

Alquerque

Danny Nicolai Larsen, Mikkel Brix Nielsen & Steffen Bach

November 17, 2021

CONTENTS

Contents

1	Introduction	3
2	Design	3
2.1	Option menu & game initialization	3
2.2	Programming methodology	4
2.3	Getters & setters:	4
2.4	Chess notation	4
3	Implementation	5
3.1	The init() method	5
3.2	The main game loop	6
3.3	Construction of the visual board	6
3.3.1	Chess notation and functionality	7
4	Test	9
4.1	Error when inputting non-integers in menus	9
4.2	Mistake in B/W input, fixed with regex	9
4.3	CPU moves being printed incorrectly	10
5	Conclusion	10
6	Appendix	10
6.1	Program Code	10

1 Introduction

For phase 1 of the project, we have been tasked with implementing the user interface for the board game, Alquerque, by developing a class in accordance with the contract for phase 1. The class has to be executable, meaning it has a main method. The program must start with prompting for the choice of which players are human and which are controlled by the computer. When playing the game, it must prompt the user for a move, and output the moves the computer makes. After each turn it must print the board to the screen. Game continues until a winner is found or there are no valid moves left, resulting in a draw. All the provider classes are precompiled, and thus we should just focus on the interface during this phase.

2 Design

This section will give an overview of how the program works, and which design choices has been made while writing the program.

2.1 Option menu & game initialization

The program works by first asking the user to select one of three playable game scenarios; (1) Player vs Player, (2) Player vs CPU, and (3) CPU vs CPU, or (0) to exit the program. The game is then initiated according to what the user picked.

If the user chooses to play against another player, they will be prompted to enter the name of player 1 and then player 2. The game will thereafter commence with player 1 being white and player 2 being black. The game will continue, first prompting the current player, starting with white, to input which piece they want to move followed by where they want to move it, updating the position of the pieces and displaying the updated gamestate to the user, then switching to the next player and repeating the same process for that player. This continues until one of three things happen, either black wins, white wins or the game is a draw.

Else, if the player chooses to play against a CPU they are first prompted to pick what color they want to play as. The CPU will then play as the opposite color. The player is then prompted to enter their name followed by the number of moves they want the CPU to look ahead while calculating its

2 DESIGN

moves, which determines the level of difficulty. The game will then commence with white making the opening move, may that be either the player or the CPU. Assuming the CPU is white it will calculate its move, in accordance with the number of moves it was allowed to look ahead by the player, and play it. The game state is then updated and displayed, then switching to the player, saying that it is their turn, prompting them to input which piece they want to move followed by where they want to move it. The pieces position are updated and the updated gamestate is displayed, then the CPU makes a move and this cycle continues until either black wins, white wins or the game is a draw.

Alternatively, the user can choose to pit two CPUs against each other, picking how many moves the CPUs are allowed to look ahead, being able to spectate their respective moves as they are calculated and executed. Updating the pieces positions and then displaying them to the user. This continues until either black wins, white wins or the game is a draw.

2.2 Programming methodology

The program is written in a top-down fashion, by constructing a main method, which included all the methods we would need to make the program function as we had intended. After constructing the main, we created the individual methods for printing the board, printing the options menu, converting coordinates from chess notation to positional integers, etc.

2.3 Getters & setters:

Getters and setters have not been implemented because the class is not going to be instantiated, and it is not defined in the contract that another class should be able to use these variables.

2.4 Chess notation

For improved QOL/user friendliness, we have chosen to assign coordinates to the visual representation of the board with letters, A through E, above and below the board, as well as numbers, 1 through 5, to the left and to the right of the board. Due to the familiarity from chess, this way of identifying board positions should feel more intuitive.

