] = laser_result
132
133             if laser_result.hit_square_bitboard:
134                 if laser_result.piece_colour == Colour.BLUE:
135                     REVIEW_WIDGETS['red_piece_display'].add_piece(laser_result.
piece_hit)
136                     elif laser_result.piece_colour == Colour.RED:

```

```

137         REVIEW_WIDGETS['blue_piece_display'].add_piece(laser_result.
piece_hit)
138
139         REVIEW_WIDGETS['move_list'].append_to_move_list(move_dict['
unparsed_move'])
140
141     def calculate_colour(self):
142         """
143         Calculates the current active colour to move.
144
145         Returns:
146             Colour: The current colour to move.
147         """
148         if self._game_info['start_fen_string'][-1].lower() == 'b':
149             initial_colour = Colour.BLUE
150         elif self._game_info['start_fen_string'][-1].lower() == 'r':
151             initial_colour = Colour.RED
152
153         if len(self._moves) % 2 == 0:
154             return initial_colour
155         else:
156             return initial_colour.get_flipped_colour()
157
158     def handle_move(self, move, add_piece=True):
159         """
160         Handles applying or undoing a move.
161
162         Args:
163             move (dict): The move to handle.
164             add_piece (bool): Whether to add the captured piece to the display.
165         Defaults to True.
166         """
167         laser_result = move['laser_result']
168         active_colour = self.calculate_colour()
169         self._laser_draw.add_laser(laser_result, laser_colour=active_colour)
170
171         if laser_result.hit_square_bitboard:
172             if laser_result.piece_colour == Colour.BLUE:
173                 if add_piece:
174                     REVIEW_WIDGETS['red_piece_display'].add_piece(laser_result.
piece_hit)
175                 else:
176                     REVIEW_WIDGETS['red_piece_display'].remove_piece(laser_result.
piece_hit)
177             elif laser_result.piece_colour == Colour.RED:
178                 if add_piece:
179                     REVIEW_WIDGETS['blue_piece_display'].add_piece(laser_result.
piece_hit)
180                 else:
181                     REVIEW_WIDGETS['blue_piece_display'].remove_piece(laser_result
.piece_hit)
182
183             self._capture_draw.add_capture(
184                 laser_result.piece_hit,
185                 laser_result.piece_colour,
186                 laser_result.piece_rotation,
187                 bitboard_to_coords(laser_result.hit_square_bitboard),
188                 laser_result.laser_path[0][0],
189                 active_colour,
190                 shake=False
191             )

```



```

192     def update_laser_mask(self):
193         """
194         Updates the laser mask for the light rays effect.
195         """
196         temp_surface = pygame.Surface(window.size, pygame.SRCALPHA)
197         self._piece_group.draw(temp_surface)
198         mask = pygame.mask.from_surface(temp_surface, threshold=127)
199         mask_surface = mask.to_surface(unsetcolor=(0, 0, 0, 255), setcolor=(255,
200         0, 0, 255))
201
202         window.set_apply_arguments(ShaderType.RAYS, occlusion=mask_surface)
203
204     def get_event(self, event):
205         """
206         Processes Pygame events.
207
208         Args:
209             event (pygame.event.Event): The event to handle.
210         """
211         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.KEYDOWN]:
212             REVIEW_WIDGETS['help'].kill()
213
214         widget_event = self._widget_group.process_event(event)
215
216         if widget_event is None:
217             return
218
219         match widget_event.type:
220             case None:
221                 return
222
223             case ReviewEventType.MENU_CLICK:
224                 self.next = 'menu'
225                 self.done = True
226
227             case ReviewEventType.PREVIOUS_CLICK:
228                 if len(self._moves) == 0:
229                     return
230
231                 # Pop last applied move off first stack
232                 move = self._moves.pop()
233                 # Pushed onto second stack
234                 self._popped_moves.append(move)
235
236                 # Undo last applied move
237                 self._board.undo_move(move['move'], laser_result=move['
238                 laser_result'])
239                 self.handle_move(move, add_piece=False)
240                 REVIEW_WIDGETS['move_list'].pop_from_move_list()
241
242                 self.refresh_pieces()
243                 self.refresh_widgets()
244                 self.update_laser_mask()
245
246             case ReviewEventType.NEXT_CLICK:
247                 if len(self._popped_moves) == 0:
248                     return
249
250                 # Peek at second stack to get last undone move
251                 move = self._popped_moves[-1]
252
253                 # Reapply last undone move

```

```

252         self._board.apply_move(move['move'])
253         self.handle_move(move, add_piece=True)
254         REVIEW_WIDGETS['move_list'].append_to_move_list(move['
unparsed_move'])
255
256         # Pop last undone move from second stack
257         self._popped_moves.pop()
258         # Push onto first stack
259         self._moves.append(move)
260
261         self.refresh_pieces()
262         self.refresh_widgets()
263         self.update_laser_mask()
264
265         case ReviewEventType.HELP_CLICK:
266             self._widget_group.add(REVIEW_WIDGETS['help'])
267             self._widget_group.handle_resize(window.size)
268
269     def handle_resize(self):
270         """
271         Handles resizing of the window.
272         """
273         super().handle_resize()
274         self._piece_group.handle_resize(self.board_position, self.board_size)
275         self._laser_draw.handle_resize(self.board_position, self.board_size)
276         self._capture_draw.handle_resize(self.board_position, self.board_size)
277
278         if self._laser_draw.firing:
279             self.update_laser_mask()
280
281     def draw(self):
282         """
283         Draws all components onto the window screen.
284         """
285         self._capture_draw.update()
286         self._widget_group.draw()
287         self._piece_group.draw(window.screen)
288         self._laser_draw.draw(window.screen)
289         self._capture_draw.draw(window.screen)
290
291 1 from data.widgets import *
292 2 from data.components.custom_event import CustomEvent
293 3 from data.constants import ReviewEventType, Colour
294 4 from data.assets import GRAPHICS
295 5
296 6 MOVE_LIST_WIDTH = 0.2
297 7
298 8 right_container = Rectangle(
299 9     relative_position=(0.05, 0),
300 10    relative_size=(0.2, 0.7),
301 11    anchor_y='center',
302 12    anchor_x='right'
303 13 )
304 14
305 15 info_container = Rectangle(
306 16    parent=right_container,
307 17    relative_position=(0, 0.5),
308 18    relative_size=(1, 0.5),
309 19    visible=True
310 20 )
311 21
312 22 arrow_container = Rectangle(

```

```

23     relative_position=(0, 0.05),
24     relative_size=(0.4, 0.1),
25     anchor_x='center',
26     anchor_y='bottom'
27 )
28
29 move_list = MoveList(
30     parent=right_container,
31     relative_position=(0, 0),
32     relative_width=1,
33     minimum_height=300,
34     move_list=[]
35 )
36
37 top_right_container = Rectangle(
38     relative_position=(0, 0),
39     relative_size=(0.15, 0.075),
40     fixed_position=(5, 5),
41     anchor_x='right',
42     scale_mode='height'
43 )
44
45 REVIEW_WIDGETS = {
46     'help':
47         Icon(
48             relative_position=(0, 0),
49             relative_size=(1.02, 1.02),
50             icon=GRAPHICS['review_help'],
51             anchor_x='center',
52             anchor_y='center',
53             border_width=0,
54             fill_colour=(0, 0, 0, 0)
55         ),
56     'default': [
57         arrow_container,
58         right_container,
59         info_container,
60         top_right_container,
61         ReactiveIconButton(
62             parent=top_right_container,
63             relative_position=(0, 0),
64             relative_size=(1, 1),
65             anchor_x='right',
66             scale_mode='height',
67             base_icon=GRAPHICS['home_base'],
68             hover_icon=GRAPHICS['home_hover'],
69             press_icon=GRAPHICS['home_press'],
70             event=CustomEvent(ReviewEventType.MENU_CLICK)
71         ),
72         ReactiveIconButton(
73             parent=top_right_container,
74             relative_position=(0, 0),
75             relative_size=(1, 1),
76             scale_mode='height',
77             base_icon=GRAPHICS['help_base'],
78             hover_icon=GRAPHICS['help_hover'],
79             press_icon=GRAPHICS['help_press'],
80             event=CustomEvent(ReviewEventType.HELP_CLICK)
81         ),
82         ReactiveIconButton(
83             parent=arrow_container,
84             relative_position=(0, 0),

```

```

85         relative_size=(1, 1),
86         scale_mode='height',
87         base_icon=GRAPHICS['left_arrow_filled_base'],
88         hover_icon=GRAPHICS['left_arrow_filled_hover'],
89         press_icon=GRAPHICS['left_arrow_filled_press'],
90         event=CustomEvent(ReviewEventType.PREVIOUS_CLICK)
91     ),
92     ReactiveIconButton(
93         parent=arrow_container,
94         relative_position=(0, 0),
95         relative_size=(1, 1),
96         scale_mode='height',
97         anchor_x='right',
98         base_icon=GRAPHICS['right_arrow_filled_base'],
99         hover_icon=GRAPHICS['right_arrow_filled_hover'],
100        press_icon=GRAPHICS['right_arrow_filled_press'],
101        event=CustomEvent(ReviewEventType.NEXT_CLICK)
102    ),
103 ],
104 'move_list':
105     move_list,
106 'scroll_area':
107     ScrollArea(
108         parent=right_container,
109         relative_position=(0, 0),
110         relative_size=(1, 0.5),
111         vertical=True,
112         widget=move_list
113     ),
114 'chessboard':
115     Chessboard(
116         relative_position=(0, 0),
117         relative_width=0.4,
118         scale_mode='width',
119         anchor_x='center',
120         anchor_y='center'
121     ),
122 'move_number_text':
123     Text(
124         parent=info_container,
125         relative_position=(0, 0),
126         relative_size=(1, 0.3),
127         anchor_y='bottom',
128         text='MOVE NO:',
129         fit_vertical=False,
130         margin=10,
131         border_width=0,
132         fill_colour=(0, 0, 0, 0),
133     ),
134 'move_colour_text':
135     Text(
136         parent=info_container,
137         relative_size=(1, 0.3),
138         relative_position=(0, 0),
139         anchor_y='center',
140         text='TO MOVE',
141         fit_vertical=False,
142         margin=10,
143         border_width=0,
144         fill_colour=(0, 0, 0, 0),
145     ),
146 'winner_text':

```

```

147     Text(
148         parent=info_container,
149         relative_size=(1, 0.3),
150         relative_position=(0, 0),
151         text='WINNER:',
152         fit_vertical=False,
153         margin=10,
154         border_width=0,
155         fill_colour=(0, 0, 0, 0),
156     ),
157     'blue_timer':
158     Timer(
159         relative_position=(0.05, 0.05),
160         anchor_y='center',
161         relative_size=(0.1, 0.1),
162         active_colour=Colour.BLUE,
163     ),
164     'red_timer':
165     Timer(
166         relative_position=(0.05, -0.05),
167         anchor_y='center',
168         relative_size=(0.1, 0.1),
169         active_colour=Colour.RED
170     ),
171     'timer_disabled_text':
172     Text(
173         relative_size=(0.2, 0.1),
174         relative_position=(0.05, 0),
175         anchor_y='center',
176         fit_vertical=False,
177         text='TIMER DISABLED',
178     ),
179     'blue_piece_display':
180     PieceDisplay(
181         relative_position=(0.05, 0.05),
182         relative_size=(0.2, 0.1),
183         anchor_y='bottom',
184         active_colour=Colour.BLUE
185     ),
186     'red_piece_display':
187     PieceDisplay(
188         relative_position=(0.05, 0.05),
189         relative_size=(0.2, 0.1),
190         active_colour=Colour.RED
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
4  from data.constants import SettingsEventType, WidgetState, ShaderType, SHADER_MAP
5  from data.states.settings.widget_dict import 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
12
13 logger = initialise_logger(__name__)
14

```

```

15 class Settings(_State):
16     def __init__(self):
17         super().__init__()
18
19         self._colour_picker = None
20         self._settings = None
21
22     def cleanup(self):
23         super().cleanup()
24
25         update_user_settings(self._settings)
26
27         return None
28
29     def startup(self, persist=None):
30         super().startup(SETTINGS_WIDGETS, music=MUSIC[f'menu_{randint(1, 3)}'])
31
32         window.set_apply_arguments(ShaderType.BASE, background_type=ShaderType.
BACKGROUND_BALATRO)
33         self._settings = get_user_settings()
34         self.reload_settings()
35
36         self.draw()
37
38     def create_colour_picker(self, mouse_pos, button_type):
39         if button_type == SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK:
40             selected_colour = self._settings['primaryBoardColour']
41             event_type = SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK
42         else:
43             selected_colour = self._settings['secondaryBoardColour']
44             event_type = SettingsEventType.SECONDARY_COLOUR_PICKER_CLICK
45
46         self._colour_picker = ColourPicker(
47             relative_position=(mouse_pos[0] / window.size[0], mouse_pos[1] /
window.size[1]),
48             relative_width=0.15,
49             selected_colour=selected_colour,
50             event_type=event_type
51         )
52         self._widget_group.add(self._colour_picker)
53
54     def remove_colour_picker(self):
55         self._colour_picker.kill()
56
57     def reload_display_mode(self):
58         relative_mouse_pos = (pygame.mouse.get_pos()[0] / window.size[0], pygame.
mouse.get_pos()[1] / window.size[1])
59
60         if self._settings['displayMode'] == 'fullscreen':
61             window.set_fullscreen(desktop=True)
62             window.handle_resize()
63
64         elif self._settings['displayMode'] == 'windowed':
65             window.set_windowed()
66             window.handle_resize()
67             window.restore()
68
69         self._widget_group.handle_resize(window.size)
70
71         new_mouse_pos = (relative_mouse_pos[0] * window.size[0],
relative_mouse_pos[1] * window.size[1])
72         pygame.mouse.set_pos(new_mouse_pos)

```

```

73
74 def reload_shaders(self):
75     window.clear_all_effects()
76
77     for shader_type in SHADER_MAP[self._settings['shader']]:
78         window.set_effect(shader_type)
79
80 def reload_settings(self):
81     SETTINGS_WIDGETS['primary_colour_button'].initialise_new_colours(self.
82 _settings['primaryBoardColour'])
83     SETTINGS_WIDGETS['secondary_colour_button'].initialise_new_colours(self.
84 _settings['secondaryBoardColour'])
85     SETTINGS_WIDGETS['primary_colour_button'].set_state_colour(WidgetState.
86 BASE)
87     SETTINGS_WIDGETS['secondary_colour_button'].set_state_colour(WidgetState.
88 BASE)
89     SETTINGS_WIDGETS['music_volume_slider'].set_volume(self._settings['
90 musicVolume'])
91     SETTINGS_WIDGETS['sfx_volume_slider'].set_volume(self._settings['sfxVolume
92 ''])
93     SETTINGS_WIDGETS['display_mode_dropdown'].set_selected_word(self._settings
94 ['displayMode'])
95     SETTINGS_WIDGETS['shader_carousel'].set_to_key(self._settings['shader'])
96     SETTINGS_WIDGETS['particles_switch'].set_toggle_state(self._settings['
97 particles'])
98     SETTINGS_WIDGETS['opengl_switch'].set_toggle_state(self._settings['opengl'
99 ''])
100
101     self.reload_shaders()
102     self.reload_display_mode()
103
104 def get_event(self, event):
105     widget_event = self._widget_group.process_event(event)
106
107     if widget_event is None:
108         if event.type == pygame.MOUSEBUTTONDOWN and self._colour_picker:
109             self.remove_colour_picker()
110             return
111
112     match widget_event.type:
113         case SettingsEventType.VOLUME_SLIDER_SLIDE:
114             return
115
116         case SettingsEventType.VOLUME_SLIDER_CLICK:
117             if widget_event.volume_type == 'music':
118                 audio.set_music_volume(widget_event.volume)
119                 self._settings['musicVolume'] = widget_event.volume
120             elif widget_event.volume_type == 'sfx':
121                 audio.set_sfx_volume(widget_event.volume)
122                 self._settings['sfxVolume'] = widget_event.volume
123
124         case SettingsEventType.DROPDOWN_CLICK:
125             selected_word = SETTINGS_WIDGETS['display_mode_dropdown'].
126 get_selected_word()
127
128             if selected_word is None or selected_word == self._settings['
129 displayMode']:
130                 return
131
132             self._settings['displayMode'] = selected_word
133
134             self.reload_display_mode()

```

