Project Report

Chess

Etkin Pınar
19980103-T130
Mert Saraç
19980107-T078

IT366G: Object-Oriented Programming
Spring 2019

Table of Contents

1 Project Description

1.1 Project Overview

The game that we have designed is basically chess game. It has the same rules and same goal with chess. Game is played with two players with a game board and chess pieces. There are two sides (two colors in other word) in the game, one color for one player. Players can only move pieces if their color is same as the pieces. Player with the white color gets to play first and after moving one piece his/her turn is over and his/her opponent takes his/her turn and this goes on like this until a player reaches the goal which is defeating opponent's king.

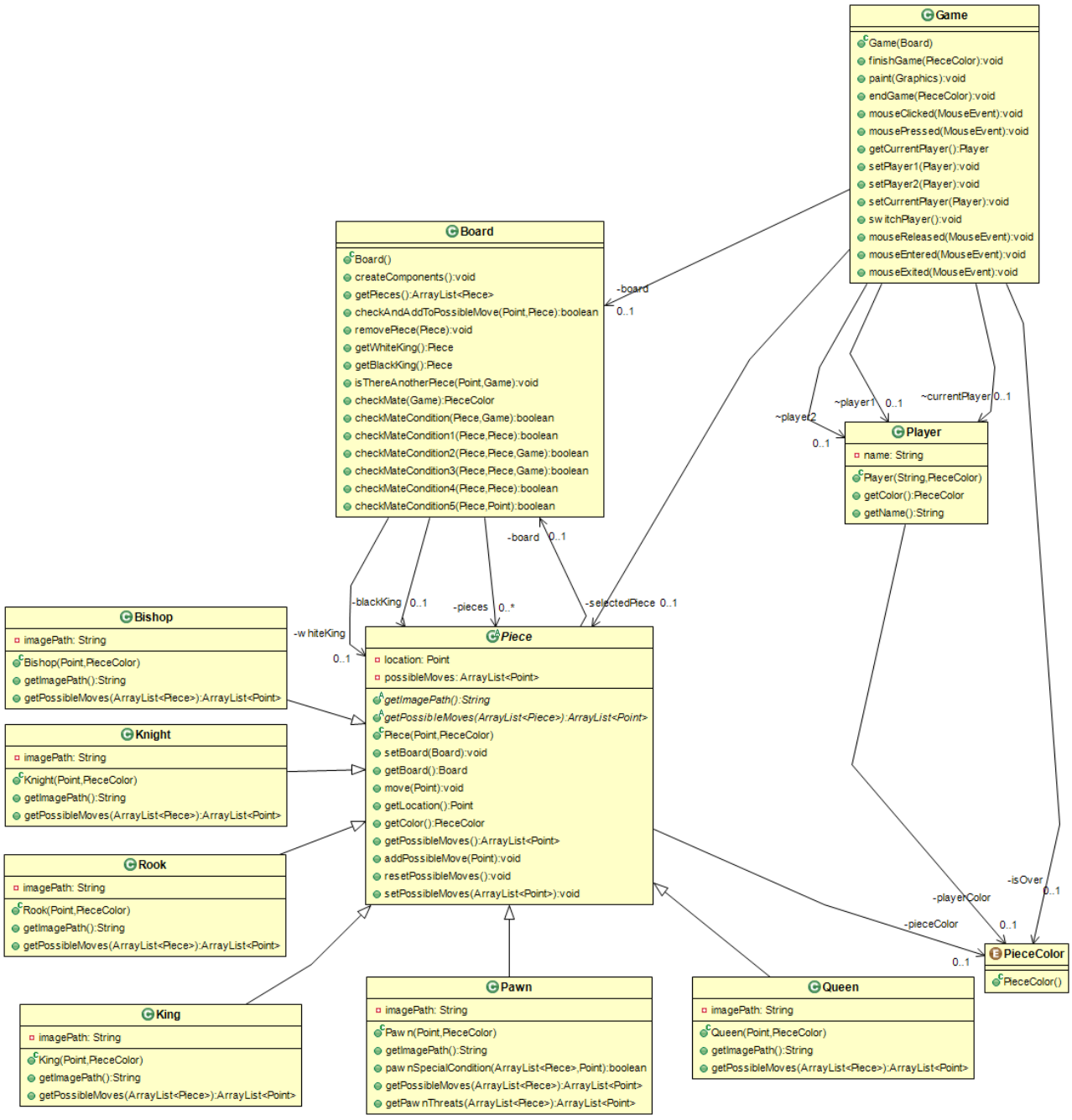
2 Project Requirements

2.1 Functional Requirements

Players can move their chess pieces. Pawns can move two tiles forward if it is their first move, one tile forward if it is not their first move and one tile diagonally forward if there is an enemy chess piece and capture it. Pawns cannot move backward. Rooks can move any number of tiles vertically or horizontally as long as there is no other piece on their way. Bishops can move any number of tiles diagonally as long as there is no other piece on their way. Knights can move in an L shape like pattern, unlike other pieces knights are able to jump over other pieces without any distinction between their color. Queens can move in any direction and any number of tiles. Kings also can move in any direction but only one tile per move. If player clicks on a chess piece of their own in their turn, possible moveable tiles for that piece should be indicated in different color (yellow in our case) so that player can see that piece's possible moves. Players should not be able to move enemy's pieces and should not be able to move their pieces in the other player's turn. A piece should be removed when an enemy piece capture it which means if one piece moves to a tile that an enemy piece is on then that piece is considered as captured and should be removed from the game board. Game should end when a king of a color has no place to escape.

Identifier	Description
Moving a piece	When a player click on a chess piece of their own, some tiles become yellow to show which tiles it can be moved. In order to do this, game class listens to mouse click and takes the location if event is occured. After the click, it checks whether there is a chess piece or not. If there is then it get its possible move set. Each chess piece has their own class and in the class, they have their raw move set which is independent from other chess pieces on the board but we also have to check other pieces so we call checkAndAddPossibleMoves() function from the board class and input raw moves to the function. This function basically checks other pieces and add the move to possible move set if it is possible to take that move. After the first click, if player clicks on the yellow tiles, it moves to that tile.
