ADVANCED PROGRAMMING TECHNIQUES

CSC 746 PROJECT REPORT

HAMZAT ZAYNAB ADEOLA

185852

AUGUST, 2021

**CHAPTER ONE**

**1.0 INTRODUCTION**

Chess is a recreational and competitive board game played between two players. It is one of the world’s most popular games, played by people worldwide. Chess is an abstract strategy game and involves no hidden information. It is played on a square chessboard with 64 squares arranged in an eight by eight grid. At the start, each player (one controlling the white pieces, the other controlling the black pieces) controls sixteen pieces: one king, one queen, two rooks, two knights, two bishops and eight pawns. The object of the game is to checkmate the opponent’s king, whereby the king is under immediate attack (in check) and there is no way to escape it to escape. There are also several ways a game can end in a draw.

**PROJECT DESCRIPTION**

For my project, I chose to design a Java application that allows two users to play chess (specifically) using the command prompt. It is entirely programmed in the Java programming language. This project aims to serve as the go to command line chess application for windows user. As more and more windows users are becoming power users, a light-weight CLI chess application provides great convenience to access chess puzzles which hopefully enables chess enthusiasts to practice more often and improve their tactical abilities.

I originally wrote this project in python and had to rewrite it in Java for this course, so it is still a work in progress.

**1.1 CLASS DESCRIPTION**

The chess game is designed by using multiple classes that work together to form the game. Following is a description of each class as well as some of the more important class methods. As the classes are described, the logic used to design the chess game as a whole will become apparent.

Class Piece

Class Piece is used to define each of the 32 pieces used in a chess game. Each Piece is constructed in one of two ways: empty, or with a color, type, and row and column coordinate specified as given parameters. The different piece types are programmed into Class Piece as constants and are as follows:

• King

• Queen

• Rook

• Knight

• Bishop

• Pawn

The methods in Piece are used to access its data and other classes utilize these methods to view and change the data in a certain Piece. One method unique to Piece is getPieceName(), which returns the piece's shortened board name for display on the text based game in the terminal. This function is used by the ChessBoard class.

Class Main

This is the chess game's top module, which allows the user to actually play the game.

Primarily this class is programmed in the main method. This is because my game is based in the command line and so all of the user's inputs have to go through the terminal. This class has nothing too special in it except for compiling everything together to play the game!

Class Board

Class Board holds the information for the chess board in each game. The board is designed to be completely dynamic, being updated each time a player makes a move. The board itself is an 8x8 Piece array. To create the board, **ChessBoard’s default constructor** was used. I implemented all of the Square objects and Piece objects in default constructor. The board has 64 Square objects and some of them have Piece objects.

To print the board, **ChessBoard’s toString method** was overridden. I created a string called **brdStr**, and I implemented the board to brdStr by using for loops. How to print all elements on the board was a challenge. I used *nested for loops* to reach each element of the board.

Defining Board and Square Classes

I created a multi dimensional Square array using board2d reference from Square class in ChessBoard. I declared the board2d as public because I needed to reach it from other classes. board2d object is a multi dimensional Square array therefore I created Square objects for each field of board2d in ChessBoard default constructor. Also I created pieces in ChessBoard default constructor.

**Implementing methods of Board and Square Classes**

**Methods of Board (ChessBoard Class)**

● public voiddecrementPiece(intcolor)

> Decrements the number of pieces according to color. If color equals 0, decrements the

number of white pieces, if not decrements the number of black pieces.

> color: color of piece which will be decremented.

> It doesn't return anything

> It checks color, if it is white method decrements the number of white pieces, else it

decrements the number of black pieces..

● publicString toString()

> It is holds brdStr variable which have board2d’s string form.

> Variable: -

> It returns a String form of board.

> It uses for loops to reach each element of the board.

● public booleanisGameEnded()

> It checks if the game ended or not. .

> Variable: -

> It returns true if the game is ended, if not it returns false.

> It checks the number of pieces, if one of them is zero it returns true. Also it controls there

are 2 kings, if there is 1 king it returns true again.

● public booleanisWhitePlaying()

> It checks whose turn.

> Variable: -

> It returns true if white is playing, else it returns false.

> It simply returns isWhite variable which controls with nextPlayer().

● publicPiece getPieceAt(String from)

> It returns a Piece object according to its parameter .

> “from” is a string which represents our selected piece’s location.

> It returns a piece object of given coordinates.

● public int[] getCoordinates(String from)

> It converts string to an integer array which have coordinates.

> “from” is a string which represents our selected piece’s coordinates.

> It returns an integer array.

> I used charAt() method to reach characters in the string. And then, I converted them

integer via using a *Character:Integer Hasthtable*.

● private voidcreateHashtable()

> It puts characters and integers which are on the board to hashtable.

> Variable: -

>Return: -

● publicSquare getSquareAt(String to)

> Returns the Square object on the given coordinates.

> “to” is a string which represents our target square’s coordinates.

> It returns a Square object..

