c0a2400443 / ProjExD_Group09

Code Pull requests Actions Projects Wiki

C0A24004/King ProjExD_Group09 / chess.py Go to file t

c0a2400443 ゲームの終了を実装 464175f · 13 minutes ago

338 lines (286 loc) · 12.6 KB

Code Blame Raw

```python
import pygame
import sys
from enum import Enum

# 初期化
pygame.init()

# 定数
BOARD_SIZE = 8
SQUARE_SIZE = 80
WINDOW_WIDTH = BOARD_SIZE * SQUARE_SIZE
WINDOW_HEIGHT = BOARD_SIZE * SQUARE_SIZE + 100  # 情報表示用のスペース
FPS = 60

# 色の定義
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
LIGHT_BROWN = (240, 217, 181)
DARK_BROWN = (181, 136, 99)
HIGHLIGHT_COLOR = (255, 255, 0, 128)
SELECTED_COLOR = (0, 255, 0, 128)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

class PieceType(Enum):
    PAWN = 1
    ROOK = 2
    KNIGHT = 3
    BISHOP = 4
    QUEEN = 5
    KING = 6

class PieceColor(Enum):
    WHITE = 1
    BLACK = 2

class Piece:
    def __init__(self, piece_type, color, row, col):
        self.type = piece_type
        self.color = color
        self.row = row
        self.col = col
        self.has_moved = False

    def get_possible_moves(self, board):
        """駒の可能な動きを取得（現在は全方向移動可能）"""
        moves = []
        if self.type == PieceType.KING:
            directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0),  (1, 1)]
            for dr, dc in directions:
                new_row = self.row + dr
                new_col = self.col + dc
                if 0 <= new_row < 8 and 0 <= new_col < 8:
                    target_piece = board.get_piece(new_row, new_col)
                    if not target_piece or target_piece.color != self.color:
                        moves.append((new_row, new_col))
        else:
            for row in range(8):
                for col in range(8):
                    if board.is_valid_move(self.row, self.col, row, col):
                        moves.append((row, col))
        return moves

    def move(self, new_row, new_col):
        """駒を移動"""
        self.row = new_row
        self.col = new_col
        self.has_moved = True

    def __str__(self):
        symbols = {
            PieceType.PAWN: "P",
            PieceType.ROOK: "R",
            PieceType.KNIGHT: "N",
            PieceType.BISHOP: "B",
            PieceType.QUEEN: "Q",
            PieceType.KING: "K"
        }
        return symbols[self.type]

    def get_display_color(self):
        """駒の表示色を取得"""
        return BLACK if self.color == PieceColor.BLACK else WHITE

class ChessBoard:
    def __init__(self):
        self.board = [[None for _ in range(8)] for _ in range(8)]
        self.current_turn = PieceColor.WHITE
        self.selected_piece = None
        self.selected_pos = None
        self.winner = None
        self.possible_moves = []
        self.setup_initial_position()

    def setup_initial_position(self):
        """初期配置を設定"""
        # 黒の駒
        piece_order = [PieceType.ROOK, PieceType.KNIGHT, PieceType.BISHOP, PieceType.QUEEN,
                       PieceType.KING, PieceType.BISHOP, PieceType.KNIGHT, PieceType.ROOK]

        for col in range(8):
            # 黒の駒
            self.board[0][col] = Piece(piece_order[col], PieceColor.BLACK, 0, col)
            self.board[1][col] = Piece(PieceType.PAWN, PieceColor.BLACK, 1, col)

            # 白の駒
            self.board[7][col] = Piece(piece_order[col], PieceColor.WHITE, 7, col)
            self.board[6][col] = Piece(PieceType.PAWN, PieceColor.WHITE, 6, col)

    def get_piece(self, row, col):
        """指定位置の駒を取得"""
        if 0 <= row < 8 and 0 <= col < 8:
            return self.board[row][col]
        return None

    def set_piece(self, row, col, piece):
        """指定位置に駒を配置"""
        if 0 <= row < 8 and 0 <= col < 8:
            self.board[row][col] = piece

    def is_valid_move(self, from_row, from_col, to_row, to_col):
        """移動が有効かチェック（基本的な範囲チェック）"""
        if not (0 <= to_row < 8 and 0 <= to_col < 8):
            return False

        piece = self.get_piece(from_row, from_col)
        if not piece:
            return False