```

124
125         case SettingsEventType.MENU_CLICK:
126             self.next = 'menu'
127             self.done = True
128
129         case SettingsEventType.RESET_DEFAULT:
130             self._settings = get_default_settings()
131             self.reload_settings()
132
133         case SettingsEventType.RESET_USER:
134             self._settings = get_user_settings()
135             self.reload_settings()
136
137         case SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK | SettingsEventType
138         .SECONDARY_COLOUR_BUTTON_CLICK:
139             if self._colour_picker:
140                 self.remove_colour_picker()
141
142                 self.create_colour_picker(event.pos, widget_event.type)
143
144         case SettingsEventType.PRIMARY_COLOUR_PICKER_CLICK | SettingsEventType
145         .SECONDARY_COLOUR_PICKER_CLICK:
146             if widget_event.colour:
147                 r, g, b = widget_event.colour.rgb
148                 hex_colour = f'0x{hex(r)[2:].zfill(2)}{hex(g)[2:].zfill(2)}{
149                 hex(b)[2:].zfill(2)}'
150
151                 if widget_event.type == SettingsEventType.
152                 PRIMARY_COLOUR_PICKER_CLICK:
153                     SETTINGS_WIDGETS['primary_colour_button'].
154                     initialise_new_colours(widget_event.colour)
155                     SETTINGS_WIDGETS['primary_colour_button'].set_state_colour
156                     (WidgetState.BASE)
157                     self._settings['primaryBoardColour'] = hex_colour
158                 elif widget_event.type == SettingsEventType.
159                 SECONDARY_COLOUR_PICKER_CLICK:
160                     SETTINGS_WIDGETS['secondary_colour_button'].
161                     initialise_new_colours(widget_event.colour)
162                     SETTINGS_WIDGETS['secondary_colour_button'].
163                     set_state_colour(WidgetState.BASE)
164                     self._settings['secondaryBoardColour'] = hex_colour
165
166         case SettingsEventType.SHADER_PICKER_CLICK:
167             self._settings['shader'] = widget_event.data
168             self.reload_shaders()
169
170         case SettingsEventType.OPENGL_CLICK:
171             self._settings['opengl'] = widget_event.toggled
172             self.reload_shaders()
173
174         case SettingsEventType.PARTICLES_CLICK:
175             self._settings['particles'] = widget_event.toggled
176
177     def draw(self):
178         self._widget_group.draw()
179
180
181 1 from data.widgets import *
182 2 from data.components.custom_event import CustomEvent
183 3 from data.constants import SettingsEventType, SHADER_MAP
184 4 from data.utils.data_helpers import get_user_settings
185 5 from data.assets import GRAPHICS, DEFAULT_FONT
186 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
9
10 user_settings = get_user_settings()
11 # font_size = text_width_to_font_size('Shaders (OPENGL GPU REQUIRED)',
12   DEFAULT_FONT, 0.4 * window.screen.width)
13 FONT_SIZE = 21
14
15 carousel_widgets = {
16     key: Text(
17         relative_position=(0, 0),
18         relative_size=(0.25, 0.04),
19         margin=0,
20         text=key.replace('_', ' ').upper(),
21         fit_vertical=True,
22         border_width=0,
23         fill_colour=(0, 0, 0, 0),
24     ) for key in SHADER_MAP.keys()
25 }
26
27 reset_container = Rectangle(
28     relative_size=(0.2, 0.2),
29     relative_position=(0, 0),
30     fixed_position=(5, 5),
31     anchor_x='right',
32     anchor_y='bottom',
33 )
34
35 SETTINGS_WIDGETS = {
36     'default': [
37         reset_container,
38         ReactiveIconButton(
39             relative_position=(0, 0),
40             relative_size=(0.075, 0.075),
41             anchor_x='right',
42             scale_mode='height',
43             base_icon=GRAPHICS['home_base'],
44             hover_icon=GRAPHICS['home_hover'],
45             press_icon=GRAPHICS['home_press'],
46             fixed_position=(5, 5),
47             event=CustomEvent(SettingsEventType.MENU_CLICK)
48         ),
49         Text(
50             relative_position=(0.01, 0.1),
51             text='Display mode',
52             relative_size=(0.4, 0.04),
53             center=False,
54             border_width=0,
55             margin=0,
56             font_size=21,
57             fill_colour=(0, 0, 0, 0)
58         ),
59         Text(
60             relative_position=(0.01, 0.2),
61             text='Music',
62             relative_size=(0.4, 0.04),
63             center=False,
64             border_width=0,
65             margin=0,
66             font_size=21,
67             fill_colour=(0, 0, 0, 0)
68         ),
69     ],
70 }

```

```

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

```

```

130         relative_position=(0, 0),
131         relative_size=(1, 0.5),
132         fit_vertical=False,
133         margin=10,
134         text='DISCARD CHANGES',
135         text_colour=theme['textSecondary'],
136         event=CustomEvent(SettingsEventType.RESET_USER)
137     ),
138     TextButton(
139         parent=reset_container,
140         relative_position=(0, 0.5),
141         relative_size=(1, 0.5),
142         fit_vertical=False,
143         margin=10,
144         text='RESET TO DEFAULT',
145         text_colour=theme['textSecondary'],
146         event=CustomEvent(SettingsEventType.RESET_DEFAULT)
147     )
148 ],
149 'display_mode_dropdown':
150 Dropdown(
151     relative_position=(0.4, 0.1),
152     relative_width=0.2,
153     word_list=['fullscreen', 'windowed'],
154     fill_colour=(255, 100, 100),
155     event=CustomEvent(SettingsEventType.DROPDOWN_CLICK)
156 ),
157 'primary_colour_button':
158 ColourButton(
159     relative_position=(0.4, 0.4),
160     relative_size=(0.08, 0.05),
161     fill_colour=user_settings['primaryBoardColour'],
162     border_width=5,
163     event=CustomEvent(SettingsEventType.PRIMARY_COLOUR_BUTTON_CLICK)
164 ),
165 'secondary_colour_button':
166 ColourButton(
167     relative_position=(0.4, 0.5),
168     relative_size=(0.08, 0.05),
169     fill_colour=user_settings['secondaryBoardColour'],
170     border_width=5,
171     event=CustomEvent(SettingsEventType.SECONDARY_COLOUR_BUTTON_CLICK)
172 ),
173 'music_volume_slider':
174 VolumeSlider(
175     relative_position=(0.4, 0.2),
176     relative_length=(0.5),
177     default_volume=user_settings['musicVolume'],
178     border_width=5,
179     volume_type='music'
180 ),
181 'sfx_volume_slider':
182 VolumeSlider(
183     relative_position=(0.4, 0.3),
184     relative_length=(0.5),
185     default_volume=user_settings['sfxVolume'],
186     border_width=5,
187     volume_type='sfx'
188 ),
189 'shader_carousel':
190 Carousel(
191     relative_position = (0.4, 0.8),

```

```

192         margin=5,
193         border_width=0,
194         fill_colour=(0, 0, 0, 0),
195         widgets_dict=carousel_widgets,
196         event=CustomEvent(SettingsEventType.SHADER_PICKER_CLICK),
197     ),
198     'particles_switch':
199     Switch(
200         relative_position=(0.4, 0.6),
201         relative_height=0.04,
202         event=CustomEvent(SettingsEventType.PARTICLES_CLICK)
203     ),
204     'opengl_switch':
205     Switch(
206         relative_position=(0.4, 0.7),
207         relative_height=0.04,
208         event=CustomEvent(SettingsEventType.OPENGL_CLICK)
209     ),
210 }

1 import pygame
2 from PIL import Image
3 from functools import cache
4 from random import sample, randint
5 import math
6
7 @cache
8 def scale_and_cache(image, target_size):
9     """
10     Caches image when resized repeatedly.
11
12     Args:
13         image (pygame.Surface): Image surface to be resized.
14         target_size (tuple[float, float]): New image size.
15
16     Returns:
17         pygame.Surface: Resized image surface.
18     """
19     return pygame.transform.scale(image, target_size)
20
21 @cache
22 def smoothscale_and_cache(image, target_size):
23     """
24     Same as scale_and_cache, but with the Pygame smoothscale function.
25
26     Args:
27         image (pygame.Surface): Image surface to be resized.
28         target_size (tuple[float, float]): New image size.
29
30     Returns:
31         pygame.Surface: Resized image surface.
32     """
33     return pygame.transform.smoothscale(image, target_size)
34
35 def gif_to_frames(path):
36     """
37     Uses the PIL library to break down GIFs into individual frames.
38
39     Args:
40         path (str): Directory path to GIF file.
41
42     Yields:

```

```

43     PIL.Image: Single frame.
44     """
45     try:
46         image = Image.open(path)
47
48         first_frame = image.copy().convert('RGBA')
49         yield first_frame
50         image.seek(1)
51
52         while True:
53             current_frame = image.copy()
54             yield current_frame
55             image.seek(image.tell() + 1)
56     except EOFError:
57         pass
58
59 def get_perimeter_sample(image_size, number):
60     """
61     Used for particle drawing class, generates roughly equally distributed points
62     around a rectangular image surface's perimeter.
63
64     Args:
65         image_size (tuple[float, float]): Image surface size.
66         number (int): Number of points to be generated.
67
68     Returns:
69         list[tuple[int, int], ...]: List of random points on perimeter of image
70         surface.
71     """
72     perimeter = 2 * (image_size[0] + image_size[1])
73     # Flatten perimeter to a single number representing the distance from the top-
74     # middle of the surface going clockwise, and create a list of equally spaced
75     # points
76     perimeter_offsets = [(image_size[0] / 2) + (i * perimeter / number) for i in
77                          range(0, number)]
78     pos_list = []
79
80     for perimeter_offset in perimeter_offsets:
81         # For every point, add a random offset
82         max_displacement = int(perimeter / (number * 4))
83         perimeter_offset += randint(-max_displacement, max_displacement)
84
85         if perimeter_offset > perimeter:
86             perimeter_offset -= perimeter
87
88         # Convert 1D distance back into 2D points on image surface perimeter
89         if perimeter_offset < image_size[0]:
90             pos_list.append((perimeter_offset, 0))
91         elif perimeter_offset < image_size[0] + image_size[1]:
92             pos_list.append((image_size[0], perimeter_offset - image_size[0]))
93         elif perimeter_offset < image_size[0] + image_size[1] + image_size[0]:
94             pos_list.append((perimeter_offset - image_size[0] - image_size[1],
95                             image_size[1]))
96         else:
97             pos_list.append((0, perimeter - perimeter_offset))
98     return pos_list
99
100 def get_angle_between_vectors(u, v, deg=True):
101     """
102     Uses the dot product formula to find the angle between two vectors.
103
104     Args:

```

```

99         u (list[int, int]): Vector 1.
100         v (list[int, int]): Vector 2.
101         deg (bool, optional): Return results in degrees. Defaults to True.
102
103     Returns:
104         float: Angle between vectors.
105     """
106     dot_product = sum(i * j for (i, j) in zip(u, v))
107     u_magnitude = math.sqrt(u[0] ** 2 + u[1] ** 2)
108     v_magnitude = math.sqrt(v[0] ** 2 + v[1] ** 2)
109
110     cos_angle = dot_product / (u_magnitude * v_magnitude)
111     radians = math.acos(min(max(cos_angle, -1), 1))
112
113     if deg:
114         return math.degrees(radians)
115     else:
116         return radians
117
118 def get_rotational_angle(u, v, deg=True):
119     """
120     Get bearing angle relative to positive x-axis centered on second vector.
121
122     Args:
123         u (list[int, int]): Vector 1.
124         v (list[int, int]): Vector 2, set as center of axes.
125         deg (bool, optional): Return results in degrees. Defaults to True.
126
127     Returns:
128         float: Bearing angle between vectors.
129     """
130     radians = math.atan2(u[1] - v[1], u[0] - v[0])
131
132     if deg:
133         return math.degrees(radians)
134     else:
135         return radians
136
137 def get_vector(src_vertex, dest_vertex):
138     """
139     Get vector describing translation between two points.
140
141     Args:
142         src_vertex (list[int, int]): Source vertex.
143         dest_vertex (list[int, int]): Destination vertex.
144
145     Returns:
146         tuple[int, int]: Vector between the two points.
147     """
148     return (dest_vertex[0] - src_vertex[0], dest_vertex[1] - src_vertex[1])
149
150 def get_next_corner(vertex, image_size):
151     """
152     Used in particle drawing system, finds coordinates of the next corner going
153     clockwise, given a point on the perimeter.
154
155     Args:
156         vertex (list[int, int]): Point on perimeter.
157         image_size (list[int, int]): Image size.
158
159     Returns:
160         list[int, int]: Coordinates of corner on perimeter.

```

```

160     """
161     corners = [(0, 0), (image_size[0], 0), (image_size[0], image_size[1]), (0,
image_size[1])]
162
163     if vertex in corners:
164         return corners[(corners.index(vertex) + 1) % len(corners)]
165
166     if vertex[1] == 0:
167         return (image_size[0], 0)
168     elif vertex[0] == image_size[0]:
169         return image_size
170     elif vertex[1] == image_size[1]:
171         return (0, image_size[1])
172     elif vertex[0] == 0:
173         return (0, 0)
174
175 def pil_image_to_surface(pil_image):
176     """
177     Args:
178         pil_image (PIL.Image): Image to be converted.
179
180     Returns:
181         pygame.Surface: Converted image surface.
182     """
183     return pygame.image.frombytes(pil_image.tobytes(), pil_image.size, pil_image.
mode).convert()
184
185 def calculate_frame_index(elapsed_milliseconds, start_index, end_index, fps):
186     """
187     Determine frame of animated GIF to be displayed.
188
189     Args:
190         elapsed_milliseconds (int): Milliseconds since GIF started playing.
191         start_index (int): Start frame of GIF.
192         end_index (int): End frame of GIF.
193         fps (int): Number of frames to be played per second.
194
195     Returns:
196         int: Displayed frame index of GIF.
197     """
198     ms_per_frame = int(1000 / fps)
199     return start_index + ((elapsed_milliseconds // ms_per_frame) % (end_index -
start_index))
200
201 def draw_background(screen, background, current_time=0):
202     """
203     Draws background to screen
204
205     Args:
206         screen (pygame.Surface): Screen to be drawn to
207         background (list[pygame.Surface, ...] | pygame.Surface): Background to be
drawn, if GIF, list of surfaces indexed to select frame to be drawn
208         current_time (int, optional): Used to calculate frame index for GIF.
Defaults to 0.
209     """
210     if isinstance(background, list):
211         # Animated background passed in as list of surfaces, calculate_frame_index
() used to get index of frame to be drawn
212         frame_index = calculate_frame_index(current_time, 0, len(background), fps
=8)
213         scaled_background = scale_and_cache(background[frame_index], screen.size)
214         screen.blit(scaled_background, (0, 0))

```

```

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

```

```

1  from data.constants import Rank, File, EMPTY_BB
2  from data.managers.logs import initialise_logger
3
4  logger = initialise_logger(__name__)
5
6  def print_bitboard(bitboard):
7      if (bitboard >= (2 ** 80)):
8          raise ValueError('Invalid bitboard: too many bits')
9
10     characters = ''
11     for rank in reversed(Rank):
12
13         for file in File:
14             mask = 1 << (rank * 10 + file)
15             if (bitboard & mask) != 0:
16                 characters += '1 '
17             else:
18                 characters += '. '
19
20         characters += '\n\n'
21
22     logger.info('\n' + characters + '\n')
23
24 def is_occupied(bitboard, target_bitboard):
25     return (target_bitboard & bitboard) != EMPTY_BB
26
27 def clear_square(bitboard, target_bitboard):
28     return (~target_bitboard & bitboard)
29
30 def set_square(bitboard, target_bitboard):
31     return (target_bitboard | bitboard)
32
33 def index_to_bitboard(index):
34     return (1 << index)
35
36 def coords_to_bitboard(coords):
37     index = coords[1] * 10 + coords[0]
38     return index_to_bitboard(index)
39
40 def bitboard_to_notation(bitboard):
41     index = bitboard_to_index(bitboard)

```



```

42     x = index // 10
43     y = index % 10
44
45     return chr(y + 97) + str(x + 1)
46
47 def notation_to_bitboard(notation):
48     index = (int(notation[1]) - 1) * 10 + int(ord(notation[0])) - 97
49
50     return index_to_bitboard(index)
51
52 def bitboard_to_index(bitboard):
53     return bitboard.bit_length() - 1
54
55 def bitboard_to_coords(bitboard):
56     list_position = bitboard_to_index(bitboard)
57     x = list_position % 10
58     y = list_position // 10
59
60     return x, y
61
62 def bitboard_to_coords_list(bitboard):
63     list_positions = []
64
65     for square in occupied_squares(bitboard):
66         list_positions.append(bitboard_to_coords(square))
67
68     return list_positions
69
70 def occupied_squares(bitboard):
71     while bitboard:
72         lsb_square = bitboard & -bitboard
73         bitboard = bitboard ^ lsb_square
74
75         yield lsb_square
76
77 def pop_count(bitboard):
78     count = 0
79     while bitboard:
80         count += 1
81         lsb_square = bitboard & -bitboard
82         bitboard = bitboard ^ lsb_square
83
84     return count
85
86 # def pop_count(bitboard):
87 #     count = 0
88 #     while bitboard:
89 #         count += 1
90 #         bitboard &= bitboard - 1
91
92 #     return count
93
94 def loop_all_squares():
95     for i in range(80):
96         yield 1 << i
97
98 #Solar
99 def get_LSB_value(bitboard: int):
100     return bitboard & -bitboard
101
102 def pop_count_2(bitboard):
103     count = 0

```

```

104     while bitboard > 0:
105         lsb_value = get_LSB_value(bitboard)
106         count += 1
107         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
4
5 user_settings = get_user_settings()
6
7 def create_board(board_size, primary_colour, secondary_colour, font=DEFAULT_FONT):
8     square_size = board_size[0] / 10
9     board_surface = pygame.Surface(board_size)
10
11     for i in range(80):
12         x = i % 10
13         y = i // 10
14
15         if (x + y) % 2 == 0:
16             square_colour = primary_colour
17         else:
18             square_colour = secondary_colour
19
20         square_x = x * square_size
21         square_y = y * square_size
22
23         pygame.draw.rect(board_surface, square_colour, (square_x, square_y,
24 square_size + 1, square_size + 1)) # +1 to fill in black lines
25
26         if y == 7:
27             text_position = (square_x + square_size * 0.7, square_y + square_size
28 * 0.55)
29             text_size = square_size / 3
30             font.render_to(board_surface, text_position, str(chr(x + 1 + 96)),
31 fgcolor=(10, 10, 10, 175), size=text_size)
32             if x == 0:
33                 text_position = (square_x + square_size * 0.1, square_y + square_size
34 * 0.1)
35                 text_size = square_size / 3
36                 font.render_to(board_surface, text_position, str(7-y + 1), fgcolor
37 =(10, 10, 10, 175), size=text_size)
38
39     return board_surface
40
41 def create_square_overlay(square_size, colour):
42     overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
43     overlay.fill(colour)
44
45     return overlay
46
47 def create_circle_overlay(square_size, colour):
48     overlay = pygame.Surface((square_size, square_size), pygame.SRCALPHA)
49     pygame.draw.circle(overlay, colour, (square_size / 2, square_size / 2),
50 square_size / 4)
51
52     return overlay
53
54 def coords_to_screen_pos(coords, board_position, square_size):
55     x = board_position[0] + (coords[0] * square_size)

```