> It uses getCoordinates method to convert string “to” to the coordinates. And then, it returns

square according to these coordinates from the board2d.

● publicSquare[] getSquaresBetween(Square location, Square targetLocation)

> Returns the squares between given squares if they are at the same row or same column or same diagonal

>location: the beginning square, targetLocation: the final square

> Array of squares as the same order from location to targetLocation square **not including location**. Null is returned if no squares are in between

> First, it controls given squares are at the same row or same column or same diagonal. If

they are, It takes coordinates of given squares and it calculates other squares between

location and targetLocation, then creates a square array and implements squares between

location and targetLocation to this array. Finally it reverses the array and returns.

● public voidnextPlayer()

> Changes isWhite value. If it is true, nextPlayer() makes it false. Else true.

● publicSquare[] getSquaresBetweenForKnight(Square location, Square

targetLocation)

> Returns the squares between given squares.

> location: the beginning square, targetLocation: the final square

> It returns an array of squares. The array only contains two squares targetLocation and

other square which can prevent Knight's jump. (Knight can only jump over 1 unit)

>Firstly, it creates a Square array which has size 2. And then adds squares to this array. It

compares targetLocation’s coordinates and location’s coordinates, thanks to comparison it

determines location of other square which can prevent moving of Knight.

**Methods of Square Class**

● publicString toString()

> Returns “ “ if piece in square null else returns piece’s toString() method.

● public booleanisAtSameColumn(Square targetLocation)

> It checks targetLocation and this square object is in the same column. If it is, it returns true

else false.

> targetLocation: location which we want to go.

> It returns boolean value, returns true If targetLocation’s column equals square’s column

else returns false.

>It simply checks if the square’s column and targetLocation’s column same.

● public booleanisAtSameDiagonal(Square targetLocation)

> It checks targetLocation and this square object is in the same diagonal. If it is, it returns

true else false.

> targetLocation: location which we want to go.

> It returns boolean value.Returns **true** If difference between targetLocation’s and square’s

row is equal to difference between targetLocation’s and square’s column else returns **false**.

> It simply checks if the square and targetLocation are at the same diagonal via using

difference of rows and columns.

● public booleanisAtSameRow(Square targetLocation)

> It works with similar to the isAtSameColumn. It checks targetLocation and square are in

the same row. If they are, returns true, else returns false.

> targetLocation: location which we want to go.

> It returns boolean value, returns true If targetLocation’s row equals square’s rowelse

returns false.

>It simply checks if the square’s row and targetLocation’s row same.

● public booleanisEmpty()

> It checks square is empty.

> Variable: -

> It returns boolean value. When square is empty returns true, in the other case returns

false.

> It simply checks piece’s value. If it is null, this means square is empty. Else square is not

empty.

● public booleanisNeighbourColumn(Square targetLocation)

> It checks targetLocation is in the neighbour column of square.

> targetLocation: location which we want to go.

> It returns boolean value. When targetLocation is in the neighbour column of square

returns true, in the other case it returns false.

> It checks difference of columns of square’s and targetLocation’s. If it is 1, this means their

columns are neighbours.

● public booleanisAtLastRow(int color)

> Checks whether the row is the last row. It uses color to learn is it last row for which color

pieces.

> color: It is an integer. Color is white for 0 and it is black for 1.

> It returns true if the row is the last row, in the other case it returns false. It compares color

and ChessBoard.WHITE static variables.

> It compares given color and ChessBoard.WHITE.

If the given color is **white**, it checks that the square's row is equal to 0. And isAtLastRow()

returns value of this.row == 0

If the given color is not equal to ChessBoard.WHITE, this means color is **black**. And

color == ChessBoard.WHITE is **false.** If it is false, it checks that the square’s row is equal to

7. And isAtLastRow() returns the value of this.row == 7.

The remaining classes are the various pieces class.

1.2 DEFINING PIECE HIERARCHY

The Piece class is an abstract class because actually there is no piece object in the board.On the board, there are Pawn, Bishop, King… objects. Pawn, Bishop, King and other objectsalso are a piece of chess game. Also, it is a superclass of Pawn, Bishop, King, Knight, Queen and Rook classes. It is a parent class of these classes because all of these classes have similar methods, for example move() method is the same for almost all piece classes.Piece class has location and color. Since, Piece has location and color attributes subclasses of Piece inherit these attributes.

public int getColor() method is not abstract because all of the pieces have color attributes. Instead of creating a getColor() method for each piece, creating one and using it is easier because subclasses can reach getColor() method which is in the parent class.

public abstract boolean canMove(String to) is abstract because all of the pieces have different movement rules. We can’t control them with one canMove method.

public void move(String to) is not abstract because all of the pieces can move with one move method after checking they can move.

protected boolean isEnemy(Piece p) is not abstract because it controls only color of piece and all pieces have color attributes so that they can use it. It is “protected” because it is used only by piece objects like Pawn, Knight…

In the other hand, other methods are “public” because Main class needs to access these methods, they must be defined as “public”.