3 IMPLEMENTATION

Figure 1: Chess Notation

	A	B	C	D	E	
1	[B]	- [B]	- [B]	- [B]	- [B]	1
	\	/	\	/		
2	[B]	- [B]	- [B]	- [B]	- [B]	2
	/	\	/	\		
3	[B]	- [B]	- []	- [W]	- [W]	3
	\	/	\	/		
4	[W]	- [W]	- [W]	- [W]	- [W]	4
	/	\	/	\		
5	[W]	- [W]	- [W]	- [W]	- [W]	5
	A	B	C	D	E	

Following this mindset, we decided to represent the board in a two-dimensional array in the size of 6 x 5. The reason we chose this size is that the side of the board which represents the numbers seems more intuitive if the row number corresponds to the rank number on the board, which goes from 1-5. On the other hand the side of the board that represent the letters doesn't have to correspond in the same way, so we kept the column as going from 0-4

3 Implementation

This section describes the actual technical implementation of the choices described in Section 2, about design, going in to detail about what methods have been implemented and how they are used.

3.1 The init() method

The method init() is the first thing being called when the program starts, when this happens all variables get initialized throughout the method. Firstly the variables which have predefined values get initialized e.g. board becomes a new board, reader becomes a new scanner. Secondly the program greets the user, prints the available options the user can pick from and then prompts the user to pick one of the options, which is then passed through a switch, which is wrapped in a do-while loop, which repeats until a valid option is picked. The switch determines which of the available options, if any, the user's input corresponds to. If the user's input does not correspond to a valid option the

3 IMPLEMENTATION

switch defaults to telling the users that their inputted option is an invalid option and to try again. This response is followed by the option menu being printed for the user to view the available options again. Depending on which option is chosen, the corresponding variables are defined, and the loop within `init()` is exited, since a valid option was picked, and the program continues to the main game loop.

3.2 The main game loop

In the main game loop the first thing that happens is that the initialized board is presented to the user. After this a check is run determining how the next move should be made. If the next one to play is a player they are prompted to choose a piece corresponding to the color of the player, which turn it is, that they want to move, and then where they want to move the piece, in accordance with standard chess notation. This move is then validated as a valid coordinate. An instance of the move class is then created and passed to `isLegal()` to see if it is a legal move that can be played on the board in the current boardstate. And if that is not the case the user is told that the entered move was invalid and a do-while repeatedly asks the user to input what piece they want to move and where to, until a valid and legal move is entered by the user. While the game is not over this process is repeated switching between black and white. If the player is a CPU, a new move is created in accordance with the calculations done by `MiniMaxTree`. If the do-while guard, encapsulating the player's moves, registers whether the game is over. An if-statement then checks whether black has won, white has won or if it is a draw between the two. This is then printed to the console for the player to view.

3.3 Construction of the visual board

The board is represented in the program by a two-dimensional array, which is constructed in the `boardWithPieces()` method. First it initiates the array, and fills it with empty spaces, defined by the `EMPTY` constant. It then uses the methods `black()` and `white()` from the `Board` class, to fill in the spaces occupied by black and white pieces.

The `printBoard()` method, which is used in the main game loop to display the board on the screen, makes an array from `boardWithPieces()` and prints it out, along with letters from A - B in the top and bottom, numbers from

3 IMPLEMENTATION

1-5 in the sides and the guiding lines between the squares.

3.3.1 Chess notation and functionality

As mentioned in the design-section, we chose to represent the board with letters for columns and numbers for rows. However, this gives us a String coordinate rather than an integer in the preconditioned range of 1 through 25. So in order to satisfy the precondition for Move, we made the method `convertCoordinate` that converts an input coordinate to the corresponding positional number, so that it may be used by methods `from()` and `to()` in `Moved`. The method works by assigning a numerical value to the coordinate letters, then adding it to a multiplum by 5, which is determined by the coordinate-numbers corresponding array-index.

```
1  /**
2  * Converts an input coordinate to the corresponding position on the ↵
   board, determined by numbers 1-25
3  * @param coord move coordinate input from user
4  * @return position on board, represented by an integer (1-25)
5  */
6  private static int convertCoordinate(String coord){
7      int position = 0;
8      switch(Character.toUpperCase(coord.charAt(0))){
9          case 'A': //value of each column is added to the row-↵
                  determined multiplum of 5 (e.g. D is 4'th, so positional ↵
                  value is +4)
10             position = (1+(5*((Integer.parseInt(coord.substring(1))-1)↵
                  )));
11             break;
12             case 'B':
13                 position = (2+(5*((Integer.parseInt(coord.substring(1))-1)↵
                  )));
14                 break;
15             case 'C':
16                 position = (3+(5*((Integer.parseInt(coord.substring(1))-1)↵
                  )));
17                 break;
18             case 'D':
19                 position = (4+(5*((Integer.parseInt(coord.substring(1))-1)↵
                  )));
20                 break;
21             case 'E':
22                 position = (5+(5*((Integer.parseInt(coord.substring(1))-1)↵
                  )));
23                 break;
24             default:
25                 return 0;
26         }
27         return position;
28     }
```