```

50     y = board_position[1] + ((7 - coords[1]) * square_size)
51
52     return (x, y)
53
54 def screen_pos_to_coords(mouse_position, board_position, board_size):
55     if (board_position[0] <= mouse_position[0] <= board_position[0] + board_size
56         [0]) and (board_position[1] <= mouse_position[1] <= board_position[1] +
57         board_size[1]):
58         x = (mouse_position[0] - board_position[0]) // (board_size[0] / 10)
59         y = (board_size[1] - (mouse_position[1] - board_position[1])) // (
60             board_size[0] / 10)
61         return (int(x), int(y))
62
63     return None
64
65
66 1 from data.constants import Miscellaneous, Colour
67 2
68 3 def get_winner_string(winner):
69 4     if winner is None:
70 5         return 'UNFINISHED'
71 6     elif winner == Miscellaneous.DRAW:
72 7         return 'DRAW'
73 8     else:
74 9         return Colour(winner).name
75
76
77 1 import sqlite3
78 2 from pathlib import Path
79 3 from datetime import datetime
80 4
81 database_path = (Path(__file__).parent / '../database/database.db').resolve()
82
83 5 def insert_into_games(game_entry):
84 6     """
85 7     Inserts a new row into games table.
86 8
87 9     Args:
88 10         game_entry (GameEntry): GameEntry object containing game information.
89 11     """
90 12     connection = sqlite3.connect(database_path, detect_types=sqlite3.
91 13         PARSE_DECLTYPES)
92 14     connection.row_factory = sqlite3.Row
93 15     cursor = connection.cursor()
94 16
95 17     # Datetime added for created_dt column
96 18     game_entry = (*game_entry, datetime.now())
97 19
98 20
99 21     cursor.execute('''
100 22         INSERT INTO games (cpu_enabled, cpu_depth, winner, time_enabled, time,
101 23         number_of_ply, moves, start_fen_string, final_fen_string, created_dt)
102 24         VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
103 25     ''', game_entry)
104 26
105 27     connection.commit()
106 28
107 29     # Return inserted row
108 30     cursor.execute('''
109 31         SELECT * FROM games WHERE id = LAST_INSERT_ROWID()
110 32     ''')
111 33     inserted_row = cursor.fetchone()
112 34
113 35     connection.close()

```

```

36     return dict(inserted_row)
37
38 def get_all_games():
39     """
40     Get all rows in games table.
41
42     Returns:
43         list[dict]: List of game entries represented as dictionaries.
44     """
45     connection = sqlite3.connect(database_path, detect_types=sqlite3.
46     PARSE_DECLTYPES)
47     connection.row_factory = sqlite3.Row
48     cursor = connection.cursor()
49
50     cursor.execute('''
51         SELECT * FROM games
52     ''')
53     games = cursor.fetchall()
54
55     connection.close()
56
57     return [dict(game) for game in games]
58
59 def delete_all_games():
60     """
61     Delete all rows in games table.
62     """
63     connection = sqlite3.connect(database_path)
64     cursor = connection.cursor()
65
66     cursor.execute('''
67         DELETE FROM games
68     ''')
69
70     connection.commit()
71     connection.close()
72
73 def delete_game(id):
74     """
75     Deletes specific row in games table using id attribute.
76
77     Args:
78         id (int): Primary key for row.
79     """
80     connection = sqlite3.connect(database_path)
81     cursor = connection.cursor()
82
83     cursor.execute('''
84         DELETE FROM games WHERE id = ?
85     ''', (id,))
86
87     connection.commit()
88     connection.close()
89
90 def get_ordered_games(column, ascend=True, start_row=1, end_row=10):
91     """
92     Get specific number of rows from games table ordered by a specific column(s).
93
94     Args:
95         column (_type_): Column to sort by.
96         ascend (bool, optional): Sort ascending or descending. Defaults to True.
97         start_row (int, optional): First row returned. Defaults to 1.

```

```

97         end_row (int, optional): Last row returned. Defaults to 10.
98
99     Raises:
100         ValueError: If ascend argument or column argument are invalid types.
101
102     Returns:
103         list[dict]: List of ordered game entries represented as dictionaries.
104     """
105     if not isinstance(ascend, bool) or not isinstance(column, str):
106         raise ValueError('(database_helpers.get_ordered_games) Invalid input arguments!')
107
108     connection = sqlite3.connect(database_path, detect_types=sqlite3.PARSE_DECLTYPES)
109     connection.row_factory = sqlite3.Row
110     cursor = connection.cursor()
111
112     # Match ascend bool to correct SQL keyword
113     if ascend:
114         ascend_arg = 'ASC'
115     else:
116         ascend_arg = 'DESC'
117
118     # Partition by winner, then order by time and number_of_ply
119     if column == 'winner':
120         cursor.execute(f'''
121             SELECT * FROM
122                 (SELECT ROW_NUMBER() OVER (
123                     PARTITION BY winner
124                     ORDER BY time {ascend_arg}, number_of_ply {ascend_arg}
125                 ) AS row_num, * FROM games)
126             WHERE row_num >= ? AND row_num <= ?
127             ''', (start_row, end_row))
128     else:
129         # Order by time or number_of_ply only
130         cursor.execute(f'''
131             SELECT * FROM
132                 (SELECT ROW_NUMBER() OVER (
133                     ORDER BY {column} {ascend_arg}
134                 ) AS row_num, * FROM games)
135             WHERE row_num >= ? AND row_num <= ?
136             ''', (start_row, end_row))
137
138     games = cursor.fetchall()
139
140     connection.close()
141
142     return [dict(game) for game in games]
143
144 def get_number_of_games():
145     """
146     Returns:
147         int: Number of rows in the games.
148     """
149     connection = sqlite3.connect(database_path)
150     cursor = connection.cursor()
151
152     cursor.execute("""
153         SELECT COUNT(ROWID) FROM games
154     """)
155
156     result = cursor.fetchall()[0][0]

```

```

157         connection.close()
158     return result
159
160 # delete_all_games()
161
162
163 import json
164 from pathlib import Path
165
166 module_path = Path(__file__).parent
167 default_file_path = (module_path / '../app_data/default_settings.json').resolve()
168 user_file_path = (module_path / '../app_data/user_settings.json').resolve()
169 themes_file_path = (module_path / '../app_data/themes.json').resolve()
170
171 def load_json(path):
172     """
173     Args:
174         path (str): Path to JSON file.
175
176     Raises:
177         Exception: Invalid file.
178
179     Returns:
180         dict: Parsed JSON file.
181     """
182     try:
183         with open(path, 'r') as f:
184             file = json.load(f)
185
186         return file
187     except:
188         raise Exception('Invalid JSON file (data_helpers.py)')
189
190 def get_user_settings():
191     return load_json(user_file_path)
192
193 def get_default_settings():
194     return load_json(default_file_path)
195
196 def get_themes():
197     return load_json(themes_file_path)
198
199 def update_user_settings(data):
200     """
201     Rewrites JSON file for user settings with new data.
202
203     Args:
204         data (dict): Dictionary storing updated user settings.
205
206     Raises:
207         Exception: Invalid file.
208     """
209     try:
210         with open(user_file_path, 'w') as f:
211             json.dump(data, f, indent=4)
212     except:
213         raise Exception('Invalid JSON file (data_helpers.py)')
214
215 def height_to_font_size(font, target_height):
216     test_size = 1
217     while True:

```



```

5         raise ValueError('Invalid move type - move type must be a string!')
6     if move_type.lower() not in MoveType:
7         raise ValueError('Invalid move - type - move type must be m or r!')
8
9     return MoveType(move_type.lower())
10
11 def parse_notation(notation):
12     if (notation[0].isalpha() is False) or (notation[1].isnumeric() is False):
13         raise ValueError('Invalid notation - invalid notation input types!')
14     if not (97 <= ord(notation[0]) <= 106):
15         raise ValueError('Invalid notation - file is out of range!')
16     elif not (0 <= int(notation[1]) <= 10):
17         raise ValueError('Invalid notation - rank is out of range!')
18
19     return notation
20
21 def parse_rotation(rotation):
22     if rotation == '':
23         return None
24     if rotation.isalpha() is False:
25         raise ValueError('Invalid rotation - rotation must be a string!')
26     if rotation.lower() not in Rotation:
27         raise ValueError('Invalid rotation - rotation is invalid!')
28
29     return Rotation(rotation.lower())
30
31 import pygame
32 from pathlib import Path
33
34 import pygame.freetype
35 from data.utils.asset_helpers import gif_to_frames, pil_image_to_surface
36
37 def convert_gfx_alpha(image, colorkey=(0, 0, 0)):
38     # if image.get_alpha():
39     #     return image.convert_alpha()
40     # else:
41     #     image = image.convert_alpha()
42     #     image.set_colorkey(colorkey)
43
44     #     return image
45
46 def load_gfx(path, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".gif")):
47     file_path = Path(path)
48     name, extension = file_path.stem, file_path.suffix
49
50     if extension.lower() in accept:
51         if extension.lower() == '.gif':
52             frames_list = []
53
54             for frame in gif_to_frames(path):
55                 image_surface = pil_image_to_surface(frame)
56                 frames_list.append(image_surface)
57
58             return frames_list
59
60         if extension.lower() == '.svg':
61             low_quality_image = pygame.image.load_sized_svg(path, (200, 200))
62             image = pygame.image.load(path)
63             image = convert_gfx_alpha(image, colorkey)
64
65             return [image, low_quality_image]

```



```

37         else:
38             image = pygame.image.load(path)
39             return convert_gfx_alpha(image, colorkey)
40
41 def load_all_gfx(directory, colorkey=(0, 0, 0), accept=(".svg", ".png", ".jpg", ".
gif")):
42     graphics = {}
43
44     for file in Path(directory).rglob('*'):
45         name, extension = file.stem, file.suffix
46         path = Path(directory / file)
47
48         if extension.lower() in accept and 'old' not in name:
49             if name == 'piece_spritesheet':
50                 data = load_spritesheet(
51                     path,
52                     (16, 16),
53                     ['pyramid_1', 'scarab_1', 'anubis_1', 'pharoah_1', 'sphinx_1',
'pyramid_0', 'scarab_0', 'anubis_0', 'pharoah_0', 'sphinx_0'],
54                     ['_a', '_b', '_c', '_d'])
55
56                 graphics = graphics | data
57                 continue
58
59                 data = load_gfx(path, colorkey, accept)
60
61                 if isinstance(data, list):
62                     graphics[name] = data[0]
63                     graphics[f'{name}_lq'] = data[1]
64                 else:
65                     graphics[name] = data
66
67     return graphics
68
69 def load_spritesheet(path, sprite_size, col_names, row_names):
70     spritesheet = load_gfx(path)
71     col_count = int(spritesheet.width / sprite_size[0])
72     row_count = int(spritesheet.height / sprite_size[1])
73
74     sprite_dict = {}
75
76     for column in range(col_count):
77         for row in range(row_count):
78             surface = pygame.Surface(sprite_size, pygame.SRCALPHA)
79             name = col_names[column] + row_names[row]
80
81             surface.blit(spritesheet, (0, 0), (column * sprite_size[0], row *
sprite_size[1], *sprite_size))
82             sprite_dict[name] = surface
83
84     return sprite_dict
85
86 def load_all_fonts(directory, accept=(".ttf", ".otf")):
87     fonts = {}
88
89     for file in Path(directory).rglob('*'):
90         name, extension = file.stem, file.suffix
91         path = Path(directory / file)
92
93         if extension.lower() in accept:
94             font = pygame.freetype.Font(path)
95             fonts[name] = font

```

```

96
97     return fonts
98
99 def load_all_sfx(directory, accept=(".mp3", ".wav", ".ogg")):
100     sound_effects = {}
101
102     for file in Path(directory).rglob('*'):
103         name, extension = file.stem, file.suffix
104         path = Path(directory / file)
105
106         if extension.lower() in accept and 'old' not in name:
107             sound_effects[name] = load_sfx(path)
108
109     return sound_effects
110
111 def load_sfx(path, accept=(".mp3", ".wav", ".ogg")):
112     file_path = Path(path)
113     name, extension = file_path.stem, file_path.suffix
114
115     if extension.lower() in accept:
116         sfx = pygame.mixer.Sound(path)
117         return sfx
118
119 def load_all_music(directory, accept=(".mp3", ".wav", ".ogg")):
120     music_paths = {}
121     for file in Path(directory).rglob('*'):
122         name, extension = file.stem, file.suffix
123         path = Path(directory / file)
124
125         if extension.lower() in accept:
126             music_paths[name] = path
127
128     return music_paths

```

  

```

1 import pygame
2 from math import sqrt
3
4 def create_slider(size, fill_colour, border_width, border_colour):
5     """
6     Creates surface for sliders.
7
8     Args:
9         size (list[int, int]): Image size.
10        fill_colour (pygame.Color): Fill (inner) colour.
11        border_width (float): Border width.
12        border_colour (pygame.Color): Border colour.
13
14    Returns:
15        pygame.Surface: Slider image surface.
16    """
17    gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
18    border_rect = pygame.Rect((0, 0, gradient_surface.width, gradient_surface.
19    height))
20
21    # Draws rectangle with a border radius half of image height, to draw an
22    # rectangle with semicircular cap (obround)
23    pygame.draw.rect(gradient_surface, fill_colour, border_rect, border_radius=int
24    (size[1] / 2))
25    pygame.draw.rect(gradient_surface, border_colour, border_rect, width=int(
26    border_width), border_radius=int(size[1] / 2))
27
28    return gradient_surface

```

```

25
26 def create_slider_gradient(size, border_width, border_colour):
27     """
28     Draws surface for colour slider, with a full colour gradient as fill colour.
29
30     Args:
31         size (list[int, int]): Image size.
32         border_width (float): Border width.
33         border_colour (pygame.Color): Border colour.
34
35     Returns:
36         pygame.Surface: Slider image surface.
37     """
38     gradient_surface = pygame.Surface(size, pygame.SRCALPHA)
39
40     first_round_end = gradient_surface.height / 2
41     second_round_end = gradient_surface.width - first_round_end
42     gradient_y_mid = gradient_surface.height / 2
43
44     # Iterate through length of slider
45     for i in range(gradient_surface.width):
46         draw_height = gradient_surface.height
47
48         if i < first_round_end or i > second_round_end:
49             # Draw semicircular caps if x-distance less than or greater than
49             radius of cap (half of image height)
50             distance_from_cutoff = min(abs(first_round_end - i), abs(i -
51             second_round_end))
52             draw_height = calculate_gradient_slice_height(distance_from_cutoff,
53             gradient_surface.height / 2)
54
55             # Get colour from distance from left side of slider
56             color = pygame.Color(0)
57             color.hsva = (int(360 * i / gradient_surface.width), 100, 100, 100)
58
59             draw_rect = pygame.FRect((0, 0, 1, draw_height - 2 * border_width))
60             draw_rect.center = (i, gradient_y_mid)
61
62             pygame.draw.rect(gradient_surface, color, draw_rect)
63
64     border_rect = pygame.FRect((0, 0, gradient_surface.width, gradient_surface.
65     height))
66     pygame.draw.rect(gradient_surface, border_colour, border_rect, width=int(
67     border_width), border_radius=int(size[1] / 2))
68
69     return gradient_surface
70
71 def calculate_gradient_slice_height(distance, radius):
72     """
73     Calculate height of vertical slice of semicircular slider cap.
74
75     Args:
76         distance (float): x-distance from center of circle.
77         radius (float): Radius of semicircle.
78
79     Returns:
80         float: Height of vertical slice.
81     """
82     return sqrt(radius ** 2 - distance ** 2) * 2 + 2
83
84 def create_slider_thumb(radius, colour, border_colour, border_width):
85     """

```

```

82     Creates surface with bordered circle.
83
84     Args:
85         radius (float): Radius of circle.
86         colour (pygame.Color): Fill colour.
87         border_colour (pygame.Color): Border colour.
88         border_width (float): Border width.
89
90     Returns:
91         pygame.Surface: Circle surface.
92     """
93     thumb_surface = pygame.Surface((radius * 2, radius * 2), pygame.SRCALPHA)
94     pygame.draw.circle(thumb_surface, border_colour, (radius, radius), radius,
95                       width=int(border_width))
96     pygame.draw.circle(thumb_surface, colour, (radius, radius), (radius -
97                       border_width))
98
99     return thumb_surface
100
101 def create_square_gradient(side_length, colour):
102     """
103     Creates a square gradient for the colour picker widget, gradient transitioning
104     between saturation and value.
105     Uses smoothscale to blend between colour values for individual pixels.
106
107     Args:
108         side_length (float): Length of a square side.
109         colour (pygame.Color): Colour with desired hue value.
110
111     Returns:
112         pygame.Surface: Square gradient surface.
113     """
114     square_surface = pygame.Surface((side_length, side_length))
115
116     mix_1 = pygame.Surface((1, 2))
117     mix_1.fill((255, 255, 255))
118     mix_1.set_at((0, 1), (0, 0, 0))
119     mix_1 = pygame.transform.smoothscale(mix_1, (side_length, side_length))
120
121     hue = colour.hsva[0]
122     saturated_rgb = pygame.Color(0)
123     saturated_rgb.hsva = (hue, 100, 100)
124
125     mix_2 = pygame.Surface((2, 1))
126     mix_2.fill((255, 255, 255))
127     mix_2.set_at((1, 0), saturated_rgb)
128     mix_2 = pygame.transform.smoothscale(mix_2, (side_length, side_length))
129
130     mix_1.blit(mix_2, (0, 0), special_flags=pygame.BLEND_MULT)
131
132     square_surface.blit(mix_1, (0, 0))
133
134     return square_surface
135
136 def create_switch(size, colour):
137     """
138     Creates surface for switch toggle widget.
139
140     Args:
141         size (list[int, int]): Image size.
142         colour (pygame.Color): Fill colour.

```

