```
# quarto_interface.py
# Ari Cohen

import pygame, sys
from pygame.locals import *
from buttons import *
from quarto_state import *

pygame.init()
MAIN_SURF = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Quarto')

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
LIGHT_GREEN = (0, 100, 0)
BLUE = (0, 0, 128)
YELLOW = (255, 255, 0)
LIGHT_YELLOW = (100, 100, 0)
RED = (255, 0, 0)
LIGHT_RED = (100, 0, 0)
GRAY = (150, 150, 150)

class Board():

    def __init__(self, position, board_width, square_width):
        self.position = position
        self.total_squares = board_width * board_width
        self.board_width = board_width
        self.square_width = square_width
        self.squares = []
        for square_num in range(self.total_squares):
            new_square = Square(square_num, self)
            self.squares.append(new_square)
        width = (board_width * square_width) + (5 * (board_width - 1))
        self.surface = pygame.Surface((width, width))
        self.board_rect = pygame.Rect(position[0], position[1], width, width)

    def set_square(self, row, col, piece):
        self.squares[(self.board_width * row) + col].click_action(piece)

    def get_square_coords(self, row, col):
        (x, y) = self.squares[(self.board_width * row) + col].get_location()
        x += self.position[0]
        y += self.position[1]
        return (x,y)

    def update_board_surface(self):
        self.surface.fill(BLACK)
        for square in self.squares:
            square_surf = square.get_square_surface()
            self.surface.blit(square_surf, square.position)
        return self.surface

    def draw_board(self):
        board_surf = self.update_board_surface()
```

```
            MAIN_SURF.blit(board_surf, self.position)

        def check_for_mouse(self, dragging_piece):
            mouse_x, mouse_y = pygame.mouse.get_pos()
            if self.board_rect.collidepoint([mouse_x, mouse_y]):
                mouse_x -= self.position[0]
                mouse_y -= self.position[1]
                col = mouse_x / (self.square_width + 5)
                row = mouse_y / (self.square_width + 5)
                bad_drag = self.squares[(self.board_width * row) + col].click_action
                    (dragging_piece)
                if(bad_drag):
                    return [False, [col, row]]
                else:
                    return [True, [col, row]]
            else:
                return [False, 0]

    class Square():

        EMPTY = 0

        def __init__(self, square_number, board):
            self.square_num = square_number
            self.board = board
            self.width = self.board.square_width
            row = self.square_num / self.board.board_width
            col = self.square_num % self.board.board_width
            self.position = ((self.width + 5) * col, (self.width + 5) * row)
            self.surface = pygame.Surface((self.width, self.width))
            self.surface.fill(GREEN)
            self.piece = Square.EMPTY

        def get_val(self):
            pass

        def set_val(self):
            pass

        def get_location(self):
            return self.position

        def get_square_surface(self):
            return self.surface

        def click_action(self, piece_num):
            if(self.piece == Square.EMPTY):
                raw_piece = pygame.image.load("Quarto_Pieces/Shadowed/Piece_"+str
                    (piece_num)+".png") #for 3d use jpg
                self.piece = pygame.transform.scale(raw_piece, (self.width, self.
                    width))
                self.surface.blit(self.piece, (0, 0))
                return False
            else:
                return True
```

```
class Piece_Holder():

    NO_CLICK = 7
    PIECE_WIDTH = 69

    def __init__(self, position, board):
        self.board = board
        self.position = position
        self.surface = pygame.Surface(((Piece_Holder.PIECE_WIDTH + 5) * 2,
            ((Piece_Holder.PIECE_WIDTH + 5) * (board.total_squares / 2)) + 5))
        self.squares = []
        for square in range(board.total_squares):
            self.squares.append(Holder_Square(square, self))
        self.holder_rect = pygame.Rect(position[0], position[1], (Piece_Holder.