3 IMPLEMENTATION

Since `convertCoordinate()` has a precondition that it can only accept coordinates that correspond to a number 1-25, we also wrote a method, `isValidCoords()`, that returns a boolean value for whether the coordinate is a valid position on the board or not. It uses regex to check if it is a string of length 2, and within A1-E5, which, by the logic used in `convertCoordinate()`, will only translate to numbers 1 through 25, thereby satisfying the precondition.

```
1      /**
2       * Test wether an entered coordinate is a valid coordinat
3       * @param coords, a coordinate to be tested
4       * @return true if the coordinat entered is a valid coordinat else↵
5       *         returns false
6       */
7      private static boolean isValidCoords(String coords){
8          return (coords.matches("[A-Ea-e][1-5]")); // Regex for ↵
               matching
9      }
```

For a move to be printed as a coordinate, rather than as the positional integer which the CPU returns when a move is calculated, we had to make a method that would convert Move objects returned from MiniMaxTree to the corresponding letters for files and numbers for ranks. The method functions by subtracting 1 from the positional integer, before subtracting 5 until the integer is between 0 and 4, which for each number corresponds to a letter for each file on the board.

To determine the ranks, the positional integer is divided by 5, where-after 1 is added to compensate for index 0 in the array. However, during testing we discovered a minor issue with this method, although, for the most part, it worked as intended, we found the need to subtract 1 from the position before dividing; this will be elaborated in the test section of this report.

```
1      /**
2       * Converts an input position, represented by a number 1-25 to the ↵
3       * corresponding coordinates in form [A-E][1-5]
4       * @param position position represented by an int
5       * @return coord position represented by coordinates [A-E][1-5]
6       */
7      private static String convertPosition(int position){
8          String coord = "";
9          switch ((position - 1) % 5){
10             case 0:
11                 coord = "A";
12                 break;
13             case 1:
14                 coord = "B";
```


4 TEST

```
14         break;
15     case 2:
16         coord = "C";
17         break;
18     case 3:
19         coord = "D";
20         break;
21     case 4:
22         coord = "E";
23         break;
24     }
25
26     coord = coord + ((position - 1) / 5 + 1);
27     return coord;
28 }
```

4 Test

This section describes some of the errors and challenges we faced during the construction of the program, either from challenges arising from the calculations, or by limitations of the methods used.

4.1 Error when inputting non-integers in menus

The game crashes when a non-integer input is typed in the menu, or when choosing the CPU depth, as the Scanner method `nextInt` is used. This causes the program to throw an `inputMismatchException`.

4.2 Mistake in B/W input, fixed with regex

During testing we found that, when choosing to play white or black, the user could type any characters after the B or W and it would still accept it. For example you could write `Bwhite`, and it would be accepted as black. When this error occurred the program would accept a string as input for what color the player wanted to play. The input string would then be passed to a switch where the character at index 0 would be checked and all characters after index 0 was ignored. To fix this error, we changed it to an if-else statement, using regex to match B and W specifically, but case insensitive.

