Alquerque

Danny Nicolai Larsen, Mikkel Brix Nielsen & Steffen Bach

November 17, 2021

Contents

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 initilization

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

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.

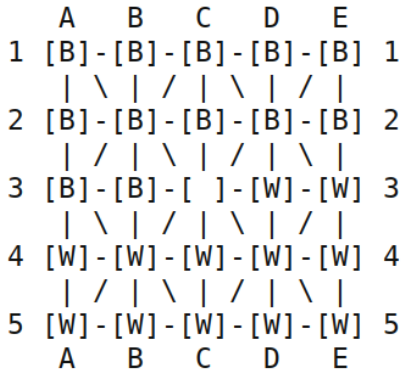
2.3 Getters & setters:

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.

Figure 1: Chess Notation



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

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

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   private static int convertCoordinate(String coord){
2       int position = 0;
3       switch(Character.toUpperCase(coord.charAt(0))){
4           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)
5               position = (1+(5*((Integer.parseInt(coord.substring(1))-1)↩
                    )));
6               break;
7           case 'B':
8               position = (2+(5*((Integer.parseInt(coord.substring(1))-1)↩
                    )));
9               break;
10          case 'C':
11              position = (3+(5*((Integer.parseInt(coord.substring(1))-1)↩
                    )));
12              break;
13          case 'D':
14              position = (4+(5*((Integer.parseInt(coord.substring(1))-1)↩
                    )));
15              break;
16          case 'E':
17              position = (5+(5*((Integer.parseInt(coord.substring(1))-1)↩
                    )));
18              break;
19          default:
20              return 0;
21      }
22      return position;
23  }
```

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        private static boolean isValidCoords(String coords){
2            return (coords.matches("[A-Ea-e][1-5]")); // Regex for ↩
                 matching
3        }
```

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    private static String convertPosition(int position){
2        String coord = "";
3        switch ((position - 1) % 5){
4            case 0:
5                coord = "A";
6                break;
7            case 1:
8                coord = "B";
9                break;
10           case 2:
11               coord = "C";
12               break;
13           case 3:
14               coord = "D";
15               break;
16           case 4:
17               coord = "E";
18               break;
19       }
20
```