Implementing methods in Piece Hierarchy

● public abstract boolean canMove(String to);

> It checks piece can move to String to. Each piece have this method so that it is abstract.

> to: location which we want to go. It is string.

> It returns boolean value for each piece in their classes.

> It simply checks piece can move to given location

Pawn’s Methods:

● public boolean canMove(String to)

> It checks targetLocation is at the same column with initial location, then it checks row distance between initial location and targetLocation. If row distance is 2 and color is white, this means white Pawn is moving twice. Squares between target and initial locations must be empty, it checks if they are empty or not. If they are empty it assigns true to validMove.

If targetLocation and initial location are not in the same column, Pawn can move only if is attacking diagonal.It checks again row distance and color, and assigns true target is not empty and targetLocation’s piece’s color is different from pawn.

Finally, if given scopes doesn’t run, canMove returns false.

● public String toString()

> It returns “P” if color is white (0), or it returns “p” if color is black (1).

Rook’s Methods:

● public boolean canMove(String to)

> It checks targetLocation and initial location are at the same column or same row. If they are not, it returns false. Because a rook cannot move diagonal, it can move only vertical or horizontal. Then, it checks if all of the squares between target are empty and it assigns boolean value of isEmpty() to validMove. Finally, it returns validMove.

● public String toString()

> It returns “R” if color is white (0), or it returns “r” if color is black (1).

King’s Methods:

● public boolean canMove(String to)

> King can move everywhere but only 1 unit. So, it checks the first target location is in the 1 unit away and it is empty and returns the boolean value of this control. If validMove is false, and target location is in the 1 unit away, it checks there is an enemy in target location. If there is an enemy, it makes validMove true and then it returns validMove.

● public String toString()

> It returns “K” if color is white (0), or it returns “k” if color is black (1).

Queen’s Methods:

● public boolean canMove(String to)

> Queen can move everywhere if the target is in the diagonal, vertical or horizontal. So that, firstly it controls the target is in the same diagonal, same row or same column. If it is not, it returns false. If target is in the same diagonal, same row or same column it checks all squares between target and initial location and assigns boolean value of this control to validMove. After this control, if validMove becomes false, It checks there is an enemy at the target location. If there is, it makes validMove true. And finally, it returns validMove.

● public String toString()

> It returns “Q” if color is white (0), or it returns “q” if color is black (1).

Bishop’s Methods:

● public boolean canMove(String to)

> Bishop can move only diagonal. Therefore, method checks if the target is at the diagonal. If it isn’t, it returns false. If target is at the same diagonal, it checks all squares between initial location and target are empty and then it assigns boolean value of this control to validMove.

If after then this control, validMove is false it checks if there is an enemy at the target location. If there is an enemy at the target location, it assigns true to validMove. And finally, method returns validMove.

● public String toString()

> It returns “B” if color is white (0), or it returns “b” if color is black (1).

Knight’s Methods:

● public boolean canMove(String to)

> Knight can move only by drawing a “L” shape. And the distance between target location and initial location must be 2.23607 units. So that method first calculates coordinates of target location and initial location. And then it measures distance between target and initial location. Secondly, it converts this value to DecimalFormat and then assigns it to a string. It checks the distance between initial location and target location is equal to “2.24” (2.23607’s decimal format which is converted to string) If it is not equal, it returns false. If it is equal, it controls only the target and the other square (that the Knight can’t jump on it) are empty. If these squares are empty, It assigns a boolean value of isEmpty() to validMove for each control. If validMove becomes false after this checking process, it checks if there is an enemy at the target location. If it finds an enemy at the target location, assigns true to validMove. Finally it returns validMove.

● public String toString()

> It returns “N” if color is white (0), or it returns “n” if color is black (1).

1.3 PLAYING THE GAME

Playing the chess game is very straightforward. Users only need to have all of the classes in the same folder and only have to compile and run Main in the terminal to play the game.

Game play is just like chess games of old: the white player starts off the game and then players take turns making moves until a checkmate is achieved!

1.4 TESTING THE GAME / ANALYSIS

The most amusing and tedious aspect of coding the game was testing each method. It was tedious because there are so many methods to check. It was amusing because of how often

I found myself having to add in methods because I had previously missed something. The game play itself is pretty efficient; ArrayLists work very well in keeping track of dynamic lists and almost all of the lists used in chess are dynamic. There are some aspects of the game that I could not include because I could not figure out how to efficiently program their methods.

1.5 CONCLUSION

In this project, I designed a chess application in the Java programming language. While the application is not complete quite yet, the main function of playing chess is present and the application is open to future improvements. Potential improvements include a GUI, finishing the saving/loading system, interfacing with the cloud to play against someone remotely using

Hadoop or something similar, etc. This project certainly tested my ability to use all aspects of the Java programming that was taught in this class and truly helped solidify my ability to visualize many aspects of the language. I have always loved the game of chess; now I have an even greater love for the game because I now know how much actually goes into the game.