```

141     Returns:
142         pygame.Surface: Switch surface.
143     """
144     switch_surface = pygame.Surface((size[0], size[1]), pygame.SRCALPHA)
145     pygame.draw.rect(switch_surface, colour, (0, 0, size[0], size[1]),
146                     border_radius=int(size[1] / 2))
147
148     return switch_surface
149
150 def create_text_box(size, border_width, colours):
151     """
152     Creates bordered textbox with shadow, flat, and highlighted vertical regions.
153
154     Args:
155         size (list[int, int]): Image size.
156         border_width (float): Border width.
157         colours (list[pygame.Color, ...]): List of 4 colours, representing border
158         colour, shadow colour, flat colour and highlighted colour.
159
160     Returns:
161         pygame.Surface: Textbox surface.
162     """
163     surface = pygame.Surface(size, pygame.SRCALPHA)
164
165     pygame.draw.rect(surface, colours[0], (0, 0, *size))
166     pygame.draw.rect(surface, colours[2], (border_width, border_width, size[0] -
167     * border_width, size[1] - 2 * border_width))
168     pygame.draw.rect(surface, colours[3], (border_width, border_width, size[0] -
169     * border_width, border_width))
170     pygame.draw.rect(surface, colours[1], (border_width, size[1] - 2 *
171     border_width, size[0] - 2 * border_width, border_width))
172
173     return surface
174
175
176 1 import pygame
177 2 from data.widgets.bases.widget import _Widget
178 3 from data.widgets.chessboard import Chessboard
179 4 from data.states.game.components.piece_group import PieceGroup
180 5 from data.states.game.components.bitboard_collection import BitboardCollection
181
182 6 class BoardThumbnail(_Widget):
183 7     def __init__(self, relative_width, fen_string='', **kwargs):
184 8         super().__init__(relative_size=(relative_width, relative_width * 0.8), **
185 9         kwargs)
186
187 10
188 11         self._board = Chessboard(
189 12             parent=self._parent,
190 13             relative_position=(0, 0),
191 14             scale_mode=kwargs.get('scale_mode'),
192 15             relative_width=relative_width
193 16         )
194
195 17
196 18         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
197
198 19
199 20         self.initialise_board(fen_string)
200 21         self.set_image()
201 22         self.set_geometry()
202
203 23
204 24     def initialise_board(self, fen_string):
205 25         if len(fen_string) == 0:
206 26             piece_list = []
207 27         else:

```

```

28         piece_list = BitboardCollection(fen_string).convert_to_piece_list()
29
30         self._piece_group = PieceGroup()
31         self._piece_group.initialise_pieces(piece_list, (0, 0), self.size)
32
33         self._board.refresh_board()
34         self.set_image()
35
36     def set_image(self):
37         self.image = pygame.transform.scale(self._empty_surface, self.size)
38
39         self._board.set_image()
40         self.image.blit(self._board.image, (0, 0))
41
42         self._piece_group.draw(self.image)
43
44     def set_geometry(self):
45         super().set_geometry()
46         self._board.set_geometry()
47
48     def set_surface_size(self, new_surface_size):
49         super().set_surface_size(new_surface_size)
50         self._board.set_surface_size(new_surface_size)
51         self._piece_group.handle_resize((0, 0), self.size)
52
53     def process_event(self, event):
54         pass
55
56
57 1 import pygame
58 2 from data.widgets.bases.pressable import _Pressable
59 3 from data.widgets.board_thumbnail import BoardThumbnail
60 4 from data.constants import WidgetState
61 5 from data.components.custom_event import CustomEvent
62
63 6
64 7 class BoardThumbnailButton(_Pressable, BoardThumbnail):
65 8     def __init__(self, event, **kwargs):
66 9         _Pressable.__init__(
67 10             self,
68 11             event=CustomEvent(**vars(event), fen_string=kwargs.get('fen_string')),
69 12             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
70 13             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
71 14             up_func=lambda: self.set_state_colour(WidgetState.BASE),
72 15         )
73 16         BoardThumbnail.__init__(self, **kwargs)
74
75 17         self.initialise_new_colours(self._fill_colour)
76 18         self.set_state_colour(WidgetState.BASE)
77
78
79 1 import pygame
80 2 from data.utils.font_helpers import text_width_to_font_size
81 3 from data.utils.browser_helpers import get_winner_string
82 4 from data.widgets.board_thumbnail import BoardThumbnail
83 5 from data.utils.asset_helpers import scale_and_cache
84 6 from data.widgets.bases.widget import _Widget
85
86 7
87 8 FONT_DIVISION = 7
89
90 9
91 10 class BrowserItem(_Widget):
92 11     def __init__(self, relative_width, game, **kwargs):
93 12         super().__init__(relative_size=(relative_width, relative_width * 2),
94 13             scale_mode='height', **kwargs)

```

```

14         self._relative_font_size = text_width_to_font_size('YYYY-MM-DD HH:MM:SS',
15 self._font, self.size[0]) / self.surface_size[1]
16
17         self._game = game
18         self._board_thumbnail = BoardThumbnail(
19             relative_position=(0, 0),
20             scale_mode='height',
21             relative_width=relative_width,
22             fen_string=self._game['final_fen_string']
23         )
24
25         self.set_image()
26         self.set_geometry()
27
28     def get_text_to_render(self):
29         depth_to_text = {
30             2: 'EASY',
31             3: 'MEDIUM',
32             4: 'HARD'
33         }
34
35         format_moves = lambda no_of_moves: int(no_of_moves / 2) if (no_of_moves /
36 2 % 1 == 0) else round(no_of_moves / 2, 1)
37
38         if self._game['cpu_enabled'] == 1:
39             depth_text = depth_to_text[self._game['cpu_depth']]
40             cpu_text = f'PVC ({depth_text})'
41         else:
42             cpu_text = 'PVP'
43
44         return [
45             cpu_text,
46             self._game['created_dt'].strftime('%Y-%m-%d %H:%M:%S'),
47             f'WINNER: {get_winner_string(self._game['winner'])}',
48             f'NO. MOVES: {format_moves(self._game['number_of_ply'])}'
49         ]
50
51     def set_image(self):
52         self.image = pygame.Surface(self.size, pygame.SRCALPHA)
53         resized_board = scale_and_cache(self._board_thumbnail.image, (self.size
54 [0], self.size[0] * 0.8))
55         self.image.blit(resized_board, (0, 0))
56
57         get_line_y = lambda line: (self.size[0] * 0.8) + ((self.size[0] * 0.8) /
58 FONT_DIVISION) * (line + 0.5)
59
60         text_to_render = self.get_text_to_render()
61
62         for index, text in enumerate(text_to_render):
63             self._font.render_to(self.image, (0, get_line_y(index)), text, fgcolor
64 =self._text_colour, size=self.font_size)
65
66     def process_event(self, event):
67         pass
68
69 1 import pygame
70 2 from data.widgets.bases.widget import _Widget
71 3 from data.widgets.browser_item import BrowserItem
72 4 from data.constants import BrowserEventType
73 5 from data.components.custom_event import CustomEvent
74
75 6
76 7 WIDTH_FACTOR = 0.3

```

```

8
9 class BrowserStrip(_Widget):
10     def __init__(self, relative_height, games_list, **kwargs):
11         super().__init__(relative_size=None, **kwargs)
12         self._relative_item_width = relative_height / 2
13         self._get_rect = None
14
15         self._games_list = []
16         self._items_list = []
17         self._selected_index = None
18
19         self.initialise_games_list(games_list)
20
21     @property
22     def item_width(self):
23         return self._relative_item_width * self.surface_size[1]
24
25     @property
26     def size(self):
27         if self._get_rect:
28             height = self._get_rect().height
29         else:
30             height = 0
31         width = max(0, len(self._games_list) * (self.item_width + self.margin) +
32 self.margin)
33
34         return (width, height)
35
36     def register_get_rect(self, get_rect_func):
37         self._get_rect = get_rect_func
38
39     def initialise_games_list(self, games_list):
40         self._items_list = []
41         self._games_list = games_list
42         self._selected_index = None
43
44         for game in games_list:
45             browser_item = BrowserItem(relative_position=(0, 0), game=game,
46 relative_width=self._relative_item_width)
47             self._items_list.append(browser_item)
48
49             self.set_image()
50             self.set_geometry()
51
52     def set_image(self):
53         self.image = pygame.Surface(self.size, pygame.SRCALPHA)
54         browser_list = []
55
56         for index, item in enumerate(self._items_list):
57             item.set_image()
58             browser_list.append((item.image, (index * (self.item_width + self.
59 margin) + self.margin, self.margin)))
60
61         self.image.blit(browser_list)
62
63         if self._selected_index is not None:
64             border_position = (self._selected_index * (self.item_width + self.
65 margin), 0)
66             border_size = (self.item_width + 2 * self.margin, self.size[1])
67             pygame.draw.rect(self.image, (255, 255, 255), (*border_position, *
68 border_size), width=int(self.item_width / 20))
69

```



```

65     def set_geometry(self):
66         super().set_geometry()
67         for item in self._items_list:
68             item.set_geometry()
69
70     def set_surface_size(self, new_surface_size):
71         super().set_surface_size(new_surface_size)
72
73         for item in self._items_list:
74             item.set_surface_size(new_surface_size)
75
76     def process_event(self, event, scrolled_pos):
77         parent_pos = self._get_rect().topleft
78         self.rect.topleft = parent_pos
79
80         if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
81             self._selected_index = None
82             self.set_image()
83             return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
84                                selected_index=None)
85
86         if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.
87 pos):
88             relative_mouse_pos = (event.pos[0] - parent_pos[0], event.pos[1] -
89 parent_pos[1])
90             self._selected_index = int(max(0, (relative_mouse_pos[0] - self.margin
91 ) // (self.item_width + self.margin)))
92             self.set_image()
93             return CustomEvent(BrowserEventType.BROWSER_STRIP_CLICK,
94                                selected_index=self._selected_index)
95
96 1 import pygame
97 2 from data.widgets.reactive_icon_button import ReactiveIconButton
98 3 from data.components.custom_event import CustomEvent
99 4 from data.widgets.bases.circular import _Circular
100 5 from data.widgets.bases.widget import _Widget
101 6 from data.constants import Miscellaneous
102 7 from data.assets import GRAPHICS, SFX
103 8
104 9 class Carousel(_Circular, _Widget):
105 10     def __init__(self, event, widgets_dict, **kwargs):
106 11         _Circular.__init__(self, items_dict=widgets_dict)
107 12         _Widget.__init__(self, relative_size=None, **kwargs)
108 13
109 14         max_widget_size = (
110 15             max([widget.rect.width for widget in widgets_dict.values()]),
111 16             max([widget.rect.height for widget in widgets_dict.values()])
112 17         )
113 18
114 19         self._relative_max_widget_size = (max_widget_size[0] / self.surface_size
115 [1], max_widget_size[1] / self.surface_size[1])
116 20         self._relative_size = ((max_widget_size[0] + 2 * (self.margin + self.
117 arrow_size[0])) / self.surface_size[1], (max_widget_size[1] / self.
118 surface_size[1])
119 21
120 22         self._left_arrow = ReactiveIconButton(
121 23             relative_position=(0, 0),
122 24             relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
123 25             scale_mode='height',
124 26             base_icon=GRAPHICS['left_arrow_base'],
125 27             hover_icon=GRAPHICS['left_arrow_hover'],
126 28             press_icon=GRAPHICS['left_arrow_press'],

```

```

29         event=CustomEvent(Miscellaneous.PLACEHOLDER),
30         sfx=SFX['carousel_click']
31     )
32     self._right_arrow = ReactiveIconButton(
33         relative_position=(0, 0),
34         relative_size=(0, self.arrow_size[1] / self.surface_size[1]),
35         scale_mode='height',
36         base_icon=GRAPHICS['right_arrow_base'],
37         hover_icon=GRAPHICS['right_arrow_hover'],
38         press_icon=GRAPHICS['right_arrow_press'],
39         event=CustomEvent(Miscellaneous.PLACEHOLDER),
40         sfx=SFX['carousel_click']
41     )
42
43     self._event = event
44     self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
45
46     self.set_image()
47     self.set_geometry()
48
49     @property
50     def max_widget_size(self):
51         return (self._relative_max_widget_size[0] * self.surface_size[1], self.
52             _relative_max_widget_size[1] * self.surface_size[1])
53
54     @property
55     def arrow_size(self):
56         height = self.max_widget_size[1] * 0.75
57         width = (GRAPHICS['left_arrow_base'].width / GRAPHICS['left_arrow_base'].
58             height) * height
59         return (width, height)
60
61     @property
62     def size(self):
63         return ((self.arrow_size[0] + self.margin) * 2 + self.max_widget_size[0],
64             self.max_widget_size[1])
65
66     @property
67     def left_arrow_position(self):
68         return (0, (self.size[1] - self.arrow_size[1]) / 2)
69
70     @property
71     def right_arrow_position(self):
72         return (self.size[0] - self.arrow_size[0], (self.size[1] - self.arrow_size
73             [1]) / 2)
74
75     def set_image(self):
76         self.image = pygame.transform.scale(self._empty_surface, self.size)
77         self.image.fill(self._fill_colour)
78
79         if self.border_width:
80             pygame.draw.rect(self.image, self._border_colour, (0, 0, *self.size),
81                 width=int(self.border_width), border_radius=int(self.border_radius))
82
83         self._left_arrow.set_image()
84         self.image.blit(self._left_arrow.image, self.left_arrow_position)
85
86         self.current_item.set_image()
87         self.image.blit(self.current_item.image, ((self.size[0] - self.
88             current_item.rect.size[0]) / 2, (self.size[1] - self.current_item.rect.size
89             [1]) / 2))

```

```

84         self._right_arrow.set_image()
85         self.image.blit(self._right_arrow.image, self.right_arrow_position)
86
87     def set_geometry(self):
88         super().set_geometry()
89
90         self.current_item.set_geometry()
91         self._left_arrow.set_geometry()
92         self._right_arrow.set_geometry()
93
94         self.current_item.rect.center = self.rect.center
95         self._left_arrow.rect.topleft = (self.position[0] + self.
left_arrow_position[0], self.position[1] + self.left_arrow_position[1])
96         self._right_arrow.rect.topleft = (self.position[0] + self.
right_arrow_position[0], self.position[1] + self.right_arrow_position[1])
97
98     def set_surface_size(self, new_surface_size):
99         super().set_surface_size(new_surface_size)
100         self._left_arrow.set_surface_size(new_surface_size)
101         self._right_arrow.set_surface_size(new_surface_size)
102
103         for item in self._items_dict.values():
104             item.set_surface_size(new_surface_size)
105
106     def process_event(self, event):
107         self.current_item.process_event(event)
108         left_arrow_event = self._left_arrow.process_event(event)
109         right_arrow_event = self._right_arrow.process_event(event)
110
111         if left_arrow_event:
112             self.set_previous_item()
113             self.current_item.set_surface_size(self._raw_surface_size)
114
115         elif right_arrow_event:
116             self.set_next_item()
117             self.current_item.set_surface_size(self._raw_surface_size)
118
119         if left_arrow_event or right_arrow_event:
120             self.set_image()
121             self.set_geometry()
122
123         return CustomEvent(**vars(self._event), data=self.current_key)
124
125         elif event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
MOUSEMOTION]:
126             self.set_image()
127             self.set_geometry()
128
129
130 1 import pygame
131 2 from data.widgets.bases.widget import _Widget
132 3 from data.utils.board_helpers import create_board
133 4 from data.utils.data_helpers import get_user_settings
134 5 from data.constants import CursorMode
135 6 from data.managers.cursor import cursor
136 7
137 8 class Chessboard(_Widget):
138 9     def __init__(self, relative_width, change_cursor=True, **kwargs):
139 10         super().__init__(relative_size=(relative_width, relative_width * 0.8), **
kwargs)
140 11
141 12         self._board_surface = None
142 13         self._change_cursor = change_cursor

```

```

14         self._cursor_is_hand = False
15
16         self.refresh_board()
17         self.set_image()
18         self.set_geometry()
19
20     def refresh_board(self):
21         user_settings = get_user_settings()
22         self._board_surface = create_board(self.size, user_settings['
primaryBoardColour'], user_settings['secondaryBoardColour'])
23
24         self.set_image()
25
26     def set_image(self):
27         self.image = pygame.transform.smoothscale(self._board_surface, self.size)
28
29     def process_event(self, event):
30         if self._change_cursor and event.type in [pygame.MOUSEMOTION, pygame.
MOUSEBUTTONUP, pygame.MOUSEBUTTONDOWN]:
31             current_cursor = cursor.get_mode()
32
33             if self.rect.collidepoint(event.pos):
34                 if current_cursor == CursorMode.ARROW:
35                     cursor.set_mode(CursorMode.OPENHAND)
36                 elif current_cursor == CursorMode.OPENHAND and (pygame.mouse.
get_pressed()[0] is True or event.type == pygame.MOUSEBUTTONDOWN):
37                     cursor.set_mode(CursorMode.CLOSEDHAND)
38                 elif current_cursor == CursorMode.CLOSEDHAND and (pygame.mouse.
get_pressed()[0] is False or event.type == pygame.MOUSEBUTTONUP):
39                     cursor.set_mode(CursorMode.OPENHAND)
40             else:
41                 if current_cursor == CursorMode.OPENHAND or (current_cursor ==
CursorMode.CLOSEDHAND and event.type == pygame.MOUSEBUTTONUP):
42                     cursor.set_mode(CursorMode.ARROW)
43
44
45 1 import pygame
46 2 from data.widgets.bases.widget import _Widget
47 3 from data.widgets.bases.pressable import _Pressable
48 4 from data.constants import WidgetState
49
50 5
51 6 class ColourButton(_Pressable, _Widget):
52 7     def __init__(self, event, **kwargs):
53 8         _Pressable.__init__(
54 9             self,
55 10             event=event,
56 11             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
57 12             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
58 13             up_func=lambda: self.set_state_colour(WidgetState.BASE),
59 14             sfx=None
60 15         )
61 16         _Widget.__init__(self, **kwargs)
62
63 17
64 18         self._empty_surface = pygame.Surface(self.size)
65
66 19
67 20         self.initialise_new_colours(self._fill_colour)
68 21         self.set_state_colour(WidgetState.BASE)
69
70 22
71 23         self.set_image()
72 24         self.set_geometry()
73
74 25
75 26     def set_image(self):
76 27         self.image = pygame.transform.scale(self._empty_surface, self.size)

```

```

28         self.image.fill(self._fill_colour)
29         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
3
4 class _ColourDisplay(_Widget):
5     def __init__(self, **kwargs):
6         super().__init__(**kwargs)
7
8         self._colour = None
9
10        self._empty_surface = pygame.Surface(self.size)
11
12        def set_colour(self, new_colour):
13            self._colour = new_colour
14
15        def set_image(self):
16            self.image = pygame.transform.scale(self._empty_surface, self.size)
17            self.image.fill(self._colour)
18
19        def process_event(self, event):
20            pass