6 APPENDIX

4.3 CPU moves being printed incorrectly

When the CPU made a move, we discovered that whenever it was to or from the E file, regardless of rank, the move, although correctly executed, would be printed as a rank 1 higher than intended. An example of this would be the move D1 to E1 being misprinted as “White/Black(CPU) moved D1 to E2”. To fix this, we found that in our method to convert a position represented by an integer to a position represented by a coordinate, we would have to subtract 1 from the position integer before dividing by 5 to get the rank. This meant that position 5 would accurately be converted to 1, instead of 2. The same thing applied to 10, 15, 20, and 25.

5 Conclusion

We did not encounter a lot of issues in this phase. We went in to the design phase with a clear picture in our minds, which was a movement system, similar to a chess game, with coordinates instead of numbers, and this proved to be a challenge to implement.

In hindsight this might have made our code a bit more complicated, but we were committed to solving this issue, and implemented several methods to overcome the challenge, without breaching the contract.

We managed to construct a working program without any major issues. The only thing left unsolved is the exception thrown if anything other than an integer is entered when the program expects an integer.

6 Appendix

6.1 Program Code

```
1 import java.util.Scanner;
2 public class Alquerque {
3     private static Scanner reader;
4     private static Board board;
5     private static final char EMPTY = ' ';
6     private static String whiteName, blackName;
7     private static int cpuDepth;
8     private static boolean isWhiteCPU, isBlackCPU, isWhite;
9 }
```

6 APPENDIX

```

10 public static void main(String[] args) {
11     String coordsFrom;
12     String coordsTo;
13     Move nextMove = new Move(0,0);
14     init();
15     do { // main game loop
16         printBoard();
17         if (!isWhiteCPU && isWhite || !isBlackCPU && !isWhite) {
18             boolean inputWithinRange = false;
19             do { // loop for validating the players input
20                 System.out.print("It's " + (isWhite ? whiteName : ↵
21                     blackName) + "'s turn" + ", please enter which " +
22                     "piece you want to move: ");
23                 coordsFrom = reader.nextLine().trim();
24                 System.out.print("Please enter where you want to move ↵
25                     the piece: ");
26                 coordsTo = reader.nextLine().trim();
27                 if (isValidCoords(coordsFrom) && isValidCoords(↵
28                     coordsTo)) { //Checks if input is a valid letter+↵
29                     number
30                     nextMove = new Move(convertCoordinate(coordsFrom),↵
31                         convertCoordinate(coordsTo)); //Converts ↵
32                         coordinate to int position
33                     if (board.isLegal(nextMove))
34                         inputWithinRange = true;
35                 }
36                 if (!inputWithinRange)
37                     System.out.println(coordsFrom + " to " + coordsTo ↵
38                         + " is " +
39                         "not a valid move, please enter a ↵
40                         coordinate A-E 1-5.");
41             } while (!inputWithinRange);
42             board.move(nextMove);
43         } else if (!board.isGameOver()) {
44             nextMove = new Minimax().nextMove(board, cpuDepth, isWhite↵
45                 );
46             System.out.println((isWhite ? whiteName : blackName) + " ↵
47                 played " +
48                 convertPosition(nextMove.from()) + " to " + ↵
49                 convertPosition(nextMove.to()));
50             board.move(nextMove);
51         }
52         isWhite = !isWhite; // changes who's turn it is at the end of ↵
53         a turn
54     } while (!board.isGameOver());
55     System.out.println("This is the final state of the board");
56     printBoard(); // prints the state of the board when game over
57     if (board.black().length > 0 && board.white().length <= 0)
58         System.out.println(blackName + " is the winner!");
59     else if (board.black().length <= 0 && board.white().length > 0)
60         System.out.println(whiteName + " is the winner!");
61     else
62         System.out.println("It's a draw!");
63 }
64
65 /**
66  * Initializes the program and runs the start menu.
67  */
68 private static void init() {
69     reader = new Scanner(System.in);
70     board = new Board();
71     whiteName = "White(CPU)";