```
21          coord = coord + ((position - 1) / 5 + 1);
22          return coord;
23      }
```

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.

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 didn't 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
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 : ↩
                            blackName) + "'s turn" + ", please enter which " +
21                              "piece you want to move: ");
22                      coordsFrom = reader.nextLine().trim();
23                      System.out.print("Please enter where you want to move ↩
                            the piece: ");
24                      coordsTo = reader.nextLine().trim();
25                      if (isValidCoords(coordsFrom) && isValidCoords(↩
                            coordsTo)) { //Checks if input is a valid letter+↩
                            number
26                          nextMove = new Move(convertCoordinate(coordsFrom),↩
                                 convertCoordinate(coordsTo)); //Converts ↩
                                coordinate to int position
27                          if (board.isLegal(nextMove))
28                              inputWithinRange = true;
29                      }
30                      if (!inputWithinRange)
31                          System.out.println(coordsFrom + " to " + coordsTo ↩
                                + " is " +
32                                  "not a valid move, please enter a ↩
                                        coordinate A-E 1-5.");
33                  } while (!inputWithinRange);
34                  board.move(nextMove);
35              } else if (!board.isGameOver()) {
36                  nextMove = new Minimax().nextMove(board, cpuDepth, isWhite↩
                        );
37                  System.out.println((isWhite ? whiteName : blackName) + " ↩
                        played " +
38                          convertPosition(nextMove.from()) + " to " + ↩
                              convertPosition(nextMove.to()));
39                  board.move(nextMove);
40              }
41              isWhite = !isWhite; // changes who's turn it is at the end of ↩
                    a turn
42          } while (!board.isGameOver());
43          System.out.println("This is the final state of the board");
44          printBoard(); // prints the state of the board when game over
45          if (board.black().length > 0 && board.white().length <= 0)
46              System.out.println(blackName + " is the winner!");
47          else if (board.black().length <= 0 && board.white().length > 0)
48              System.out.println(whiteName + " is the winner!");
49          else
50              System.out.println("It's a draw!");
51      }
52
53      /**
54       * Initializes the program and runs the start menu.
55       */
56      private static void init() {
57          reader = new Scanner(System.in);
58          board = new Board();
59          whiteName = "White(CPU)";
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 ↩
                            + ": Exit program");
72                      System.out.println("Thank you for playing, have a nice↩
                             day!");
73                      break;
74                  case 1: // Player vs Player
75                      System.out.println("You have chosen option " + option ↩
                            + ": Player vs Player");
76                      System.out.print("Please enter the name of player 1: "↩
                            );
77                      reader.nextLine(); // clears input
78                      whiteName = reader.nextLine().trim();
79                      System.out.print("Please enter the name of player 2: "↩
                            );
80                      blackName = reader.nextLine().trim();
81                      break;
82                  case 2: // Player vs CPU
83                      System.out.println("You have chosen option " + option ↩
                            + ": Player vs CPU");
84                      String color;
85                      reader.nextLine(); // clears input
86                      do {
87                          System.out.print("Please enter the color you want ↩
                                to play " +
88                                  "black or white (B/W): ");
89                          color = reader.nextLine();
90                          if (color.matches("[Bb]")){
91                              System.out.println("\nYou have chosen to play ↩
                                    black.\n" +
92                                      "The CPU will therefore play white");
93                              System.out.print("Please enter the name of the↩
                                     player: ");
94                              blackName = reader.nextLine().trim();
95                              isWhiteCPU = true;
96                          } else if (color.matches("[Ww]")){
97                              System.out.println("\nYou have chosen to play ↩
                                    white.\n" +
98                                      "The CPU will therefore play black");
99                              System.out.print("Please enter the name of the↩
                                     player: ");
100                             whiteName = reader.nextLine().trim();
101                             isBlackCPU = true;
102                         } else {
103                             System.out.println("'" + color + "'" + " is ↩
                                    not a valid color " +
104                                     "option, please try again.\n");
105                         }
106                     } while (!color.matches("[BbWw]"));
107                     System.out.print("How far ahead do you want the CPU to↩
                             analyze: ");
108                     cpuDepth = reader.nextInt();
109                     reader.nextLine(); // clears input
110                     break;
111                 case 3: // CPU vs CPU
112                     System.out.println("You have chosen option " + option ↩
                            + ": CPU vs CPU");
113                     System.out.print("How far ahead do you want the CPU's ↩
                            to analyze: ");
114                     cpuDepth = reader.nextInt();
115                     isWhiteCPU = true;
116                     isBlackCPU = true;
```

```
117                     break;
118                 default:
119                     System.out.println("Invalid option, " + option + " is ↩
                        not a valid option.\n");
120             }
121         } while (option > 3 || option < 0);
122     }
123 
124     /**
125      * Prints the option menu to the terminal.
126      */
127     private static void printOptions() {
128         System.out.println("Now, what do you wish to do?");
129         System.out.println("****************************");
130         System.out.println("Option 0: Exit program");
131         System.out.println("Option 1: Player vs Player");
132         System.out.println("Option 2: Player vs CPU");
133         System.out.println("Option 3: CPU vs CPU");
134         System.out.println("****************************");
135         System.out.println();
136         System.out.print("Please enter the number corresponding " +
137                 "to the option you want executed: ");
138     }
139 
140     /**
141      * Creates a representation of the game board with the pieces ↩
            correctly placed
142      * in the form of a two dimensional array.
143      * Precondition: Relies on method black() and white() to return valid ↩
            positions numbered from 1-25
144      * @return a two dimensional array 5 x 5 with the game pieces placed ↩
            correctly
145      */
146     private static char[][] boardWithPieces() {
147         char[][] boardArr = new char[6][5]; //A-E & (no 0) 1-5
148         for (int j = 1; j < boardArr.length; j++)
149             for (int i = 0; i < boardArr[j].length; i++)
150                 boardArr[j][i] = EMPTY; // Fills board with empty spaces
151         for (int i = 0; i < board.black().length; i++)
152             boardArr[((board.black()[i] - 1) / 5) + 1][((board.black()[i] ↩
                    - 1) % 5)] = 'B'; // Places black pieces
153         for (int i = 0; i < board.white().length; i++)
154             boardArr[((board.white()[i] - 1) / 5) + 1][((board.white()[i] ↩
                    - 1) % 5)] = 'W'; // Places white pieces
155         return boardArr;
156     }
157 
158     /**
159      * prints a representation of the board to the terminal
160      */
161     private static void printBoard() {
162         System.out.println(); // new line
163         int i = 0, j = 1;
164         System.out.println("   A   B   C   D   E"); //upper-coordinate-↩
                line (A-E)
165         char[][] boardWithPieces = boardWithPieces();
166         while (j < 6) {
167             System.out.print(j + " "); //left-hand coordinate (1-5)
168             while (i < 5) {
169                 System.out.print("[" + boardWithPieces[j][i] + "]");
170                 if (i < 4)
171                     System.out.print("-");
```

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

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