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
7
8 class ColourPicker(_Widget):
9     def __init__(self, relative_width, event_type, **kwargs):
10        super().__init__(relative_size=(relative_width, relative_width),
scale_mode='width', **kwargs)
11
12        self.image = pygame.Surface(self.size)
13        self.rect = self.image.get_rect()
14
15        self._square = _ColourSquare(
16            parent=self,
17            relative_position=(0.1, 0.1),
18            relative_width=0.5,
19            event_type=event_type
20        )
21        self._square.set_colour(kwargs.get('selected_colour'))
22
23        self._slider = _ColourSlider(
24            parent=self,
25            relative_position=(0.0, 0.7),
26            relative_width=1.0,
27            border_width=self.border_width,
28            border_colour=self._border_colour
29        )
30        self._slider.set_colour(kwargs.get('selected_colour'))
31
32        self._display = _ColourDisplay(
33            parent=self,
34            relative_position=(0.7, 0.1),
35            relative_size=(0.2, 0.5)
36        )

```

```

37         self._display.set_colour(kwargs.get('selected_colour'))
38
39         self._event_type = event_type
40         self._hover_event_type = event_type
41
42         self.set_image()
43         self.set_geometry()
44
45     def global_to_relative_pos(self, global_pos):
46         return (global_pos[0] - self.position[0], global_pos[1] - self.position
47                [1])
48
49     def set_image(self):
50         self.image = pygame.Surface(self.size)
51         self.image.fill(self._fill_colour)
52
53         self._square.set_image()
54         self._square.set_geometry()
55         self.image.blit(self._square.image, self.global_to_relative_pos(self.
56                                _square.position))
57
58         self._slider.set_image()
59         self._slider.set_geometry()
60         self.image.blit(self._slider.image, self.global_to_relative_pos(self.
61                                _slider.position))
62
63         self._display.set_image()
64         self._display.set_geometry()
65         self.image.blit(self._display.image, self.global_to_relative_pos(self.
66                                _display.position))
67
68         pygame.draw.rect(self.image, self._border_colour, (0, 0, self.size[0],
69                                self.size[1]), width=int(self.border_width))
70
71     def set_surface_size(self, new_surface_size):
72         super().set_surface_size(new_surface_size)
73         self._square.set_surface_size(self.size)
74         self._slider.set_surface_size(self.size)
75         self._display.set_surface_size(self.size)
76
77     def get_picker_position(self):
78         return self.position
79
80     def process_event(self, event):
81         slider_colour = self._slider.process_event(event)
82         square_colour = self._square.process_event(event)
83
84         if square_colour:
85             self._display.set_colour(square_colour)
86             self.set_image()
87
88         if slider_colour:
89             self._square.set_colour(slider_colour)
90             self.set_image()
91
92         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.
93                                MOUSEMOTION] and self.rect.collidepoint(event.pos):
94             return CustomEvent(self._event_type, colour=square_colour)
95
96 import pygame
97 from data.utils.widget_helpers import create_slider_gradient
98 from data.utils.asset_helpers import smoothscale_and_cache

```

```

4 from data.widgets.slider_thumb import _SliderThumb
5 from data.widgets.bases.widget import _Widget
6 from data.constants import WidgetState
7
8 class _ColourSlider(_Widget):
9     def __init__(self, relative_width, **kwargs):
10         super().__init__(relative_size=(relative_width, relative_width * 0.2), **
11             kwargs)
12
13         # Initialise slider thumb.
14         self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
15             _border_colour)
16
17         self._selected_percent = 0
18         self._last_mouse_x = None
19
20         self._gradient_surface = create_slider_gradient(self.gradient_size, self.
21             border_width, self._border_colour)
22         self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
23
24 @property
25 def gradient_size(self):
26     return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
27
28 @property
29 def gradient_position(self):
30     return (self.size[1] / 2, self.size[1] / 4)
31
32 @property
33 def thumb_position(self):
34     return (self.gradient_size[0] * self._selected_percent, 0)
35
36 @property
37 def selected_colour(self):
38     colour = pygame.Color(0)
39     colour.hsva = (int(self._selected_percent * 360), 100, 100, 100)
40     return colour
41
42 def calculate_gradient_percent(self, mouse_pos):
43     """
44     Calculate what percentage slider thumb is at based on change in mouse
45     position.
46
47     Args:
48         mouse_pos (list[int, int]): Position of mouse on window screen.
49
50     Returns:
51         float: Slider scroll percentage.
52     """
53     if self._last_mouse_x is None:
54         return
55
56     x_change = (mouse_pos[0] - self._last_mouse_x) / (self.gradient_size[0] -
57         2 * self.border_width)
58     return max(0, min(self._selected_percent + x_change, 1))
59
60 def relative_to_global_position(self, position):
61     """
62     Transforms position from being relative to widget rect, to window screen.
63
64     Args:
65         position (list[int, int]): Position relative to widget rect.

```

```

61
62     Returns:
63         list[int, int]: Position relative to window screen.
64     """
65     relative_x, relative_y = position
66     return (relative_x + self.position[0], relative_y + self.position[1])
67
68 def set_colour(self, new_colour):
69     """
70     Sets selected_percent based on the new colour's hue.
71
72     Args:
73         new_colour (pygame.Color): New slider colour.
74     """
75     colour = pygame.Color(new_colour)
76     hue = colour.hsva[0]
77     self._selected_percent = hue / 360
78     self.set_image()
79
80 def set_image(self):
81     """
82     Draws colour slider to widget image.
83     """
84     # Scales initialised gradient surface instead of redrawing it everytime
85     # set_image is called
86     gradient_scaled = smoothscale_and_cache(self._gradient_surface, self.
87     gradient_size)
88
89     self.image = pygame.transform.scale(self._empty_surface, (self.size))
90     self.image.blit(gradient_scaled, self.gradient_position)
91
92     # Resets thumb colour, image and position, then draws it to the widget
93     # image
94     self._thumb.initialise_new_colours(self.selected_colour)
95     self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
96     border_width)
97     self._thumb.set_position(self.relative_to_global_position((self.
98     thumb_position[0], self.thumb_position[1])))
99
100     thumb_surface = self._thumb.get_surface()
101     self.image.blit(thumb_surface, self.thumb_position)
102
103 def process_event(self, event):
104     """
105     Processes Pygame events.
106
107     Args:
108         event (pygame.Event): Event to process.
109
110     Returns:
111         pygame.Color: Current colour slider is displaying.
112     """
113     if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
114     MOUSEBUTTONUP]:
115         return
116
117     # Gets widget state before and after event is processed by slider thumb
118     before_state = self._thumb.state
119     self._thumb.process_event(event)
120     after_state = self._thumb.state
121
122     # If widget state changes (e.g. hovered -> pressed), redraw widget

```



```

117         if before_state != after_state:
118             self.set_image()
119
120         if event.type == pygame.MOUSEMOTION:
121             if self._thumb.state == WidgetState.PRESS:
122                 # Recalculates slider colour based on mouse position change
123                 selected_percent = self.calculate_gradient_percent(event.pos)
124                 self._last_mouse_x = event.pos[0]
125
126                 if selected_percent is not None:
127                     self._selected_percent = selected_percent
128
129                 return self.selected_colour
130
131         if event.type == pygame.MOUSEBUTTONUP:
132             # When user stops scrolling, return new slider colour
133             self._last_mouse_x = None
134             return self.selected_colour
135
136         if event.type == pygame.MOUSEBUTTONDOWN or before_state != after_state:
137             # Redraws widget when slider thumb is hovered or pressed
138             return self.selected_colour

```

```

1  import pygame
2  from data.widgets.bases.widget import _Widget
3  from data.utils.widget_helpers import create_square_gradient
4
5  class _ColourSquare(_Widget):
6      def __init__(self, relative_width, **kwargs):
7          super().__init__(relative_size=(relative_width, relative_width),
8                           scale_mode='width', **kwargs)
9
10         self._colour = None
11
12     def set_colour(self, new_colour):
13         self._colour = pygame.Color(new_colour)
14
15     def get_colour(self):
16         return self._colour
17
18     def set_image(self):
19         self.image = create_square_gradient(side_length=self.size[0], colour=self._colour)
20
21     def process_event(self, event):
22         if event.type == pygame.MOUSEBUTTONDOWN:
23             relative_mouse_pos = (event.pos[0] - self.position[0], event.pos[1] -
24                                   self.position[1])
25
26             if (
27                 0 > relative_mouse_pos[0] or
28                 self.size[0] < relative_mouse_pos[0] or
29                 0 > relative_mouse_pos[1] or
30                 self.size[1] < relative_mouse_pos[1]
31             ): return None
32
33             self.set_colour(self.image.get_at(relative_mouse_pos))
34
35             return self._colour
36
37     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
5 from data.utils.data_helpers import 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
8
9 user_settings = get_user_settings()
10
11 class Dropdown(_Pressable, _Widget):
12     def __init__(self, word_list, event=None, **kwargs):
13         _Pressable.__init__(
14             self,
15             event=event,
16             hover_func=self.hover_func,
17             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
18             up_func=self.up_func,
19             sfx=None
20         )
21         _Widget.__init__(self, relative_size=None, **kwargs)
22
23         if kwargs.get('relative_width'):
24             self._relative_font_size = text_width_to_font_size(max(word_list, key=
len), self._font, kwargs.get('relative_width') * self.surface_size[0] - self.
margin) / self.surface_size[1]
25         elif kwargs.get('relative_height'):
26             self._relative_font_size = text_height_to_font_size(max(word_list, key
=len), self._font, kwargs.get('relative_height') * self.surface_size[1] - self
.margin) / self.surface_size[1]
27
28         self._word_list = [word_list[0].capitalize()]
29         self._word_list_copy = [word.capitalize() for word in word_list]
30
31         self._expanded = False
32         self._hovered_index = None
33
34         self._empty_surface = pygame.Surface((0, 0))
35         self._background_colour = self._fill_colour
36
37         self.initialise_new_colours(self._fill_colour)
38         self.set_state_colour(WidgetState.BASE)
39
40         self.set_image()
41         self.set_geometry()
42
43     @property
44     def size(self):
45         max_word = sorted(self._word_list_copy, key=len)[-1]
46         max_word_rect = self._font.get_rect(max_word, size=self.font_size)
47         all_words_rect = pygame.FRect(0, 0, max_word_rect.size[0], (max_word_rect.
size[1] * len(self._word_list)) + (self.margin * (len(self._word_list) - 1)))
48         all_words_rect = all_words_rect.inflate(2 * self.margin, 2 * self.margin)
49         return (all_words_rect.size[0] + max_word_rect.size[1], all_words_rect.
size[1])
50
51     def get_selected_word(self):
52         return self._word_list[0].lower()
53
54     def toggle_expanded(self):
55         if self._expanded:

```

```

56         self._word_list = [self._word_list_copy[0]]
57     else:
58         self._word_list = [*self._word_list_copy]
59
60     self._expanded = not(self._expanded)
61
62     def hover_func(self):
63         mouse_position = pygame.mouse.get_pos()
64         relative_position = (mouse_position[0] - self.position[0], mouse_position
[1] - self.position[1])
65         self._hovered_index = self.calculate_hovered_index(relative_position)
66         self.set_state_colour(WidgetState.HOVER)
67
68     def set_selected_word(self, word):
69         index = self._word_list_copy.index(word.capitalize())
70         selected_word = self._word_list_copy.pop(index)
71         self._word_list_copy.insert(0, selected_word)
72
73         if self._expanded:
74             self._word_list.pop(index)
75             self._word_list.insert(0, selected_word)
76         else:
77             self._word_list = [selected_word]
78
79         self.set_image()
80
81     def up_func(self):
82         if self.get_widget_state() == WidgetState.PRESS:
83             if self._expanded and self._hovered_index is not None:
84                 self.set_selected_word(self._word_list_copy[self._hovered_index])
85
86                 self.toggle_expanded()
87
88             self._hovered_index = None
89
90             self.set_state_colour(WidgetState.BASE)
91             self.set_geometry()
92
93     def calculate_hovered_index(self, mouse_pos):
94         return int(mouse_pos[1] // (self.size[1] / len(self._word_list)))
95
96     def set_image(self):
97         text_surface = pygame.transform.scale(self._empty_surface, self.size)
98         self.image = text_surface
99
100         fill_rect = pygame.Rect(0, 0, self.size[0], self.size[1])
101         pygame.draw.rect(self.image, self._background_colour, fill_rect)
102         pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int(
self.border_width))
103
104         word_box_height = (self.size[1] - (2 * self.margin) - ((len(self.
_word_list) - 1) * self.margin)) / len(self._word_list)
105
106         arrow_size = (GRAPHICS['dropdown_arrow_open'].width / GRAPHICS['
dropdown_arrow_open'].height * word_box_height, word_box_height)
107         open_arrow_surface = pygame.transform.scale(GRAPHICS['dropdown_arrow_open'
], arrow_size)
108         closed_arrow_surface = pygame.transform.scale(GRAPHICS['
dropdown_arrow_close'], arrow_size)
109         arrow_position = (self.size[0] - arrow_size[0] - self.margin, (
word_box_height) / 3)
110

```

```

111         if self._expanded:
112             self.image.blit(closed_arrow_surface, arrow_position)
113         else:
114             self.image.blit(open_arrow_surface, arrow_position)
115
116         for index, word in enumerate(self._word_list):
117             word_position = (self.margin, self.margin + (word_box_height + self.
margin) * index)
118             self._font.render_to(self.image, word_position, word, fgcolor=self.
_text_colour, size=self.font_size)
119
120         if self._hovered_index is not None:
121             overlay_surface = pygame.Surface((self.size[0], word_box_height + 2 *
self.margin), pygame.SRCALPHA)
122             overlay_surface.fill((*self._fill_colour.rgb, 128))
123             overlay_position = (0, (word_box_height + self.margin) * self.
_hovered_index)
124             self.image.blit(overlay_surface, overlay_position)

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.utils.widget_helpers import create_text_box
4
5 class Icon(_Widget):
6     def __init__(self, icon, stretch=False, is_mask=False, smooth=False, fit_icon=
False, box_colours=None, **kwargs):
7         super().__init__(**kwargs)
8
9         if fit_icon:
10             aspect_ratio = icon.width / icon.height
11             self._relative_size = (self._relative_size[1] * aspect_ratio, self.
_relative_size[1])
12
13         self._icon = icon
14         self._is_mask = is_mask
15         self._stretch = stretch
16         self._smooth = smooth
17         self._box_colours = box_colours
18
19         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
20
21         self.set_image()
22         self.set_geometry()
23
24     def set_icon(self, icon):
25         self._icon = icon
26         self.set_image()
27
28     def set_image(self):
29         if self._box_colours:
30             self.image = create_text_box(self.size, self.border_width, self.
_box_colours)
31         else:
32             self.image = pygame.transform.scale(self._empty_surface, self.size)
33
34         if self._fill_colour:
35             pygame.draw.rect(self.image, self._fill_colour, self.image.
get_rect(), border_radius=int(self.border_radius))
36
37         if self._stretch:
38             if self._smooth:
39                 scaled_icon = pygame.transform.smoothscale(self._icon, (self.size

```

```

[0] - (2 * self.margin), self.size[1] - (2 * self.margin)))
40         else:
41             scaled_icon = pygame.transform.scale(self._icon, (self.size[0] -
(2 * self.margin), self.size[1] - (2 * self.margin)))
42
43             icon_position = (self.margin, self.margin)
44         else:
45             max_height = self.size[1] - (2 * self.margin)
46             max_width = self.size[0] - (2 * self.margin)
47             scale_factor = min(max_width / self._icon.width, max_height / self.
_icon.height)
48
49             if self._smooth:
50                 scaled_icon = pygame.transform.smoothscale_by(self._icon, (
scale_factor, scale_factor))
51             else:
52                 scaled_icon = pygame.transform.scale_by(self._icon, (scale_factor,
scale_factor))
53                 icon_position = ((self.size[0] - scaled_icon.width) / 2, (self.size[1]
- scaled_icon.height) / 2)
54
55             if self._is_mask:
56                 self.image.blit(scaled_icon, icon_position, None, pygame.
BLEND_RGBA_MULT)
57             else:
58                 self.image.blit(scaled_icon, icon_position)
59
60             if self._box_colours is None and self.border_width:
61                 pygame.draw.rect(self.image, self._border_colour, self.image.get_rect
(), width=int(self.border_width), border_radius=int(self.border_radius))
62
63     def process_event(self, event):
64         pass

1 from data.widgets.bases.pressable import _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
5
6 class IconButton(_Box, _Pressable, Icon):
7     def __init__(self, event, box_colours=RED_BUTTON_COLOURS, **kwargs):
8         _Box.__init__(self, box_colours=box_colours)
9         _Pressable.__init__(
10             self,
11             event=event,
12             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
13             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
14             up_func=lambda: self.set_state_colour(WidgetState.BASE),
15         )
16         Icon.__init__(self, box_colours=box_colours[WidgetState.BASE], **kwargs)
17
18         self.initialise_new_colours(self._fill_colour)
19         self.set_state_colour(WidgetState.BASE)

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.utils.font_helpers import width_to_font_size
4
5 class MoveList(_Widget):
6     def __init__(self, relative_width, minimum_height=0, move_list=[], **kwargs):
7         super().__init__(relative_size=None, **kwargs)
8

```

```

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

```

```

65         self._font.render_to(self.image, move_number_position, text=str(
move_number), size=self.font_size, fgcolor=self._text_colour)
66
67     def process_event(self, event, scrolled_pos=None):
68         pass