            PIECE_WIDTH + 5) * 2, ((Piece_Holder.PIECE_WIDTH + 5) * (board.
            total_squares / 2)) + 5)

    def update_holder_surface(self):
        self.surface.fill(WHITE)
        #pygame.draw.rect(self.surface, BLACK, (0, 0, (Piece_Holder.PIECE_WIDTH +
            5) * 2, ((Piece_Holder.PIECE_WIDTH + 5) * (self.board.total_squares /
            2)) + 5), 5)
        for square in self.squares:
            square_surf = square.get_square_surface()
            self.surface.blit(square_surf, square.position)
        return self.surface


    def draw_holder(self):
        self.surface = self.update_holder_surface()
        MAIN_SURF.blit(self.surface, self.position)


    def check_for_click(self):
        mouse_x, mouse_y = pygame.mouse.get_pos()
        if self.holder_rect.collidepoint([mouse_x, mouse_y]):
            mouse_x -= self.position[0]
            mouse_y -= self.position[1]
            col = mouse_x / ((Piece_Holder.PIECE_WIDTH + 5))
            row = mouse_y / ((Piece_Holder.PIECE_WIDTH + 5))
            clicked_piece = self.squares[(row * 2) + col].click_action()
            if(clicked_piece == Holder_Square.EMPTY):
                return (Piece_Holder.NO_CLICK, 0)
            else:
                return (clicked_piece, (row * 2) + col)
        else:
            return (Piece_Holder.NO_CLICK, 0)

    def piece_returned(self, piece_num):
        self.squares[piece_num].returned()

    def get_square(self, square):
        return self.squares[square]

    def get_square_pos(self, square_num):
        return self.squares[square_num].get_pos()
```

```
class Holder_Square():

    EMPTY = -1

    def __init__(self, square_number, piece_holder):
        self.square_num = square_number
        self.holder = piece_holder
        self.position = ((self.square_num % 2) * (Piece_Holder.PIECE_WIDTH + 5),
                         ((self.square_num / 2) * (Piece_Holder.PIECE_WIDTH + 5))
                             + 5)
        self.surface = pygame.Surface(((Piece_Holder.PIECE_WIDTH + 2),
            (Piece_Holder.PIECE_WIDTH + 2)))
        if(self.square_num != 15):
            raw_piece = pygame.image.load("Quarto_Pieces/Shadowless/Piece_"+str
                (self.square_num)+".png")
            self.primary_piece = pygame.transform.scale(raw_piece, (Piece_Holder.
                PIECE_WIDTH, Piece_Holder.PIECE_WIDTH))
            self.piece = self.primary_piece
        else:
            self.primary_piece = Holder_Square.EMPTY
            self.piece = self.primary_piece

    def get_piece(self):
        return self.piece

    def set_piece(self, piece):
        self.piece = piece

    def get_square_surface(self):
        self.surface.fill(WHITE)
        pygame.draw.rect(self.surface, BLACK, (0, 0, (Piece_Holder.PIECE_WIDTH +
            2),
                                                (Piece_Holder.PIECE_WIDTH + 2)), 5
                                                    )
        if (self.piece != Holder_Square.EMPTY):
            self.surface.blit(self.piece, (1, 1))
        return self.surface

    def click_action(self):
        place_holder = self.piece
        self.piece = Holder_Square.EMPTY
        return place_holder

    def returned(self):
        self.piece = self.primary_piece

    def get_pos(self):
        return self.position

class Next_Piece_Box():

    def __init__(self, position):
        self.position = position
        self.surface = pygame.Surface((Piece_Holder.PIECE_WIDTH + 2, Piece_Holder
            .PIECE_WIDTH + 2))
```

```
        raw_piece = pygame.image.load("Quarto_Pieces/Shadowless/Piece_15.png")
        self.piece = pygame.transform.scale(raw_piece, (Piece_Holder.PIECE_WIDTH,
            Piece_Holder.PIECE_WIDTH))
        self.square_rect = pygame.Rect(position[0], position[1], Piece_Holder.