```

6 APPENDIX

```
60     blackName = "Black(CPU)";
61     isWhite = true;
62     int option;
63     System.out.println("*****");
64     System.out.println("Greetings Master! And welcome to Alquerque.");
65     System.out.println("*****");
66     do {
67         printOptions();
68         option = reader.nextInt();
69         switch (option) {
70             case 0:
71                 System.out.println("You have chosen option " + option +
72                                     " : Exit program");
73                 System.out.println("Thank you for playing, have a nice
74                                     day!");
75                 break;
76             case 1: // Player vs Player
77                 System.out.println("You have chosen option " + option +
78                                     " : Player vs Player");
79                 System.out.print("Please enter the name of player 1: ");
80                 reader.nextLine(); // clears input
81                 whiteName = reader.nextLine().trim();
82                 System.out.print("Please enter the name of player 2: ");
83                 blackName = reader.nextLine().trim();
84                 break;
85             case 2: // Player vs CPU
86                 System.out.println("You have chosen option " + option +
87                                     " : Player vs CPU");
88                 String color;
89                 reader.nextLine(); // clears input
90                 do {
91                     System.out.print("Please enter the color you want
92                                     to play " +
93                                     "black or white (B/W): ");
94                     color = reader.nextLine();
95                     if (color.matches("[Bb]")){
96                         System.out.println("\nYou have chosen to play
97                                     black.\n" +
98                                     "The CPU will therefore play white");
99                         System.out.print("Please enter the name of the
100                                     player: ");
101                         blackName = reader.nextLine().trim();
102                         isWhiteCPU = true;
103                     } else if (color.matches("[Ww]")){
104                         System.out.println("\nYou have chosen to play
105                                     white.\n" +
106                                     "The CPU will therefore play black");
107                         System.out.print("Please enter the name of the
108                                     player: ");
109                         whiteName = reader.nextLine().trim();
110                         isBlackCPU = true;
111                     } else {
112                         System.out.println("'" + color + "'" + " is
113                                     not a valid color " +
114                                     "option, please try again.\n");
115                     }
116                 } while (!color.matches("[BbWw]"));
117                 System.out.print("How far ahead do you want the CPU to
118                                     analyze: ");
119                 cpuDepth = reader.nextInt();
```

6 APPENDIX

```

109         reader.nextLine(); // clears input
110         break;
111     case 3: // CPU vs CPU
112         System.out.println("You have chosen option " + option +
113             + ": CPU vs CPU");
114         System.out.print("How far ahead do you want the CPU's
115             to analyze: ");
116         cpuDepth = reader.nextInt();
117         isWhiteCPU = true;
118         isBlackCPU = true;
119         break;
120     default:
121         System.out.println("Invalid option, " + option + " is
122             not a valid option.\n");
123     }
124     } while (option > 3 || option < 0);
125 }
126
127 /**
128  * Prints the option menu to the terminal.
129  */
130 private static void printOptions() {
131     System.out.println("Now, what do you wish to do?");
132     System.out.println("*****");
133     System.out.println("Option 0: Exit program");
134     System.out.println("Option 1: Player vs Player");
135     System.out.println("Option 2: Player vs CPU");
136     System.out.println("Option 3: CPU vs CPU");
137     System.out.println("*****");
138     System.out.println();
139     System.out.print("Please enter the number corresponding " +
140         "to the option you want executed: ");
141 }
142
143 /**
144  * Creates a representation of the game board with the pieces
145  * correctly placed
146  * in the form of a two dimensional array.
147  * Precondition: Relies on method black() and white() to return valid
148  * positions numbered from 1-25
149  * @return a two dimensional array 5 x 5 with the game pieces placed
150  * correctly
151  */
152 private static char[][] boardWithPieces() {
153     char[][] boardArr = new char[6][5]; //A-E & (no 0) 1-5
154     for (int j = 1; j < boardArr.length; j++)
155         for (int i = 0; i < boardArr[j].length; i++)
156             boardArr[j][i] = EMPTY; // Fills board with empty spaces
157     for (int i = 0; i < board.black().length; i++)
158         boardArr[((board.black()[i] - 1) / 5) + 1][((board.black()[i] -
159             1) % 5)] = 'B'; // Places black pieces
160     for (int i = 0; i < board.white().length; i++)
161         boardArr[((board.white()[i] - 1) / 5) + 1][((board.white()[i] -
162             1) % 5)] = 'W'; // Places white pieces
163     return boardArr;
164 }
165
166 /**
167  * prints a representation of the board to the terminal
168  */
169 private static void printBoard() {
170     System.out.println(); // new line