        target_piece = self.get_piece(to_row, to_col)
        if target_piece and target_piece.color == piece.color:
            return False

        return True

    def make_move(self, from_row, from_col, to_row, to_col):
        """駒を移動"""
        piece = self.get_piece(from_row, from_col)

        # 基本的な移動可能性チェック
        if not piece:
            return False

        # 現在のターンの駒かチェック
        if piece.color != self.current_turn:
            return False

        # 移動先が有効かチェック
        if not self.is_valid_move(from_row, from_col, to_row, to_col):
            return False

        #target_pieceを定義
        target_piece = self.get_piece(to_row, to_col)

        # キングが取られたら勝敗を設定
        if target_piece and target_piece.type == PieceType.KING:
            self.set_piece(to_row, to_col, piece)
            self.set_piece(from_row, from_col, None)
            piece.move(to_row, to_col)
            self.winner = piece.color  # 勝った側の色
            return True


        # 移動実行
        self.set_piece(to_row, to_col, piece)
        self.set_piece(from_row, from_col, None)
        piece.move(to_row, to_col)

        # ターン切り替え
        self.current_turn = PieceColor.BLACK if self.current_turn == PieceColor.WHITE else PieceColor.
        return True

    def select_piece(self, row, col):
        """駒を選択"""
        piece = self.get_piece(row, col)
        if piece and piece.color == self.current_turn:
            self.selected_piece = piece
            self.selected_pos = (row, col)
            self.possible_moves = piece.get_possible_moves(self)
            return True
        return False

    def deselect_piece(self):
        """駒の選択を解除"""
        self.selected_piece = None
        self.selected_pos = None
        self.possible_moves = []

class ChessGame:
    def __init__(self):
        self.screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
        pygame.display.set_caption("チェスゲーム")
        self.clock = pygame.time.Clock()
        self.board = ChessBoard()
        # フォントの設定（日本語対応）
        try:
            self.font = pygame.font.Font("msgothic.ttc", 24)  # Windows
        except:
            try:
                self.font = pygame.font.Font("NotoSansCJK-Regular.ttc", 24)  # Linux
            except:
                self.font = pygame.font.Font(None, 24)  # フォールバック

        self.piece_font = pygame.font.Font(None, 60)

    def get_board_pos(self, mouse_pos):
        """マウス位置をボード座標に変換"""
        x, y = mouse_pos
        if 0 <= x < WINDOW_WIDTH and 0 <= y < BOARD_SIZE * SQUARE_SIZE:
            col = x // SQUARE_SIZE
            row = y // SQUARE_SIZE
            return row, col
        return None, None

    def draw_board(self):
        """チェスボードを描画"""
        for row in range(BOARD_SIZE):
            for col in range(BOARD_SIZE):
                color = LIGHT_BROWN if (row + col) % 2 == 0 else DARK_BROWN
                rect = pygame.Rect(col * SQUARE_SIZE, row * SQUARE_SIZE, SQUARE_SIZE, SQUARE_SIZE)
                pygame.draw.rect(self.screen, color, rect)

                # 選択中のマスをハイライト
                if self.board.selected_pos == (row, col):
                    highlight_surface = pygame.Surface((SQUARE_SIZE, SQUARE_SIZE), pygame.SRCALPHA)
                    highlight_surface.fill(SELECTED_COLOR)
                    self.screen.blit(highlight_surface, (col * SQUARE_SIZE, row * SQUARE_SIZE))

                # 可能な移動先をハイライト
                if (row, col) in self.board.possible_moves:
                    highlight_surface = pygame.Surface((SQUARE_SIZE, SQUARE_SIZE), pygame.SRCALPHA)
                    highlight_surface.fill(HIGHLIGHT_COLOR)
                    self.screen.blit(highlight_surface, (col * SQUARE_SIZE, row * SQUARE_SIZE))

    def draw_pieces(self):
        """駒を描画"""
        for row in range(BOARD_SIZE):
            for col in range(BOARD_SIZE):
                piece = self.board.get_piece(row, col)
                if piece:
                    # 駒のテキストを描画
                    text_color = piece.get_display_color()