            PIECE_WIDTH + 2, Piece_Holder.PIECE_WIDTH + 2)
        self.piece_num = 15

    def get_current_piece(self):
        return (self.piece, self.piece_num)

    def update_box_surface(self):
        self.surface.fill(WHITE)
        pygame.draw.rect(self.surface, BLACK, (0, 0, (Piece_Holder.PIECE_WIDTH +
            2),
                                                (Piece_Holder.PIECE_WIDTH + 2)), 5
                                                    )
        if(self.piece != Holder_Square.EMPTY):
            self.surface.blit(self.piece, (1, 1))
        return self.surface

    def draw_box(self):
        self.surface = self.update_box_surface()
        MAIN_SURF.blit(self.surface, self.position)

    def check_for_click(self, piece_num):
        mouse_x, mouse_y = pygame.mouse.get_pos()
        if self.square_rect.collidepoint([mouse_x, mouse_y]):
            if(self.piece != Holder_Square.EMPTY):
                place_holder = (self.piece, self.piece_num)
                self.piece = Holder_Square.EMPTY
                self.piece_num = Holder_Square.EMPTY
                return place_holder
            else:
                self.on_mouse_drop(piece_num)
                return (0, 0)
        else:
            return (Piece_Holder.NO_CLICK, 0)

    def clear_box(self):
        self.piece = Holder_Square.EMPTY
        self.piece_num = Holder_Square.EMPTY

    def on_mouse_drop(self, piece_num):
        self.piece_num = piece_num
        raw_piece = pygame.image.load("Quarto_Pieces/Shadowless/Piece_"+str
            (piece_num)+".png")
        self.piece = pygame.transform.scale(raw_piece, (Piece_Holder.PIECE_WIDTH,
            Piece_Holder.PIECE_WIDTH))
        self.surface.blit(self.piece, (1, 1))

class MainInterface():

    def __init__(self):
        self.board = Board((350, 88), 4, 100)
        self.piece_holder = Piece_Holder((10, 0), self.board)
        self.next_piece_box = Next_Piece_Box((200, 250))
```

```
        self.dragging_piece = False
        self.piece_moving = False
        self.piece_num = -1

    def get_board(self):
        return self.board

    def get_piece_holder(self):
        return self.piece_holder

    def get_next_piece_box(self):
        return self.next_piece_box

    def do_event_fetch(self):
        MAIN_SURF.fill(WHITE)
        self.board.draw_board()
        self.piece_holder.draw_holder()
        self.next_piece_box.draw_box()

        for event in pygame.event.get():
            if(event.type == QUIT):
                pygame.quit()
                sys.exit()
            elif(event.type == KEYDOWN):
                if event.key == K_ESCAPE:
                    pygame.quit()
                    sys.exit()
                key_map = pygame.key.get_pressed()
                return key_map
            elif(event.type == MOUSEBUTTONDOWN):
                return MOUSEBUTTONDOWN
            elif(event.type == MOUSEBUTTONUP):
                return MOUSEBUTTONUP


def get_players_information(interface_state):
    player_radios = [RadioButtonGroup((200, 50), 3, MAIN_SURF,
                                      names=["Human", "Computer", "Network"],
                                      color=BLACK, text_size=32),
                     RadioButtonGroup((400, 50), 3, MAIN_SURF,
                                      names=["Human", "Computer", "Network"],
                                      color=BLACK, text_size=32)]
    button_font = pygame.font.Font('freesansbold.ttf', 52)
    text_surface = button_font.render("Go!", True, BLACK, GREEN)
    text_rect = text_surface.get_rect()
    text_rect.topleft = (350, 250)

    while True: #Wait for "Go!" to be pressed
        events = interface_state.do_event_fetch()
        MAIN_SURF.fill(WHITE)
        player_radios[0].draw_surface()
        player_radios[1].draw_surface()
        MAIN_SURF.blit(text_surface, (350, 250))
        pygame.display.update()

        if (events == MOUSEBUTTONDOWN):
```

```
            for radio in player_radios:
                radio.check_for_click()
            if (text_rect.collidepoint(pygame.mouse.get_pos())):
                break
    vals = ["h", "c", "n"]
    selected = []
    for radio in player_radios:
        selected.append(vals[radio.get_selected()])
        selected.append(3) #This would be computer strength
    return selected

def display_game_state(game_state):
    #Board, Piece Holder, and Next Piece Box act as an interface version of
    #the GameState class, so since they keep track of everything we don't need
    #To do much here
    interface_state = game_state.get_interface()
    interface_state.do_event_fetch()
    pygame.display.update()

def make_move_and_display(game_state, move):
    interface_state = game_state.interface_state
    board = interface_state.get_board()
    game_state.make_move(move)
    new_piece = move.get_piece()
    current_interface_piece = interface_state.get_next_piece_box().
        get_current_piece()
    if (new_piece != current_interface_piece[1]):
        #Check if interface has already made move (if it was a human move)
        #Otherwise animate the move for user
        #Animate piece to board
        interface_state.get_next_piece_box().clear_box()
        col = move.get_col_placement()
        row = move.get_row_placement()
        move_surface(current_interface_piece[0], (200, 250), board.
            get_square_coords(row, col), interface_state)
        board.set_square(row, col, current_interface_piece[1])
        #Animate piece to next piece square
        if (new_piece != -1):
            holder_square = interface_state.get_piece_holder().get_square
                (new_piece)
            new_current_piece = holder_square.click_action()
            location = holder_square.get_pos()
            move_surface(new_current_piece, location, (200, 250), interface_state
                )
            interface_state.get_next_piece_box().on_mouse_drop(new_piece)

    display_game_state(game_state)

def get_human_move(game_state):
    interface_state = game_state.get_interface()
    while True: #Loop until user has dropped piece on a good board space
        while True: #Waiting for user to pick up piece
            events = interface_state.do_event_fetch()
            pygame.display.update()
            if (events == MOUSEBUTTONDOWN):
                [piece_surf, piece_num] = interface_state.get_next_piece_box().