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
7
8 class MultipleIconButton(_Circular, IconButton):
9     def __init__(self, icons_dict, **kwargs):
10         _Circular.__init__(self, items_dict=icons_dict)
11         IconButton.__init__(self, icon=self.current_item, **kwargs)
12
13         self._fill_colour_copy = self._fill_colour
14
15         self._locked = None
16
17     def set_locked(self, is_locked):
18         self._locked = is_locked
19         if self._locked:
20             r, g, b, a = pygame.Color(self._fill_colour_copy).rgba
21             if self._box_colours_dict == BLUE_BUTTON_COLOURS:
22                 _Box.__init__(self, box_colours=LOCKED_BLUE_BUTTON_COLOURS)
23             elif self._box_colours_dict == RED_BUTTON_COLOURS:
24                 _Box.__init__(self, box_colours=LOCKED_RED_BUTTON_COLOURS)
25             else:
26                 self.initialise_new_colours((max(r + 50, 0), max(g + 50, 0), max(b + 50,
0), a))
27         else:
28             if self._box_colours_dict == LOCKED_BLUE_BUTTON_COLOURS:
29                 _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
30             elif self._box_colours_dict == LOCKED_RED_BUTTON_COLOURS:
31                 _Box.__init__(self, box_colours=RED_BUTTON_COLOURS)
32             else:
33                 self.initialise_new_colours(self._fill_colour_copy)
34
35         if self.rect.collidepoint(pygame.mouse.get_pos()):
36             self.set_state_colour(WidgetState.HOVER)
37         else:
38             self.set_state_colour(WidgetState.BASE)
39
40     def set_next_icon(self):
41         super().set_next_item()
42         self._icon = self.current_item
43         self.set_image()
44
45     def process_event(self, event):
46         widget_event = super().process_event(event)
47
48         if widget_event:
49             return CustomEvent(**vars(widget_event), data=self.current_key)

1 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
7
8 class PieceDisplay(_Widget):
9     def __init__(self, active_colour, **kwargs):
10         super().__init__(**kwargs)
11
12         self._active_colour = active_colour
13         self._piece_list = []
14         self._piece_surface = None
15         self._box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour
16         == Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]
17
18         self.initialise_piece_surface()
19
20         self.set_image()
21         self.set_geometry()
22
23     def add_piece(self, piece):
24         self._piece_list.append(piece)
25         self._piece_list.sort(key=lambda piece: Score[piece.name])
26         self.initialise_piece_surface()
27
28     def remove_piece(self, piece):
29         self._piece_list.remove(piece)
30         self.initialise_piece_surface()
31
32     def reset_piece_list(self):
33         self._piece_list = []
34         self.initialise_piece_surface()
35
36     def initialise_piece_surface(self):
37         self._piece_surface = pygame.Surface((self.size[0] - 2 * self.margin, self
38         .size[1] - 2 * self.margin), pygame.SRCALPHA)
39
40         if (len(self._piece_list) == 0):
41             self.set_image()
42             return
43
44         piece_width = min(self.size[1] - 2 * self.margin, (self.size[0] - 2 * self
45         .margin) / len(self._piece_list))
46         piece_list = []
47
48         for index, piece in enumerate(self._piece_list):
49             piece_instance = PieceSprite(piece, self._active_colour,
50             get_flipped_colour(), Rotation.UP)
51             piece_instance.set_geometry((0, 0), piece_width)
52             piece_instance.set_image()
53             piece_list.append((piece_instance.image, (piece_width * index, (self.
54             _piece_surface.height - piece_width) / 2)))
55
56         self._piece_surface.fblits(piece_list)
57
58         self.set_image()
59
60     def set_image(self):
61         self.image = create_text_box(self.size, self.border_width, self.
62         _box_colours)
63
64         resized_piece_surface = scale_and_cache(self._piece_surface, (self.size[0]

```



```

        - 2 * self.margin, self.size[1] - 2 * self.margin))
        self.image.blit(resized_piece_surface, (self.margin, self.margin))
59
60
61     def process_event(self, event):
62         pass

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
6
7  class ReactiveButton(_Pressable, _Circular, _Widget):
8      def __init__(self, widgets_dict, event, center=False, **kwargs):
9          # Multiple inheritance used here, to combine the functionality of multiple
          super classes
10         _Pressable.__init__(
11             self,
12             event=event,
13             hover_func=lambda: self.set_to_key(WidgetState.HOVER),
14             down_func=lambda: self.set_to_key(WidgetState.PRESS),
15             up_func=lambda: self.set_to_key(WidgetState.BASE),
16             **kwargs
17         )
18         # Aggregation used to cycle between external widgets
19         _Circular.__init__(self, items_dict=widgets_dict)
20         _Widget.__init__(self, **kwargs)
21
22         self._center = center
23
24         self.initialise_new_colours(self._fill_colour)
25
26     @property
27     def position(self):
28         """
29         Overrides position getter method, to always position icon in the center if
30         self._center is True.
31
32         Returns:
33             list[int, int]: Position of widget.
34         """
35         position = super().position
36
37         if self._center:
38             self._size_diff = (self.size[0] - self.rect.width, self.size[1] - self
39             .rect.height)
40             return (position[0] + self._size_diff[0] / 2, position[1] + self.
41             _size_diff[1] / 2)
42         else:
43             return position
44
45     def set_image(self):
46         """
47         Sets current icon to image.
48         """
49         self.current_item.set_image()
50         self.image = self.current_item.image
51
52     def set_geometry(self):
53         """
54         Sets size and position of widget.
55         """

```

```

53         super().set_geometry()
54         self.current_item.set_geometry()
55         self.current_item.rect.topleft = self.rect.topleft
56
57     def set_surface_size(self, new_surface_size):
58         """
59         Overrides base method to resize every widget state icon, not just the
60         current one.
61
62         Args:
63             new_surface_size (list[int, int]): New surface size.
64         """
65         super().set_surface_size(new_surface_size)
66         for item in self._items_dict.values():
67             item.set_surface_size(new_surface_size)
68
69     def process_event(self, event):
70         """
71         Processes Pygame events.
72
73         Args:
74             event (pygame.Event): Event to process.
75
76         Returns:
77             CustomEvent: CustomEvent of current item, with current key included
78         """
79         widget_event = super().process_event(event)
80         self.current_item.process_event(event)
81
82         if widget_event:
83             return CustomEvent(**vars(widget_event), data=self.current_key)
84
85 from data.widgets.reactive_button import ReactiveButton
86 from data.constants import WidgetState
87 from data.widgets.icon import Icon
88
89 class ReactiveIconButton(ReactiveButton):
90     def __init__(self, base_icon, hover_icon, press_icon, **kwargs):
91         # Composition is used here, to initialise the Icon widgets for each widget
92         # state
93         widgets_dict = {
94             WidgetState.BASE: Icon(
95                 parent=kwargs.get('parent'),
96                 relative_size=kwargs.get('relative_size'),
97                 relative_position=(0, 0),
98                 icon=base_icon,
99                 fill_colour=(0, 0, 0, 0),
100                 border_width=0,
101                 margin=0,
102                 fit_icon=True,
103             ),
104             WidgetState.HOVER: Icon(
105                 parent=kwargs.get('parent'),
106                 relative_size=kwargs.get('relative_size'),
107                 relative_position=(0, 0),
108                 icon=hover_icon,
109                 fill_colour=(0, 0, 0, 0),
110                 border_width=0,
111                 margin=0,
112                 fit_icon=True,
113             ),
114             WidgetState.PRESS: Icon(

```

```

30         parent=kwargs.get('parent'),
31         relative_size=kwargs.get('relative_size'),
32         relative_position=(0, 0),
33         icon=press_icon,
34         fill_colour=(0, 0, 0, 0),
35         border_width=0,
36         margin=0,
37         fit_icon=True,
38     )
39 }
40
41     super().__init__(
42         widgets_dict=widgets_dict,
43         **kwargs
44     )

1 import pygame
2 from data.widgets.bases.widget import _Widget
3
4 class Rectangle(_Widget):
5     def __init__(self, visible=False, **kwargs):
6         super().__init__(**kwargs)
7
8         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
9         self._visible = visible
10
11         self.set_image()
12         self.set_geometry()
13
14     def set_image(self):
15         self.image = pygame.transform.scale(self._empty_surface, self.size)
16         if self._visible:
17             pygame.draw.rect(self.image, self._fill_colour, self.image.get_rect(),
18                             border_radius=int(self.border_radius))
19
20             if self.border_width:
21                 pygame.draw.rect(self.image, self._border_colour, self.image.
22 get_rect(), width=int(self.border_width), border_radius=int(self.border_radius
23 ))
24
25     def process_event(self, event):
26         pass
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

20     self._vertical = vertical
21     self._last_mouse_px = None
22
23     self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
24
25     self.initialise_new_colours(self._fill_colour)
26     self.set_state_colour(WidgetState.BASE)
27
28     self.set_image()
29     self.set_geometry()
30
31     def down_func(self):
32         if self._vertical:
33             self._last_mouse_px = pygame.mouse.get_pos()[1]
34         else:
35             self._last_mouse_px = pygame.mouse.get_pos()[0]
36
37         self.set_state_colour(WidgetState.PRESS)
38
39     def up_func(self):
40         self._last_mouse_px = None
41         self.set_state_colour(WidgetState.BASE)
42
43     def set_relative_position(self, relative_position):
44         self._relative_position = relative_position
45         self.set_geometry()
46
47     def set_relative_size(self, new_relative_size):
48         self._relative_size = new_relative_size
49
50     def set_image(self):
51         self.image = pygame.transform.scale(self._empty_surface, self.size)
52
53         if self._vertical:
54             rounded_radius = self.size[0] / 2
55         else:
56             rounded_radius = self.size[1] / 2
57
58         pygame.draw.rect(self.image, self._fill_colour, (0, 0, self.size[0], self.size[1]), border_radius=int(rounded_radius))
59
60     def process_event(self, event):
61         before_state = self.get_widget_state()
62         widget_event = super().process_event(event)
63         after_state = self.get_widget_state()
64
65         if event.type == pygame.MOUSEMOTION and self._last_mouse_px:
66             if self._vertical:
67                 offset_from_last_frame = event.pos[1] - self._last_mouse_px
68                 self._last_mouse_px = event.pos[1]
69
70                 return offset_from_last_frame
71             else:
72                 offset_from_last_frame = event.pos[0] - self._last_mouse_px
73                 self._last_mouse_px = event.pos[0]
74
75                 return offset_from_last_frame
76
77
78     if widget_event or before_state != after_state:
79         return 0

```

```

1 import pygame
2 from data.widgets.bases.widget import _Widget
3 from data.widgets.scrollbar import _Scrollbar
4 from data.managers.theme import theme
5
6 SCROLLBAR_WIDTH_FACTOR = 0.05
7
8 class ScrollArea(_Widget):
9     def __init__(self, widget, vertical, scroll_factor=15, **kwargs):
10         super().__init__(**kwargs)
11         if vertical is False:
12             self._relative_size = kwargs.get('relative_size')
13
14             self._relative_scroll_factor = scroll_factor / self.surface_size[1]
15
16             self._scroll_percentage = 0
17             self._widget = widget
18             self._vertical = vertical
19
20             self._widget.register_get_rect(self.calculate_widget_rect)
21
22             if self._vertical:
23                 anchor_x = 'right'
24                 anchor_y = 'top'
25                 scale_mode = 'height'
26             else:
27                 anchor_x = 'left'
28                 anchor_y = 'bottom'
29                 scale_mode = 'width'
30
31             self._scrollbar = _Scrollbar(
32                 parent=self,
33                 relative_position=(0, 0),
34                 relative_size=None,
35                 anchor_x=anchor_x,
36                 anchor_y=anchor_y,
37                 fill_colour=theme['borderPrimary'],
38                 scale_mode=scale_mode,
39                 vertical=vertical,
40             )
41
42             self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
43
44             self.set_image()
45             self.set_geometry()
46
47     @property
48     def scroll_factor(self):
49         return self._relative_scroll_factor * self.surface_size[1]
50
51     @property
52     def scrollbar_size(self):
53         if self._vertical:
54             return (self.size[0] * SCROLLBAR_WIDTH_FACTOR, min(1, self.size[1] /
55 self._widget.rect.height) * self.size[1])
56         else:
57             return (min(1, self.size[0] / (self._widget.rect.width + 0.001)) *
58 self.size[0], self.size[1] * SCROLLBAR_WIDTH_FACTOR)
59
60     @property
61     def size(self):
62         if self._vertical is False:

```

```

61         return (self._relative_size[0] * self.surface_size[0], self.
        _relative_size[1] * self.surface_size[1]) # scale with horizontal width to
        always fill entire length of screen
62     else:
63         return super().size
64
65     def calculate_scroll_percentage(self, offset, scrollbar=False):
66         if self._vertical:
67             widget_height = self._widget.rect.height
68
69             if widget_height < self.size[1]:
70                 return 0
71
72             if scrollbar:
73                 self._scroll_percentage += offset / (self.size[1] - self.
scrollbar_size[1] + 0.001)
74             else:
75                 max_scroll_height = widget_height - self.size[1]
76                 current_scroll_height = self._scroll_percentage *
max_scroll_height
77                 self._scroll_percentage = (current_scroll_height + offset) / (
max_scroll_height + 0.001)
78             else:
79                 widget_width = self._widget.rect.width
80
81                 if widget_width < self.size[0]:
82                     return 0
83
84                 if scrollbar:
85                     self._scroll_percentage += offset / (self.size[0] - self.
scrollbar_size[0] + 0.001)
86                 else:
87                     max_scroll_width = widget_width - self.size[0]
88                     current_scroll_width = self._scroll_percentage * max_scroll_width
89                     self._scroll_percentage = (current_scroll_width + offset) /
max_scroll_width
90
91                 return min(1, max(0, self._scroll_percentage))
92
93     def calculate_widget_rect(self):
94         widget_position = self.calculate_widget_position()
95         return pygame.Rect(widget_position[0] - self.position[0], self.position
[1] + widget_position[1], self.size[0], self.size[1])
96
97     def calculate_widget_position(self):
98         if self._vertical:
99             return (0, -self._scroll_percentage * (self._widget.rect.height - self
.size[1]))
100         else:
101             return (-self._scroll_percentage * (self._widget.rect.width - self.
size[0]), 0)
102
103     def calculate_relative_scrollbar_position(self):
104         if self._vertical:
105             vertical_offset = (self.size[1] - self.scrollbar_size[1]) * self.
_scroll_percentage
106             scrollbar_position = (0, vertical_offset)
107         else:
108             horizontal_offset = (self.size[0] - self.scrollbar_size[0]) * self.
_scroll_percentage
109             scrollbar_position = (horizontal_offset, 0)
110

```

```

111         return (scrollbar_position[0] / self.size[0], scrollbar_position[1] / self
112                 .size[1])
113
114     def set_widget(self, new_widget):
115         self._widget = new_widget
116         self.set_image()
117         self.set_geometry()
118
119     def set_image(self):
120         self.image = pygame.transform.scale(self._empty_surface, self.size)
121         self.image.fill(theme['fillPrimary'])
122
123         self._widget.set_image()
124         self.image.blit(self._widget.image, self.calculate_widget_position())
125
126         self._scrollbar.set_relative_position(self.
127 calculate_relative_scrollbar_position()) # WRONG USING RELATIVE
128         self._scrollbar.set_relative_size((self.scrollbar_size[0] / self.size[1],
129 self.scrollbar_size[1] / self.size[1]))
130         self._scrollbar.set_image()
131         relative_scrollbar_position = (self._scrollbar.rect.left - self.position
132 [0], self._scrollbar.rect.top - self.position[1])
133         self.image.blit(self._scrollbar.image, relative_scrollbar_position)
134
135     def set_geometry(self):
136         super().set_geometry()
137         self._widget.set_geometry()
138         self._scrollbar.set_geometry()
139
140     def set_surface_size(self, new_surface_size):
141         super().set_surface_size(new_surface_size)
142         self._widget.set_surface_size(new_surface_size)
143         self._scrollbar.set_surface_size(new_surface_size)
144
145     def process_event(self, event):
146         # WAITING FOR PYGAME-CE 2.5.3 TO RELEASE TO FIX SCROLL FLAGS
147         # self.image.scroll(0, SCROLL_FACTOR)
148         # self.image.scroll(0, -SCROLL_FACTOR)
149
150         offset = self._scrollbar.process_event(event)
151
152         if offset is not None:
153             self.set_image()
154
155             if abs(offset) > 0:
156                 self._scroll_percentage = self.calculate_scroll_percentage(offset,
157 scrollbar=True)
158
159             if self.rect.collidepoint(pygame.mouse.get_pos()):
160                 if event.type == pygame.MOUSEBUTTONDOWN:
161                     if event.button == 4:
162                         self._scroll_percentage = self.calculate_scroll_percentage(-
163 self.scroll_factor)
164                         self.set_image()
165                         return
166                     elif event.button == 5:
167                         if self._scroll_percentage == 100:
168                             return
169
170                         self._scroll_percentage = self.calculate_scroll_percentage(
171 self.scroll_factor)
172                         self.set_image()

```

```

166         return
167
168     widget_event = self._widget.process_event(event, scrolled_pos=self.
calculate_widget_position())
169     if widget_event is not None:
170         self.set_image()
171     return widget_event

1 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
5 from data.managers.theme import theme
6
7 class _SliderThumb(_Pressable):
8     def __init__(self, radius, border_colour=theme['borderPrimary'], fill_colour=
theme['fillPrimary']):
9         super().__init__(
10             event=None,
11             down_func=self.down_func,
12             up_func=self.up_func,
13             hover_func=self.hover_func,
14             prolonged=True,
15             sfx=None
16         )
17         self._border_colour = border_colour
18         self._radius = radius
19         self._percent = None
20
21         self.state = WidgetState.BASE
22         self.initialise_new_colours(fill_colour)
23
24     def get_position(self):
25         return (self.rect.x, self.rect.y)
26
27     def set_position(self, position):
28         self.rect = self._thumb_surface.get_rect()
29         self.rect.topleft = position
30
31     def get_surface(self):
32         return self._thumb_surface
33
34     def set_surface(self, radius, border_width):
35         self._thumb_surface = create_slider_thumb(radius, self._colours[self.state
], self._border_colour, border_width)
36
37     def get_pressed(self):
38         return self._pressed
39
40     def down_func(self):
41         self.state = WidgetState.PRESS
42
43     def up_func(self):
44         self.state = WidgetState.BASE
45
46     def hover_func(self):
47         self.state = WidgetState.HOVER