Capturing a piece	When there is an enemy piece on the possible move tile, the player can click on that tile and capture (defeat) that piece. Player's piece moves to that tile and captured piece is removed from the board. In order to do this, when player clicks on a possible move tile, mouse listener in the game class calls a function named isThereAnotherPiece() from board class and this function checks if there is a piece on that tile and if there is it removes that piece from the game board (piece list).
Turn Control	Players should not be able to move pieces other than their color. In order to do this, there is a variable named currentPlayer in the game class. When a player clicked on a chess piece, mouse listener first controls if the color of the piece is same as the current player's color before showing possible moves.
Turn Switch	Players should be able to move only one piece and after moving it, their turn should end. In order to do this, after moving a piece, a function named switchPlayer() is called and it reassigns the currentPlayer variable.
Checkmate Control	Game should end if one player's king is threatened with capture and cannot escape from that situation. In order to check this situtation, we have a function called checkMate() in the board class. This function calls a function named checkMateCondition() which checks if there is a piece which threats the king. If there is then it calls checkMateCondition2() and this checks if there is any free frame that king can go if it fails then it calls checkMateCondition5() and it checks if king can beat that piece and be safe after moving to that position. If this also fails, it calls checkMateCondition3() which looks for a friendly piece that can defeat the threatening piece if this also fails we call the last function checkMateCondition4() to look for a piece that can block the enemy piece by sacrificing itself if this also fails then that means there is no way to escape and this player lose the game.

3 Project Design



4 Project Implementation

4.1 User Interfaces

Since there is not any screen other than actual game board in most of chess games, we also do not have much additional screens. Only screen we have other than actual game is a small window that asks players their names.

4.2 Code Illustrations

```
25      @Override
26      public ArrayList<Point> getPossibleMoves(ArrayList<Piece> pieces) {
27
28          resetPossibleMoves();
29          ArrayList<Point> possibleMoves = getPossibleMoves();
30          //Break the loop if there is a piece that block possible moves.
31
32          Point tempLocation = getLocation();
33
34          boolean flag = true;
35          //(x--, y--)
36          while(tempLocation.x > 1 && tempLocation.y > 1 && flag) {
37              tempLocation = new Point(tempLocation.x - 1, tempLocation.y - 1);
38              flag = getBoard().checkAndAddToPossibleMove(tempLocation, this);
39          }
40
41          //reset the current location
42          tempLocation = getLocation();
43          //reset the flag
44          flag = true;
45          //(x++, y++)
46          while(tempLocation.x < 8 && tempLocation.y < 8 && flag) {
47              tempLocation = new Point(tempLocation.x + 1, tempLocation.y + 1);
48              flag = getBoard().checkAndAddToPossibleMove(tempLocation, this);
49          }
```