```

6 APPENDIX

```

163     int i = 0, j = 1;
164     System.out.println("    A    B    C    D    E"); //upper-coordinate ←↔
165         line (A-E)
166     char[][] boardWithPieces = boardWithPieces();
167     while (j < 6) {
168         System.out.print(j + " "); //left-hand coordinate (1-5)
169         while (i < 5) {
170             System.out.print("[ " + boardWithPieces[j][i] + " ]");
171             if (i < 4)
172                 System.out.print("-");
173             i++;
174         }
175         System.out.print(" " + (j)); //right-hand coordinate (1-5)
176         System.out.println("");
177         i = 0;
178         if (j % 2 == 1 && j < 5)
179             System.out.println("    | \\ | / | \\ | / |");
180         else if (j % 2 == 0)
181             System.out.println("    | / | \\ | / | \\ |");
182         j++;
183     }
184     System.out.println("    A    B    C    D    E"); //bottom-coordinate ←↔
185         line (A-E)
186     System.out.println(""); // new line
187
188     /**
189     * Test whether an entered coordinate is a valid coordinate
190     * @param coords, a coordinate to be tested
191     * @return true if the coordinate entered is a valid coordinate else ←↔
192     *         returns false
193     */
194     private static boolean isValidCoords(String coords){
195         return (coords.matches("[A-Ea-e][1-5]")); // Regex for matching
196     }
197
198     /**
199     * Converts an input coordinate to the corresponding position on the ←↔
200     * board, determined by numbers 1-25
201     * @param coord move coordinate input from user
202     * @return position on board, represented by an integer (1-25)
203     */
204     private static int convertCoordinate(String coord){
205         int position = 0;
206         switch(Character.toUpperCase(coord.charAt(0))){
207             case 'A': //value of each column is added to the row ←↔
208                 //determined multiple of 5 (e.g. D is 4'th, so positional ←↔
209                 //value is +4)
210                 position = (1+(5*((Integer.parseInt(coord.substring(1))-1) ←↔
211                     )));
212                 break;
213             case 'B':
214                 position = (2+(5*((Integer.parseInt(coord.substring(1))-1) ←↔
215                     )));
216                 break;
217             case 'C':
218                 position = (3+(5*((Integer.parseInt(coord.substring(1))-1) ←↔
219                     )));
220                 break;
221             case 'D':
222                 position = (4+(5*((Integer.parseInt(coord.substring(1))-1) ←↔
223                     )));
224         }

```

6 APPENDIX

```
215         break;
216     case 'E':
217         position = (5+(5*((Integer.parseInt(coord.substring(1))-1)↵
218             )));
219         break;
220     default:
221         return 0;
222 }
223 return position;
224 }
225 /**
226  * Converts an input position, represented by a number 1-25 to the ↵
227  * corresponding coordinates in form [A-E][1-5]
228  * @param position position represented by an int
229  * @return coord position represented by coordinates [A-E][1-5]
230  */
231 private static String convertPosition(int position){
232     String coord = "";
233     switch ((position - 1) % 5){
234     case 0:
235         coord = "A";
236         break;
237     case 1:
238         coord = "B";
239         break;
240     case 2:
241         coord = "C";
242         break;
243     case 3:
244         coord = "D";
245         break;
246     case 4:
247         coord = "E";
248         break;
249     }
250     coord = coord + ((position - 1) / 5 + 1);
251     return coord;
252 }
253 } // end of alquerque class
```