```

```
                    check_for_click(0)
                if (piece_surf != Piece_Holder.NO_CLICK):
                    break
            while True: #User is dragging piece
                events = interface_state.do_event_fetch()
                MAIN_SURF.blit(piece_surf, pygame.mouse.get_pos())
                pygame.display.update()
                if (events == MOUSEBUTTONUP):
                    #square is the variable needed for making an instance of GameMove
                    [piece_on_board, square] = interface_state.get_board().
                        check_for_mouse(piece_num)
                    break

            if (piece_on_board): #Piece was dropped on board in good square
                break
            else: #Bad move, animate return of the piece
                move_surface(piece_surf, pygame.mouse.get_pos(), (200, 250),
                    interface_state)
                interface_state.get_next_piece_box().on_mouse_drop(piece_num)

        while True: #Waiting for user to drag in the next piece to play
            while True: #Waiting for user to pick up piece
                events = interface_state.do_event_fetch()
                pygame.display.update()
                if (events == MOUSEBUTTONDOWN):
                    [piece_surf, piece_num] = interface_state.get_piece_holder().
                        check_for_click()
                    if (piece_surf != Piece_Holder.NO_CLICK):
                        break
            while True: #User is dragging piece
                events = interface_state.do_event_fetch()
                MAIN_SURF.blit(piece_surf, pygame.mouse.get_pos())
                pygame.display.update()
                if (events == MOUSEBUTTONUP):
                    click_response = interface_state.get_next_piece_box().
                        check_for_click(piece_num)
                    break

            if (click_response[0] != Piece_Holder.NO_CLICK): #Piece was successfully
                dropped in square
                break
            else: #Bad drop, animate return of the piece
                move_surface(piece_surf, pygame.mouse.get_pos(), interface_state.
                    get_piece_holder().get_square_pos(piece_num), interface_state)
                interface_state.get_piece_holder().piece_returned(piece_num)

        move = GameMove()
        move.set_move(square[0], square[1], piece_num)
        return [move, GameStatus.PLAYING]

    def move_surface(surface, start, destination, interface_state):
        #For animated movements
        #1.1 is the optimal multiplying factor
        #If animating from one corner of the screen to the other (1000 pixels)
        #It will take about 46 frames, or approximately 1 second (since there is no
            fps clock)
```

```
        current_pos = start
        while ((abs(current_pos[0] - destination[0]) > 5) or (abs(current_pos[1] -
            destination[1]) > 5)):
            new_x = (int((current_pos[0] - destination[0])/1.1) + destination[0])
            new_y = (int((current_pos[1] - destination[1])/1.1) + destination[1])
            current_pos = (new_x, new_y)
            #simple exponential decay in the form of An=f(An-1)
            interface_state.do_event_fetch()
            MAIN_SURF.blit(surface, current_pos)
            pygame.display.update()

    def signal_end_of_game(game_status, game_state, player_1,
                           player_2, current_player):
        interface_state = game_state.interface_state
        player_1.game_over(game_state)
        player_2.game_over(game_state)
        if game_status == GameStatus.QUITTING: #Won't ever happen with an interface
            game_text = "Game over - player quitting."
        elif game_status == GameStatus.TIE:
            game_text = "It's a tie."
        elif game_status == GameStatus.WIN:
            game_text = "Player "+current_player.player_num+" wins!"
        else:
            game_text = "Game over - unknown reason"

        my_font = pygame.font.Font('freesansbold.ttf', 52)
        text_surface = my_font.render(game_text, True, BLACK, YELLOW)
        button_surface = my_font.render("Play Again!", True, BLACK, YELLOW)
        button_rect = button_surface.get_rect()
        button_rect.topleft = (300, 520)

        while True:
            events = interface_state.do_event_fetch()
            MAIN_SURF.blit(text_surface, (250, 20))
            MAIN_SURF.blit(button_surface, (300, 520))
            pygame.display.update()
            if (events == MOUSEBUTTONDOWN):
                if (button_rect.collidepoint(pygame.mouse.get_pos())):
                    break
```