Veselin Metodiev

**COS 340 – Programming in Python**

**Prof. Metodi Traykov**

**14.04.2021**

**Project – Chess Game**

1. Description

I have always dreamed of building my own chess game. So, I decided to build such game as a project for the COS 340 course. It is a simplified version of chess and it does not include the rules of an-passant, check and checkmate. The game ends when one of the kings is captured or one side loses all their pieces. When a pawn reaches the last vertical, it is automatically promoted to a queen. In the current version of the program the user cannot choose which piece to promote the pawn. The user can set the initial position of the chess pieces and to choose which pieces to play the game from a text file called start.txt. When the game is over, the moves played are written in a text file called games.txt. The GUI is done with the pygame module. When the user clicks on a piece, the valid moves of the piece are displayed on the screen.

1. The project contains the following classes:
2. Board – this class contains the two-dimensional board. It draws the board, the pieces as well as it controls the position and the movement of the pieces on the board.
3. Game – this class updates the board, tracks which piece the user has clicked and where it is supposed to be moved. And acts as an interface between the main class and the rest of the classes.
4. Piece – contains the information about each pieces. King, Queen, Knight, Rook and Bishop classes inherits from pieces. Each class has a method that finds the valid moves of each clicked pieces.
5. Main – the startUp class of the program. Contains the game loop.
6. Description of special functions and/or algorithms used:
7. #determines whether or not we should move sth
8. def select(self, row, col):
9. if self.selected:
10. result = self.\_move(row, col)
11. if not result: #invalid move - reselect
12. self.selected = None
13. self.valid\_moves = {}
14. self.select(row, col) #try to reselect different piece
15. else:
16. piece = self.board.get\_piece(row, col)
17. if piece != 0 and piece.color == self.turn:
18. #select the piece if it's your turn
19. self.selected = piece
20. self.valid\_moves = self.board.get\_valid\_moves(piece)
21. return True #selection was valid
23. return False #selection is not valid

This method determines whether the user has selected a piece and whether the piece can be moved to the place he has chosen. If it is not valid move, the function is called again. If the piece can go to the place the user wants, the function returns true. Its result is passed to the board class where the move will be made.

 def \_move(self, row, col):

        piece = self.board.get\_piece(row, col) #get the piece

        #print((row, col) in self.valid\_moves)

        if self.selected and (piece == 0 or piece.getColor() != self.turn) and (row, col) in self.valid\_moves: #we can't move to a square that is already occupied by our piece

            if piece != 0 and piece.getColor() != self.turn:

                self.board.remove(piece)

            self.board.move(self.selected, row, col) #move the piece

            self.change\_turn()

            #print(self.board.printBoard())

        else:

            return False

        return True

These methods checks whether the move the user intends to make is valid. The move is made and the method for changing turns is called so that the other side will be able to make their move. Also, if the move is valid and the piece will capture a rival’s piece, the rival’s piece will be removed from the board.

def remove(self, piece):

        if piece.color == BLACK:

            self.black\_pieces\_left -= 1

            if(isinstance(piece, King)):

                self.black\_pieces\_left = 0

                print(self.black\_pieces\_left)

        else:

            self.white\_pieces\_left -= 1

            if(isinstance(piece, King)):

                self.white\_pieces\_left = 0

                print(self.white\_pieces\_left)

        self.board[piece.row][piece.col] = 0

Removes the captured piece from the board and decrements the number of left pieces from the color. If the king is captured, the number of the left pieces is set to 0, signaling that the game is over.

 def move(self, piece, row, col):

        #delete the piece from where it is and move its position

        self.board[piece.row][piece.col], self.board[row][col] = self.board[row][col], self.board[piece.row][piece.col] #swap positions

        piece.move(row, col)

        if isinstance(piece, King): #for castling

            if piece.getRow() == 7 and col == 1:

                print("short")

                if self.board[7][0].moved == False:

                    self.board[7][0].move(7,2)

                    self.board[7][0], self.board[7][2] = self.board[7][2], self.board[7][0] #swap positions

            elif piece.getRow() == 7 and col == 5:

                print("long")

                if self.board[7][7].moved == False:

                    self.board[7][7].move(7,4)

                    self.board[7][7], self.board[7][4] = self.board[7][4], self.board[7][7] #swap positions

                piece.moved = True

            if piece.getRow() == 0 and col == 1: #white

                print("short")

                if self.board[0][0].moved == False:

                    self.board[0][0].move(0,2)

                    self.board[0][0], self.board[0][2] = self.board[0][2], self.board[0][0] #swap positions

            elif piece.getRow() == 0 and col == 5:

                print("long")

                if self.board[0][7].moved == False:

                    self.board[0][7].move(0,4)

                    self.board[0][7], self.board[0][4] = self.board[0][4], self.board[0][7] #swap positions

                piece.moved = True

        if isinstance(piece, Rook) or isinstance(piece, King): #for castling

            piece.moved = True

        letter = ord('h') - col

        if(self.numberOfMoves % 2 == 0):

            moving = str((self.numberOfMoves // 2) + 1) + "." + str(piece) + chr(letter) + str(row+1) + " "

        else:

            moving = ", " + str(piece) + chr(letter) + str(row+1) + "\n"

        self.notation += moving

        self.numberOfMoves += 1

This method changes the values of the selected piece row and column and switches them to the row and column of the place where the user intends to move the selected piece. Then, we check whether the selected piece is King and whether the user wants to castle short or long. We get which king is selected and we also implement the movement of the rook if the user intends to castle. Both long and short castles for the two sides is implemented. When the king, or a rook is moved, its moved value is changed to True. If they are moved, it is impossible to castle.

Finally, we have an algorithm that keeps the moves in a string variable that will be later transferred to the text file. We get the letter by subtracting the ascii value of ‘h’ with the column number. In this way, we can easily get on which vertical the piece is standing. Since the list starts from zero, we have to add 1 before concatenating the horizontal to the notation variable.

def get\_random\_move(board, color, game):

    move = None

    pieces = board.get\_all\_pieces(color)

    rand = random.randint(0, len(pieces)-1)

    randomPiece = pieces[rand]

    valid\_moves = board.get\_valid\_moves(randomPiece)

    while len(valid\_moves) == 0:  #sometimes it chooses a piece that does not have valid moves

        pieces = board.get\_all\_pieces(color) #so choose a different piece until a piece that has valid moves is chosen

        rand = random.randint(0, len(pieces)-1)

        randomPiece = pieces[rand]

        valid\_moves = board.get\_valid\_moves(randomPiece)

    rand = random.randint(0, len(valid\_moves)-1)

    randMove = valid\_moves[rand]

    draw\_moves(game, board, randomPiece)

    temp\_piece = board.get\_piece(randomPiece.row, randomPiece.col)

    if game.winner() != "BLACK":

        move = make\_move(temp\_piece, randMove, board, game)

    return move

This method takes all of the pieces on the board that the computer uses, then it chooses a random piece, then if this piece has valid moves, a random valid\_move is chosen and it is return so that it is passed to the move method that will move the piece.

4. **Instructions how to start and work with the program.**

1) choose the pieces and write their starting position on the first horizontal from the start.txt file.

2) Run the program from the main class from Visual Studio.

3) Choose the piece you want to move, then choose one of the valid squares.

4) Continue until one of the kings is taken or all the pieces are captured.

5) Press any key for a new game or close the program.