Each chess piece has its own class and extends the abstract class Piece and from that parent class they have some common methods and one of them is getPossibleMoves(). This method specifies raw move set for the chess piece and one of the most used method in the programme. The code we have above is from Bishop class since bishop can move diagonally we need to indicate all 4 direction it can move. In each direction it takes one tile and send it to checkAndAddToPossibleMove() method to check other pieces on that tile. If there is not, then it adds it to the possible move set. Since bishop can move as long as there is not a piece on the way, it is written in a while loop and flag variable is used to break the loop if there is a piece that block the possible moves.

```
176     public void mousePressed(MouseEvent e) {
177         //get the player's click as long as the game continues
178         if(isOver == null) {
179             int x, y;
180             double ex, ey;
181             //if there is selectedPiece but the chosen frame is out of the possible moves
182             boolean flag = false;
183             //assign the coordinate of frames which are chosen by the players
184             ex = e.getPoint().getX();
185             ey = e.getPoint().getY();
186 
187             //if there is a piece which is chosen
188             if(selectedPiece != null) {
189                 for(Point cell : selectedPiece.getPossibleMoves(board.getPieces())) {
190                     x = cell.x * Game.n;
191                     y = cell.y * Game.n;
192                     //check if the click is on the frame of possible moves
193                     if(ex < x && ex > (x - Game.n) && ey < y && ey > (y - Game.n)) {
194                         //check if there is an enemy piece at the moving point.
195                         board.isThereAnotherPiece(new Point(cell.x, cell.y), this);
196                         //move the selected piece on the chosen frame
197                         selectedPiece.move(new Point(cell.x, cell.y));
198                         //make this null in order to stop showing possible moves
199                         selectedPiece = null;
200                         flag = true;
201                         //check for check-mate
202                         if(isOver == null) {
203                             isOver = board.checkMate(this);
204                             if(isOver != null) {
205                                 repaint();
206                                 endGame(isOver);
207                             }}switchPlayer();}}}
208 
209             //if there is no selected piece or player choose another piece
210             if(!flag) {
211                 selectedPiece = null;
212                 //check all pieces
213                 for(Piece piece : board.getPieces()) {
214                     if(piece.getColor() == currentPlayer.getColor()) {
215 
216                         x = piece.getLocation().x * Game.n;
217                         y = piece.getLocation().y * Game.n;
218 
219                         //if this piece is on the chosen frame
220                         if(ex < x && ex > (x - Game.n) && ey < y && ey > (y - Game.n)) {
221                             selectedPiece = piece;
222                         }}}}
223             if(possibleCheckMate != null && possibleCheckMate.equals(currentPlayer.getColor())) {
224                 if(possibleCheckMate.equals(PieceColor.WHITE))
225                     selectedPiece = board.getWhiteKing();
226                 else
227                     selectedPiece = board.getBlackKing();
228                 flag = true;
229             }repaint();}}
```

Game is played with only mouse. Therefore, mouse listener does a lot of work. When player clicked on a position, it takes the location of that click and saves its coordination to ex and ey variables. After that it search the board and looks for a piece that is on that location if there is a piece and that piece has the same color as current player, then it assigns that piece to the selectedPiece variable and repaint the board to show tiles that is possible to move for that piece. If player clicks one more time, this time we enter the first if condition since we have already chosen a piece. Code takes location of this click as well and check if it is on the possible move tile. If it is, it calls isThereAnotherPiece() method from Board class to check if there is a enemy piece on that tile and finally it moves to the chosen tile. We also do some controls if the move ends the game or not.

```
164     //if there is any enemy piece that threats the king
165     public boolean checkMateCondition(Piece king, Game game) {
166         game.setPossibleCheckMate(null);
167         for(Piece piece : pieces) {
168             if(piece.getColor() != king.getColor()) {
169                 for(Point point : piece.getPossibleMoves(pieces)) {
170                     if(point.x == king.getLocation().x && point.y == king.getLocation().y) {
171 
172                         if(checkMateCondition2(king, piece, game)) {
173                             return true;
174                         }}}}}
175         return false;
176     }
```

One of the most important and hardest method to implement is checkMateCondition() methods. The code that is above is just one of the 6 conditions we have to check checkmate. This one controls if there is an enemy piece that threatens the king, checkMateCondition1() controls if there is a friendly piece that can block the way of the enemy piece, checkMateCondition2() controls if there is a possible tile that is safe to move for the king, checkMateCondition3() controls if there is a friendly piece that can capture the enemy piece, checkMateCondition4() controls if king itself can capture the enemy piece and checkMateCondition5() checks if it is safe to capture the enemy with the king because in some cases if king captures a piece, there is another piece that can capture it. This method is the most troublesome part of the game. We first tried to implement this algorithm we did not think that would be hard but there were too many possible movement patterns and we even now do not know whether we covered all checkmates possibilities or not.

5 Future Work

There can be several things to add this game. Some kind of timer can be added to the game to make the competition a bit more professional and we can add an option to choose whether play with timer or without timer. There are some kind of special moves in chess. We were not able to add them to the game, so those movements can be added to the game after developing a working algorithm. Since chess does not require so much hardware power, we did not pay attention enough to the performance loss. Therefore, in the future, game can be optimized better since some algorithms work very bad in terms of performance. There are few bugs in the game which we could not find solutions, they can be fixed.