                    # 背景色を設定（見やすくするため）
                    bg_color = WHITE if text_color == BLACK else BLACK

                    text = self.piece_font.render(str(piece), True, text_color)
                    text_rect = text.get_rect(center=(col * SQUARE_SIZE + SQUARE_SIZE // 2,
                                                    row * SQUARE_SIZE + SQUARE_SIZE // 2))

                    # 背景の円を描画
                    pygame.draw.circle(self.screen, bg_color, text_rect.center, 25)
                    pygame.draw.circle(self.screen, text_color, text_rect.center, 25, 2)

                    self.screen.blit(text, text_rect)

    def draw_info(self):
        """ゲーム情報を描画"""
        info_y = BOARD_SIZE * SQUARE_SIZE + 10

        if self.board.winner != None:
            winner_color = "White" if self.board.winner == PieceColor.WHITE else "Black"
            result_text = f"{winner_color} wins!"
            text = self.font.render(result_text, True, RED)
            self.screen.blit(text, (10, info_y))
            return  # 勝敗が決まったら他の表示は不要

        # 現在のターン（英語で表示）
        turn_text = f"Current Turn: {'White' if self.board.current_turn == PieceColor.WHITE else 'Blac
        text = self.font.render(turn_text, True, BLACK)
        self.screen.blit(text, (10, info_y))

        # 選択中の駒（英語で表示）
        if self.board.selected_piece:
            piece = self.board.selected_piece
```

```python
                color_name = "White" if piece.color == PieceColor.WHITE else "Black"
                selected_text = f"Selected: {color_name} {str(piece)} at ({piece.row}, {piece.col})"
                text = self.font.render(selected_text, True, BLACK)
                self.screen.blit(text, (10, info_y + 30))

        def handle_click(self, mouse_pos):
            """マウスクリックを処理"""
            if self.board.winner:
                return # 勝敗が決まったらクリック操作無効

            row, col = self.get_board_pos(mouse_pos)
            if row is not None and col is not None:
                if self.board.selected_piece:
                    # 駒が選択されている場合
                    if (row, col) in self.board.possible_moves:
                        # 有効な移動先がクリックされた場合
                        from_row, from_col = self.board.selected_pos
                        if self.board.make_move(from_row, from_col, row, col):
                            print(f"Move made: {from_row},{from_col} -> {row},{col}")
                            print(f"Turn changed to: {self.board.current_turn}")
                            self.board.deselect_piece()
                        else:
                            print("Move failed")
                    elif self.board.select_piece(row, col):
                        # 別の駒を選択（現在のターンの駒のみ）
                        print(f"Selected piece: {self.board.selected_piece} at ({row}, {col})")
                    else:
                        # 無効な場所がクリックされた場合、選択解除
                        self.board.deselect_piece()
                        print("Deselected piece")
                else:
                    # 駒が選択されていない場合
                    if self.board.select_piece(row, col):
                        print(f"Selected piece: {self.board.selected_piece} at ({row}, {col})")
                    else:
                        print(f"Cannot select piece at ({row}, {col})")

        def run(self):
            """メインゲームループ"""
            running = True
            while running:
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        running = False
                    elif event.type == pygame.MOUSEBUTTONDOWN:
                        if event.button == 1:  # 左クリック
                            self.handle_click(event.pos)

                # 描画
                self.screen.fill(WHITE)
                self.draw_board()
                self.draw_pieces()
                self.draw_info()

                pygame.display.flip()
                self.clock.tick(FPS)

            pygame.quit()
            sys.exit()

# メイン実行
    if __name__ == "__main__":
        game = ChessGame()
        game.run()
```