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

```



```

5 from data.utils.widget_helpers import create_switch
6 from data.components.custom_event import CustomEvent
7 from data.managers.theme import theme
8
9 class Switch(_Pressable, _Widget):
10     def __init__(self, relative_height, event, fill_colour=theme['fillTertiary'],
11         on_colour=theme['fillSecondary'], off_colour=theme['fillPrimary'], **kwargs):
12         _Pressable.__init__(
13             self,
14             event=event,
15             hover_func=self.hover_func,
16             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
17             up_func=self.up_func,
18         )
19         _Widget.__init__(self, relative_size=(relative_height * 2, relative_height
20             ), scale_mode='height', fill_colour=fill_colour, **kwargs)
21
22         self._on_colour = on_colour
23         self._off_colour = off_colour
24         self._background_colour = None
25
26         self._is_toggled = None
27         self.set_toggle_state(False)
28
29         self.initialise_new_colours(self._fill_colour)
30         self.set_state_colour(WidgetState.BASE)
31
32         self.set_image()
33         self.set_geometry()
34
35     def hover_func(self):
36         self.set_state_colour(WidgetState.HOVER)
37
38     def set_toggle_state(self, is_toggled):
39         self._is_toggled = is_toggled
40         if is_toggled:
41             self._background_colour = self._on_colour
42         else:
43             self._background_colour = self._off_colour
44
45         self.set_image()
46
47     def up_func(self):
48         if self.get_widget_state() == WidgetState.PRESS:
49             toggle_state = not(self._is_toggled)
50             self.set_toggle_state(toggle_state)
51
52         self.set_state_colour(WidgetState.BASE)
53
54     def draw_thumb(self):
55         margin = self.size[1] * 0.1
56         thumb_radius = (self.size[1] / 2) - margin
57
58         if self._is_toggled:
59             thumb_center = (self.size[0] - margin - thumb_radius, self.size[1] /
60                 2)
61         else:
62             thumb_center = (margin + thumb_radius, self.size[1] / 2)
63
64         pygame.draw.circle(self.image, self._fill_colour, thumb_center,
65             thumb_radius)

```

```

63     def set_image(self):
64         self.image = create_switch(self.size, self._background_colour)
65         self.draw_thumb()
66
67     def process_event(self, event):
68         data = super().process_event(event)
69
70         if data:
71             return CustomEvent(**vars(data), toggled=self._is_toggled)

```

```

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,
   text_height_to_font_size, height_to_font_size
5  from data.utils.widget_helpers import create_text_box
6  from data.assets import GRAPHICS
7
8  class Text(_Widget): # Pure text
9      def __init__(self, text, center=True, fit_vertical=True, box_colours=None,
   strength=0.05, font_size=None, **kwargs):
10         super().__init__(**kwargs)
11         self._text = text
12         self._fit_vertical = fit_vertical
13         self._strength = strength
14         self._box_colours = box_colours
15
16         if fit_vertical:
17             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         else:
19             self._relative_font_size = text_width_to_font_size(self._text, self.
   _font, (self.size[0] - 2 * (self.margin + self.border_width))) / self.
   surface_size[1]
20
21         if font_size:
22             self._relative_font_size = font_size / self.surface_size[1]
23
24         self._center = center
25         self.rect = self._font.get_rect(self._text, size=self.font_size)
26         self.rect.topleft = self.position
27
28         self._empty_surface = pygame.Surface((0, 0), pygame.SRCALPHA)
29
30         self.set_image()
31         self.set_geometry()
32
33     def resize_text(self):
34         if self._fit_vertical:
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]
36         else:
37             ideal_font_size = height_to_font_size(self._font, target_height=(self.
   size[1] - (self.margin + self.border_width))) / self.surface_size[1]
38             new_font_size = text_width_to_font_size(self._text, self._font, (self.
   size[0] - (self.margin + self.border_width))) / self.surface_size[1]
39
40             if new_font_size < ideal_font_size:
41                 self._relative_font_size = new_font_size
42             else:

```

```

43         self._relative_font_size = ideal_font_size
44
45     def set_text(self, new_text):
46         self._text = new_text
47
48         self.resize_text()
49         self.set_image()
50
51     def set_image(self):
52         if self._box_colours:
53             self.image = create_text_box(self.size, self.border_width, self.
54             _box_colours)
55         else:
56             text_surface = pygame.transform.scale(self._empty_surface, self.size)
57             self.image = text_surface
58
59             if self._fill_colour:
60                 fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
61                 pygame.draw.rect(self.image, self._fill_colour, fill_rect,
62                 border_radius=int(self.border_radius))
63
64                 self._font.strength = self._strength
65                 font_rect_size = self._font.get_rect(self._text, size=self.font_size).size
66                 if self._center:
67                     font_position = ((self.size[0] - font_rect_size[0]) / 2, (self.size[1]
68                     - font_rect_size[1]) / 2)
69                 else:
70                     font_position = (self.margin / 2, (self.size[1] - font_rect_size[1]) /
71                     2)
72                 self._font.render_to(self.image, font_position, self._text, fgcolor=self.
73                 _text_colour, size=self.font_size)
74
75                 if self._box_colours is None and self.border_width:
76                     fill_rect = pygame.FRect(0, 0, self.size[0], self.size[1])
77                     pygame.draw.rect(self.image, self._border_colour, fill_rect, width=int
78                     (self.border_width), border_radius=int(self.border_radius))
79
80     def process_event(self, event):
81         pass
82
83 from data.widgets.bases.pressable import _Pressable
84 from data.widgets.bases.box import _Box
85 from data.widgets.text import Text
86 from data.constants import WidgetState, BLUE_BUTTON_COLOURS
87
88 class TextButton(_Box, _Pressable, Text):
89     def __init__(self, event, **kwargs):
90         _Box.__init__(self, box_colours=BLUE_BUTTON_COLOURS)
91         _Pressable.__init__(
92             self,
93             event=event,
94             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
95             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
96             up_func=lambda: self.set_state_colour(WidgetState.BASE),
97         )
98         Text.__init__(self, box_colours=BLUE_BUTTON_COLOURS[WidgetState.BASE], **
99         kwargs)
100
101         self.initialise_new_colours(self._fill_colour)
102         self.set_state_colour(WidgetState.BASE)
103
104 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
12
13 logger = initialise_logger(__name__)
14
15 class TextInput(_Box, _Pressable, Text):
16     def __init__(self, event, blinking_interval=530, validator=(lambda x: True),
17         default='', placeholder='PLACEHOLDER TEXT', placeholder_colour=(200, 200, 200)
18         , cursor_colour=theme['textSecondary'], **kwargs):
19         self._cursor_index = None
20         # Multiple inheritance used here, adding the functionality of pressing,
21         and custom box colours, to the text widget
22         _Box.__init__(self, box_colours=INPUT_COLOURS)
23         _Pressable.__init__(
24             self,
25             event=None,
26             hover_func=lambda: self.set_state_colour(WidgetState.HOVER),
27             down_func=lambda: self.set_state_colour(WidgetState.PRESS),
28             up_func=lambda: self.set_state_colour(WidgetState.BASE),
29             sfx=None
30         )
31         Text.__init__(self, text="", center=False, box_colours=INPUT_COLOURS[
32             WidgetState.BASE], **kwargs)
33
34         self.initialise_new_colours(self._fill_colour)
35         self.set_state_colour(WidgetState.BASE)
36
37         pygame.key.set_repeat(500, 50)
38
39         self._blinking_fps = 1000 / blinking_interval
40         self._cursor_colour = cursor_colour
41         self._cursor_colour_copy = cursor_colour
42         self._placeholder_colour = placeholder_colour
43         self._text_colour_copy = self._text_colour
44
45         self._placeholder_text = placeholder
46         self._is_placeholder = None
47         if default:
48             self._text = default
49             self.is_placeholder = False
50         else:
51             self._text = self._placeholder_text
52             self.is_placeholder = True
53
54         self._event = event
55         self._validator = validator
56         self._blinking_cooldown = 0
57
58         self._empty_cursor = pygame.Surface((0, 0), pygame.SRCALPHA)
59
60         self.resize_text()
61         self.set_image()
62         self.set_geometry()

```

```

60 @property
61 # Encapsulated getter method
62 def is_placeholder(self):
63     return self._is_placeholder
64
65 @is_placeholder.setter
66 # Encapsulated setter method, used to replace text colour if placeholder text
67 # is shown
68 def is_placeholder(self, is_true):
69     self._is_placeholder = is_true
70
71     if is_true:
72         self._text_colour = self._placeholder_colour
73     else:
74         self._text_colour = self._text_colour_copy
75
76 @property
77 def cursor_size(self):
78     cursor_height = (self.size[1] - self.border_width * 2) * 0.75
79     return (cursor_height * 0.1, cursor_height)
80
81 @property
82 def cursor_position(self):
83     current_width = (self.margin / 2)
84     for index, metrics in enumerate(self._font.get_metrics(self._text, size=
85 self.font_size)):
86         if index == self._cursor_index:
87             return (current_width - self.cursor_size[0], (self.size[1] - self.
88 cursor_size[1]) / 2)
89
90         glyph_width = metrics[4]
91         current_width += glyph_width
92     return (current_width - self.cursor_size[0], (self.size[1] - self.
93 cursor_size[1]) / 2)
94
95 @property
96 def text(self):
97     if self.is_placeholder:
98         return ''
99
100     return self._text
101
102 def relative_x_to_cursor_index(self, relative_x):
103     """
104     Calculates cursor index using mouse position relative to the widget
105     position.
106
107     Args:
108         relative_x (int): Horizontal distance of the mouse from the left side
109         of the widget.
110
111     Returns:
112         int: Cursor index.
113     """
114     current_width = 0
115
116     for index, metrics in enumerate(self._font.get_metrics(self._text, size=
117 self.font_size)):
118         glyph_width = metrics[4]
119
120         if current_width >= relative_x:
121             return index

```

```

115         current_width += glyph_width
116
117     return len(self._text)
118
119
120 def set_cursor_index(self, mouse_pos):
121     """
122     Sets cursor index based on mouse position.
123
124     Args:
125         mouse_pos (list[int, int]): Mouse position relative to window screen.
126     """
127     if mouse_pos is None:
128         self._cursor_index = mouse_pos
129         return
130
131     relative_x = mouse_pos[0] - (self.margin / 2) - self.rect.left
132     relative_x = max(0, relative_x)
133     self._cursor_index = self.relative_x_to_cursor_index(relative_x)
134
135 def focus_input(self, mouse_pos):
136     """
137     Draws cursor and sets cursor index when user clicks on widget.
138
139     Args:
140         mouse_pos (list[int, int]): Mouse position relative to window screen.
141     """
142     if self.is_placeholder:
143         self._text = ''
144         self.is_placeholder = False
145
146     self.set_cursor_index(mouse_pos)
147     self.set_image()
148     cursor.set_mode(CursorMode.IBEAM)
149
150 def unfocus_input(self):
151     """
152     Removes cursor when user unselects widget.
153     """
154     if self._text == '':
155         self._text = self._placeholder_text
156         self.is_placeholder = True
157         self.resize_text()
158
159     self.set_cursor_index(None)
160     self.set_image()
161     cursor.set_mode(CursorMode.ARROW)
162
163 def set_text(self, new_text):
164     """
165     Called by a state object to change the widget text externally.
166
167     Args:
168         new_text (str): New text to display.
169
170     Returns:
171         CustomEvent: Object containing the new text to alert state of a text
172         update.
173     """
174     super().set_text(new_text)
175     return CustomEvent(**vars(self._event), text=self.text)

```

```

176 def process_event(self, event):
177     """
178     Processes Pygame events.
179
180     Args:
181         event (pygame.Event): Event to process.
182
183     Returns:
184         CustomEvent: Object containing the new text to alert state of a text
185         update.
186     """
187     previous_state = self.get_widget_state()
188     super().process_event(event)
189     current_state = self.get_widget_state()
190
191     match event.type:
192         case pygame.MOUSEMOTION:
193             if self._cursor_index is None:
194                 return
195
196             # If mouse is hovering over widget, turn mouse cursor into an I-
197             beam
198             if self.rect.collidepoint(event.pos):
199                 if cursor.get_mode() != CursorMode.IBEAM:
200                     cursor.set_mode(CursorMode.IBEAM)
201             else:
202                 if cursor.get_mode() == CursorMode.IBEAM:
203                     cursor.set_mode(CursorMode.ARROW)
204
205             return
206
207         case pygame.MOUSEBUTTONDOWN:
208             # When user selects widget
209             if previous_state == WidgetState.PRESS:
210                 self.focus_input(event.pos)
211             # When user unselects widget
212             if current_state == WidgetState.BASE and self._cursor_index is not
213             None:
214                 self.unfocus_input()
215                 return CustomEvent(**vars(self._event), text=self.text)
216
217         case pygame.KEYDOWN:
218             if self._cursor_index is None:
219                 return
220
221             # Handling Ctrl-C and Ctrl-V shortcuts
222             if event.mod & (pygame.KMOD_CTRL):
223                 if event.key == pygame.K_c:
224                     pyperclip.copy(self.text)
225                     logger.info(f'COPIED {self.text}')
226
227                 elif event.key == pygame.K_v:
228                     pasted_text = pyperclip.paste()
229                     pasted_text = ''.join(char for char in pasted_text if 32
230                     <= ord(char) <= 127)
231                     self._text = self._text[:self._cursor_index] + pasted_text
232                     + self._text[self._cursor_index:]
233                     self._cursor_index += len(pasted_text)
234
235                 elif event.key == pygame.K_BACKSPACE or event.key == pygame.
236                 K_DELETE:
237                     self._text = ''

```

```

232         self._cursor_index = 0
233
234         self.resize_text()
235         self.set_image()
236         self.set_geometry()
237
238         return
239
240     match event.key:
241     case pygame.K_BACKSPACE:
242         if self._cursor_index > 0:
243             self._text = self._text[:self._cursor_index - 1] +
self._text[self._cursor_index:]
244             self._cursor_index = max(0, self._cursor_index - 1)
245
246     case pygame.K_RIGHT:
247         self._cursor_index = min(len(self._text), self.
_cursor_index + 1)
248
249     case pygame.K_LEFT:
250         self._cursor_index = max(0, self._cursor_index - 1)
251
252     case pygame.K_ESCAPE:
253         self.unfocus_input()
254         return CustomEvent(**vars(self._event), text=self.text)
255
256     case pygame.K_RETURN:
257         self.unfocus_input()
258         return CustomEvent(**vars(self._event), text=self.text)
259
260     case _:
261         if not event.unicode:
262             return
263
264         potential_text = self._text[:self._cursor_index] + event.
unicode + self._text[self._cursor_index:]
265
266         # Validator lambda function used to check if inputted text
267         # is valid before displaying
268         # e.g. Time control input has a validator function
269         # checking if text represents a float
270         if self._validator(potential_text) is False:
271             return
272
273         self._text = potential_text
274         self._cursor_index += 1
275
276         self._blinking_cooldown += 1
277         animation.set_timer(500, lambda: self.subtract_blinking_cooldown
(1))
278
279         self.resize_text()
280         self.set_image()
281         self.set_geometry()
282
283     def subtract_blinking_cooldown(self, cooldown):
284         """
285         Subtracts blinking cooldown after certain timeframe. When
286         blinking_cooldown is 1, cursor is able to be drawn.
287
288         Args:
289             cooldown (float): Duration before cursor can no longer be drawn.

```



```

287         """
288         self._blinking_cooldown = self._blinking_cooldown - cooldown
289
290     def set_image(self):
291         """
292         Draws text input widget to image.
293         """
294         super().set_image()
295
296         if self._cursor_index is not None:
297             scaled_cursor = pygame.transform.scale(self._empty_cursor, self.
cursor_size)
298             scaled_cursor.fill(self._cursor_colour)
299             self.image.blit(scaled_cursor, self.cursor_position)
300
301     def update(self):
302         """
303         Overrides based update method, to handle cursor blinking.
304         """
305         super().update()
306         # Calculate if cursor should be shown or not
307         cursor_frame = animation.calculate_frame_index(0, 2, self._blinking_fps)
308         if cursor_frame == 1 and self._blinking_cooldown == 0:
309             self._cursor_colour = (0, 0, 0, 0)
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
6
7  class Timer(Text):
8      def __init__(self, active_colour, event=None, start_mins=60, **kwargs):
9          box_colours = BLUE_BUTTON_COLOURS[WidgetState.BASE] if active_colour ==
Colour.BLUE else RED_BUTTON_COLOURS[WidgetState.BASE]
10
11          self._current_ms = float(start_mins) * 60 * 1000
12          self._active_colour = active_colour
13          self._active = False
14          self._timer_running = False
15          self._event = event
16
17          super().__init__(text=self.format_to_text(), fit_vertical=False,
box_colours=box_colours, **kwargs)
18
19     def set_active(self, is_active):
20         if self._active == is_active:
21             return
22
23         if is_active and self._timer_running is False:
24             self._timer_running = True
25             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
31         self._text = self.format_to_text()

```

```

32         self.set_image()
33         self.set_geometry()
34
35     def get_time(self):
36         return self._current_ms / (1000 * 60)
37
38     def decrement_second(self):
39         if self._active:
40             self.set_time(self._current_ms - 1000)
41
42             if self._current_ms <= 0:
43                 self._active = False
44                 self._timer_running = False
45                 self.set_time(0)
46                 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:
50             self._timer_running = False
51
52     def format_to_text(self):
53         raw_seconds = self._current_ms / 1000
54         minutes, seconds = divmod(raw_seconds, 60)
55         return f'{str(int(minutes)).zfill(2)}:{str(int(seconds)).zfill(2)}'
56
57     def process_event(self, event):
58         if self._current_ms <= 0:
59             return CustomEvent(**vars(self._event), active_colour=self.
_active_colour)
60
61
62 1 import pygame
63 2 from data.widgets.bases.widget import _Widget
64 3 from data.widgets.slider_thumb import _SliderThumb
65 4 from data.components.custom_event import CustomEvent
66 5 from data.constants import SettingsEventType
67 6 from data.constants import WidgetState
68 7 from data.utils.widget_helpers import create_slider
69 8 from data.utils.asset_helpers import scale_and_cache
70 9 from data.managers.theme import theme
71
72 10
73 11 class VolumeSlider(_Widget):
74 12     def __init__(self, relative_length, default_volume, volume_type, thumb_colour=
theme['fillSecondary'], **kwargs):
75 13         super().__init__(relative_size=(relative_length, relative_length * 0.2),
**kwargs)
76
77 14
78 15         self._volume_type = volume_type
79 16         self._selected_percent = default_volume
80 17         self._last_mouse_x = None
81
82 18
83 19         self._thumb = _SliderThumb(radius=self.size[1] / 2, border_colour=self.
_border_colour, fill_colour=thumb_colour)
84 20         self._gradient_surface = create_slider(self.calculate_slider_size(), self.
_fill_colour, self.border_width, self._border_colour)
85
86 21
87 22         self._empty_surface = pygame.Surface(self.size, pygame.SRCALPHA)
88
89 23
90 24 @property
91 25 def position(self):
92 26     '''Minus so easier to position slider by starting from the left edge of
the slider instead of the thumb'''

```

```

27         return (self._relative_position[0] * self.surface_size[0] - (self.size[1]
/ 2), self._relative_position[1] * self.surface_size[1])
28
29     def calculate_slider_position(self):
30         return (self.size[1] / 2, self.size[1] / 4)
31
32     def calculate_slider_size(self):
33         return (self.size[0] - 2 * (self.size[1] / 2), self.size[1] / 2)
34
35     def calculate_selected_percent(self, mouse_pos):
36         if self._last_mouse_x is None:
37             return
38
39         x_change = (mouse_pos[0] - self._last_mouse_x) / (self.
calculate_slider_size()[0] - 2 * self.border_width)
40         return max(0, min(self._selected_percent + x_change, 1))
41
42     def calculate_thumb_position(self):
43         gradient_size = self.calculate_slider_size()
44         x = gradient_size[0] * self._selected_percent
45         y = 0
46
47         return (x, y)
48
49     def relative_to_global_position(self, position):
50         relative_x, relative_y = position
51         return (relative_x + self.position[0], relative_y + self.position[1])
52
53     def set_image(self):
54         gradient_scaled = scale_and_cache(self._gradient_surface, self.
calculate_slider_size())
55         gradient_position = self.calculate_slider_position()
56
57         self.image = pygame.transform.scale(self._empty_surface, (self.size))
58         self.image.blit(gradient_scaled, gradient_position)
59
60         thumb_position = self.calculate_thumb_position()
61         self._thumb.set_surface(radius=self.size[1] / 2, border_width=self.
border_width)
62         self._thumb.set_position(self.relative_to_global_position((thumb_position
[0], thumb_position[1])))
63
64         thumb_surface = self._thumb.get_surface()
65         self.image.blit(thumb_surface, thumb_position)
66
67     def set_volume(self, volume):
68         self._selected_percent = volume
69         self.set_image()
70
71     def process_event(self, event):
72         if event.type not in [pygame.MOUSEMOTION, pygame.MOUSEBUTTONDOWN, pygame.
MOUSEBUTTONUP]:
73             return
74
75         before_state = self._thumb.state
76         self._thumb.process_event(event)
77         after_state = self._thumb.state
78
79         if before_state != after_state:
80             self.set_image()
81
82         if event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:

```

```

83         self._last_mouse_x = None
84         return CustomEvent(SettingsEventType.VOLUME_SLIDER_CLICK, volume=
round(self._selected_percent, 3), volume_type=self._volume_type)
85
86     if self._thumb.state == WidgetState.PRESS:
87         selected_percent = self.calculate_selected_percent(event.pos)
88         self._last_mouse_x = event.pos[0]
89
90     if selected_percent:
91         self._selected_percent = selected_percent
92         self.set_image()
93         return CustomEvent(SettingsEventType.VOLUME_SLIDER_SLIDE)

1 from data.widgets.bases.widget import _Widget
2 from data.widgets.bases.pressable import _Pressable
3 from data.widgets.bases.circular import _Circular
4 from data.widgets.bases.box import _Box
5 from data.widgets.colour_display import _ColourDisplay
6 from data.widgets.colour_square import _ColourSquare
7 from data.widgets.colour_slider import _ColourSlider
8 from data.widgets.slider_thumb import _SliderThumb
9 from data.widgets.scrollbar import _Scrollbar
10
11 from data.widgets.board_thumbnail_button import BoardThumbnailButton
12 from data.widgets.multiple_icon_button import MultipleIconButton
13 from data.widgets.reactive_icon_button import ReactiveIconButton
14 from data.widgets.board_thumbnail import BoardThumbnail
15 from data.widgets.reactive_button import ReactiveButton
16 from data.widgets.volume_slider import VolumeSlider
17 from data.widgets.colour_picker import ColourPicker
18 from data.widgets.colour_button import ColourButton
19 from data.widgets.browser_strip import BrowserStrip
20 from data.widgets.piece_display import PieceDisplay
21 from data.widgets.browser_item import BrowserItem
22 from data.widgets.text_button import TextButton
23 from data.widgets.icon_button import IconButton
24 from data.widgets.scroll_area import ScrollArea
25 from data.widgets.chessboard import Chessboard
26 from data.widgets.text_input import TextInput
27 from data.widgets.rectangle import Rectangle
28 from data.widgets.move_list import MoveList
29 from data.widgets.dropdown import Dropdown
30 from data.widgets.carousel import Carousel
31 from data.widgets.switch import Switch
32 from data.widgets.timer import Timer
33 from data.widgets.text import Text
34 from data.widgets.icon import Icon
35
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
2
3 class _Box:
4     def __init__(self, box_colours):
5         self._box_colours_dict = box_colours
6         self._box_colours = self._box_colours_dict[WidgetState.BASE]
7
8     def set_state_colour(self, state):

```

```

9         self._box_colours = self._box_colours_dict[state]
10        super().set_state_colour(state)

1  from data.components.circular_linked_list import CircularLinkedList
2
3  class _Circular:
4      def __init__(self, items_dict, **kwargs):
5          # The key, value pairs are stored within a dictionary, while the keys to
6          # access them are stored within circular linked list.
7          self._items_dict = items_dict
8          self._keys_list = CircularLinkedList(list(items_dict.keys()))
9
10     @property
11     def current_key(self):
12         """
13         Gets the current head node of the linked list, and returns a key stored as
14         the node data.
15         Returns:
16             Data of linked list head.
17         """
18         return self._keys_list.get_head().data
19
20     @property
21     def current_item(self):
22         """
23         Gets the value in self._items_dict with the key being self.current_key.
24         Returns:
25             Value stored with key being current head of linked list.
26         """
27         return self._items_dict[self.current_key]
28
29     def set_next_item(self):
30         """
31         Sets the next item in as the current item.
32         """
33         self._keys_list.shift_head()
34
35     def set_previous_item(self):
36         """
37         Sets the previous item as the current item.
38         """
39         self._keys_list.unshift_head()
40
41     def set_to_key(self, key):
42         """
43         Sets the current item to the specified key.
44         Args:
45             key: The key to set as the current item.
46         Raises:
47             ValueError: If no nodes within the circular linked list contains the
48             key as its data.
49         """
50         if self._keys_list.data_in_list(key) is False:
51             raise ValueError('(_Circular.set_to_key) Key not found:', key)
52
53         for _ in range(len(self._items_dict)):
54             if self.current_key == key:
55                 self.set_image()
56                 self.set_geometry()

```

```

57         return
58
59     self.set_next_item()

1  import pygame
2  from data.constants import WidgetState
3  from data.managers.audio import audio
4  from data.assets import SFX
5
6  class _Pressable:
7      def __init__(self, event, down_func=None, up_func=None, hover_func=None,
8         prolonged=False, sfx=SFX['button_click'], **kwargs):
9          self._down_func = down_func
10         self._up_func = up_func
11         self._hover_func = hover_func
12         self._pressed = False
13         self._prolonged = prolonged
14         self._sfx = sfx
15
16         self._event = event
17
18         self._widget_state = WidgetState.BASE
19
20         self._colours = {}
21
22     def set_state_colour(self, state):
23         self._fill_colour = self._colours[state]
24
25         self.set_image()
26
27     def initialise_new_colours(self, colour):
28         r, g, b, a = pygame.Color(colour).rgba
29
30         self._colours = {
31             WidgetState.BASE: pygame.Color(r, g, b, a),
32             WidgetState.HOVER: pygame.Color(min(r + 25, 255), min(g + 25, 255),
33             min(b + 25, 255), a),
34             WidgetState.PRESS: pygame.Color(min(r + 50, 255), min(g + 50, 255),
35             min(b + 50, 255), a)
36         }
37
38     def get_widget_state(self):
39         return self._widget_state
40
41     def process_event(self, event):
42         match event.type:
43             case pygame.MOUSEBUTTONDOWN:
44                 if self.rect.collidepoint(event.pos):
45                     self._down_func()
46                     self._widget_state = WidgetState.PRESS
47
48             case pygame.MOUSEBUTTONUP:
49                 if self.rect.collidepoint(event.pos):
50                     if self._widget_state == WidgetState.PRESS:
51                         if self._sfx:
52                             audio.play_sfx(self._sfx)
53
54                     self._up_func()
55                     self._widget_state = WidgetState.HOVER
56                     return self._event
57
58                 elif self._widget_state == WidgetState.BASE:

```

```

56         self._hover_func()
57
58     elif self._prolonged and self._widget_state == WidgetState.PRESS:
59         if self._sfx:
60             audio.play_sfx(self._sfx)
61         self._up_func()
62         self._widget_state = WidgetState.BASE
63         return self._event
64
65     case pygame.MOUSEMOTION:
66         if self.rect.collidepoint(event.pos):
67             if self._widget_state == WidgetState.PRESS:
68                 return
69             elif self._widget_state == WidgetState.BASE:
70                 self._hover_func()
71                 self._widget_state = WidgetState.HOVER
72             elif self._widget_state == WidgetState.HOVER:
73                 self._hover_func()
74         else:
75             if self._prolonged is False:
76                 if self._widget_state in [WidgetState.PRESS, WidgetState.
HOVER]:
77                     self._widget_state = WidgetState.BASE
78                     self._up_func()
79                 elif self._widget_state == WidgetState.BASE:
80                     return
81             elif self._prolonged is True:
82                 if self._widget_state in [WidgetState.PRESS, WidgetState.
BASE]:
83                     return
84                 else:
85                     self._widget_state = WidgetState.BASE
86                     self._up_func()
87
88     1 import pygame
89     2 from data.constants import SCREEN_SIZE
90     3 from data.managers.theme import theme
91     4 from data.assets import DEFAULT_FONT
92
93     5
94     6 DEFAULT_SURFACE_SIZE = SCREEN_SIZE
95     7 REQUIRED_KWARGS = ['relative_position', 'relative_size']
96
97     8
98     9 class _Widget(pygame.sprite.Sprite):
99     10     def __init__(self, **kwargs):
100     11         """
101     12         Every widget has the following attributes:
102     13
103     14         surface (pygame.Surface): The surface the widget is drawn on.
104     15         raw_surface_size (tuple[int, int]): The initial size of the window screen,
105     16         remains constant.
106     17         parent (_Widget, optional): The parent widget position and size is
107     18         relative to.
109     19         Relative to current surface:
110     20         relative_position (tuple[float, float]): The position of the widget
111     21         relative to its surface.
112     22         relative_size (tuple[float, float]): The scale of the widget relative to
113     23         its surface.
114
115     24         Remains constant, relative to initial screen size:
116     25         relative_font_size (float, optional): The relative font size of the widget
117
118     .

```

```

24         relative_margin (float): The relative margin of the widget.
25         relative_border_width (float): The relative border width of the widget.
26         relative_border_radius (float): The relative border radius of the widget.
27
28         anchor_x (str): The horizontal anchor direction ('left', 'right', 'center
29         anchor_y (str): The vertical anchor direction ('top', 'bottom', 'center').
30         fixed_position (tuple[int, int], optional): The fixed position of the
31         widget in pixels.
32         border_colour (pygame.Color): The border color of the widget.
33         text_colour (pygame.Color): The text color of the widget.
34         fill_colour (pygame.Color): The fill color of the widget.
35         font (pygame.freetype.Font): The font used for the widget.
36         """
37         super().__init__()
38
39         for required_kwarg in REQUIRED_KWARGS:
40             if required_kwarg not in kwargs:
41                 raise KeyError(f'(_Widget.__init__) Required keyword "{
42                 required_kwarg}" not in base kwargs')
43
44         self._surface = None # Set in WidgetGroup, as needs to be reassigned every
45         frame
46         self._raw_surface_size = DEFAULT_SURFACE_SIZE
47
48         self._parent = kwargs.get('parent')
49
50         self._relative_font_size = None # Set in subclass
51
52         self._relative_position = kwargs.get('relative_position')
53         self._relative_margin = theme['margin'] / self._raw_surface_size[1]
54         self._relative_border_width = theme['borderWidth'] / self.
55         _raw_surface_size[1]
56         self._relative_border_radius = theme['borderRadius'] / self.
57         _raw_surface_size[1]
58
59         self._border_colour = pygame.Color(theme['borderPrimary'])
60         self._text_colour = pygame.Color(theme['textPrimary'])
61         self._fill_colour = pygame.Color(theme['fillPrimary'])
62         self._font = DEFAULT_FONT
63
64         self._anchor_x = kwargs.get('anchor_x') or 'left'
65         self._anchor_y = kwargs.get('anchor_y') or 'top'
66         self._fixed_position = kwargs.get('fixed_position')
67         self._scale_mode = kwargs.get('scale_mode') or 'both'
68
69         if kwargs.get('relative_size'):
70             match scale_mode:
71                 case 'height':
72                     self._relative_size = kwargs.get('relative_size')
73                 case 'width':
74                     self._relative_size = ((kwargs.get('relative_size')[0] * self.
75                     surface_size[0]) / self.surface_size[1], (kwargs.get('relative_size')[1] *
76                     self.surface_size[0]) / self.surface_size[1])
77                 case 'both':
78                     self._relative_size = ((kwargs.get('relative_size')[0] * self.
79                     surface_size[0]) / self.surface_size[1], kwargs.get('relative_size')[1])
80                 case _:
81                     raise ValueError(f'(_Widget.__init__) Unknown scale mode:',
82                     scale_mode)
83         else:
84             self._relative_size = (1, 1)

```



```

76
77         if 'margin' in kwargs:
78             self._relative_margin = kwargs.get('margin') / self._raw_surface_size
[1]
79
80             if (self._relative_margin * 2) > min(self._relative_size[0], self.
_relative_size[1]):
81                 raise ValueError('(_Widget.__init__) Margin larger than specified
size!')
82
83             if 'border_width' in kwargs:
84                 self._relative_border_width = kwargs.get('border_width') / self.
_raw_surface_size[1]
85
86             if 'border_radius' in kwargs:
87                 self._relative_border_radius = kwargs.get('border_radius') / self.
_raw_surface_size[1]
88
89             if 'border_colour' in kwargs:
90                 self._border_colour = pygame.Color(kwargs.get('border_colour'))
91
92             if 'fill_colour' in kwargs:
93                 self._fill_colour = pygame.Color(kwargs.get('fill_colour'))
94
95             if 'text_colour' in kwargs:
96                 self._text_colour = pygame.Color(kwargs.get('text_colour'))
97
98             if 'font' in kwargs:
99                 self._font = kwargs.get('font')
100
101     @property
102     def surface_size(self):
103         """
104         Gets the size of the surface widget is drawn on.
105         Can be either the window size, or another widget size if assigned to a
parent.
106
107         Returns:
108             tuple[int, int]: The size of the surface.
109         """
110         if self._parent:
111             return self._parent.size
112         else:
113             return self._raw_surface_size
114
115     @property
116     def position(self):
117         """
118         Gets the position of the widget.
119         Accounts for fixed position attribute, where widget is positioned in
pixels regardless of screen size.
120         Accounts for anchor direction, where position attribute is calculated
relative to one side of the screen.
121
122         Returns:
123             tuple[int, int]: The position of the widget.
124         """
125         x, y = None, None
126         if self._fixed_position:
127             x, y = self._fixed_position
128         if x is None:
129             x = self._relative_position[0] * self.surface_size[0]

```

```

130         if y is None:
131             y = self._relative_position[1] * self.surface_size[1]
132
133         if self._anchor_x == 'left':
134             x = x
135         elif self._anchor_x == 'right':
136             x = self.surface_size[0] - x - self.size[0]
137         elif self._anchor_x == 'center':
138             x = (self.surface_size[0] / 2 - self.size[0] / 2) + x
139
140         if self._anchor_y == 'top':
141             y = y
142         elif self._anchor_y == 'bottom':
143             y = self.surface_size[1] - y - self.size[1]
144         elif self._anchor_y == 'center':
145             y = (self.surface_size[1] / 2 - self.size[1] / 2) + y
146
147         # Position widget relative to parent, if exists.
148         if self._parent:
149             return (x + self._parent.position[0], y + self._parent.position[1])
150         else:
151             return (x, y)
152
153     @property
154     def size(self):
155         return (self._relative_size[0] * self.surface_size[1], self._relative_size
156                [1] * self.surface_size[1])
157
158     @property
159     def margin(self):
160         return self._relative_margin * self._raw_surface_size[1]
161
162     @property
163     def border_width(self):
164         return self._relative_border_width * self._raw_surface_size[1]
165
166     @property
167     def border_radius(self):
168         return self._relative_border_radius * self._raw_surface_size[1]
169
170     @property
171     def font_size(self):
172         return self._relative_font_size * self.surface_size[1]
173
174     def set_image(self):
175         """
176         Abstract method to draw widget.
177         """
178         raise NotImplementedError
179
180     def set_geometry(self):
181         """
182         Sets the position and size of the widget.
183         """
184         self.rect = self.image.get_rect()
185
186         if self._anchor_x == 'left':
187             if self._anchor_y == 'top':
188                 self.rect.topleft = self.position
189             elif self._anchor_y == 'bottom':
190                 self.rect.topleft = self.position
191             elif self._anchor_y == 'center':

```

```

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

```