Programming Project Assignment 1

Candidate No: 105936

9th May 2019

Contents

1. Introduction

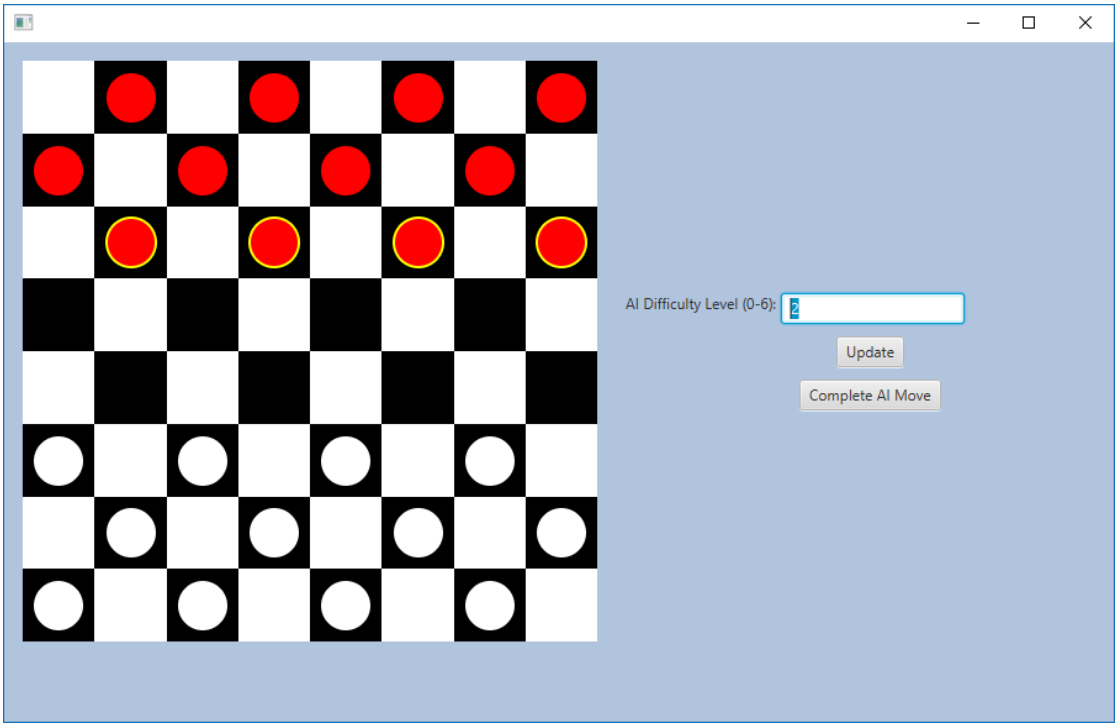

Figure 1: An Image of the Starting screen for the game

Figure 1 shows the main screen of the game. To start with the user(you) is red and the AI is white these may be referred to dark and light in the document respectively. I probably started developing this the wrong way round developing the GUI and the game part first then adding the AI implemented with minimax later. However the alternative would to make a console game then covert it into a GUI game which probably would have produced a better output but would have taken longer to implement. I originally planed on creating a MVVM GUI application however due to time constraints and difficulty setting up the mvvm project in JavaFX with mvvmFX I decided to not follow a formal framework/structure as it would have taken more time to learn and implement. I have a number of classes in my project:

1.1. Main

My main class focuses on the visuals as all of the GUI code is in this class and it also manages some of the control as well as this allows the user to interact with the board.

1.2. AI

This class contains the minimax function and the evaluation function more time could and probably should have been spend on the evaluation function. I will go on in more detail about how the evaluation function works further on in the document.

1.3. Colour

This is actually a very simple enum which is used to identify the two different players Light and Dark

1.4. Draught

This represents a draught piece and contains it's location information if it is crowned and its Colour.

1.5. Game

This manages the game and the game board state if I were redoing the project from scratch I probably would have separated out the game board data from the methods and logic as this lead to problems when implementing minimax. The game class contains methods for:

- finding possible moves
- carrying out moves
- the successor function
- checking for a winner

This contains the majority of the logic for the game internals.

1.6. Move

This represents the movement of a draught and allows me to separate out some logic. And allows me to store a turn choice as an object which can be passed to other classes.

1.7. Capturing Move

This class extends the move class and allows me to store capturing moves as a different but compatible type. This is mainly because if you capture a draught and it is possible to capture another you get to continue your turn therefore it is vital to store the difference.

2. Program Functionality

2.1. Game Internals

2.1.1. Interactive Checkers Gameplay

As mentioned in the introduction I started by developing the game internals and the GUI therefore the gameplay is fairly smooth and functionally correct in terms of rules of the game. The way it works is the Main class asks for a list of possible moves from the game object and then displays them to the user for them to then decide what move to make once chosen it is run through the select move function in the game class which carries out the move for the player. For the AI a similar thing happens once the user presses the complete AI move button then the Main class gets the game and passes it to the AI which returns a move which the Main thread then passes on to the game object. The user starts as the red/dark player and the AI is the white/light player.

2.1.2. Valid State Representation

The state representation used is the Game class which can be found at section C.3 and its made up of the following

Draughts Array List An array list holds the draughts which are on the board currently. the draughts are stored in random order and their position information is stored in the draught class itself. The draughts also store colour information. The position information is an x and y coordinate starting from the top left of the board as that makes it easy to display in the GUI.

Current Turn This stores the colour of the current player so the game object knows who's turn it is and therefore which moves to return when calculating them.

Selected Draught This is used for the completing of multiple jumps in one turn as if the player has the option to perform multiple jumps it is only possible to

make moves on the draught which completed the original jump and then only if a jump is available.

Multi-step move This stores if the game is currently a part of a multi step move this means that the game can restrict the moves to only the currently selected draught.

Game Winner This stores which colour is the game winner so that Main class can display to the user which colour is the winner when the game completes.

Game Complete This stores if the game is complete or not. and therefore display who the winner is to the user from the main class.

2.1.3. Successor function

The calculation of the moves is in three steps firstly the draught calculates its possible moves given its position and if its crowned or not and removes any which aren't in bounds. Secondly the Game class then removes any blocked moves unless it can make a jump. Thirdly the findAllPossibleMoves() method checks for if we are currently in a multi-step move and if so restricts the moves accordingly otherwise it gets all of the moves for the current players draughts. It also makes use of the findAllPossibleMoves() method which works by firstly checking if we are currently on a multi-step move and if so only returning the valid capture moves from the selected draught. The function that I have named successor is the function in the Game class section C.3 which returns a following game object when passed a move.

2.1.4. Use of Successor Function

The find all moves method is used in both the game and AI classes and the find moves method is used in the main class to display the possible moves for each draught. The find all moves is used in the game class to check if there are possible moves if there aren't then the other player wins. It is used in the AI class as a part of the minimax method to get the children of the current game board state. The successor functions don't specifically validate the moves however the only moves either the AI or user have available to them come from the successor function therefore they must be valid.

2.1.5. Invalid User Moves

Invalid user moves are handled by simply not providing the function to complete

incorrect moves or more accurately only providing the options to select valid moves. As there is no way for a user to select an incorrect move there is no explanation given with one exception. If the user attempts to deselect the draught during a multi-step move the program gives an error warning with an explanation saying they are only able to move the currently selected draught.

2.1.6. Minimax Evaluation

The minimax algorithm has been implemented in the AI class section B. And uses the evaluation function evaluateGameBoard() which has a very simple implementation it simply counts the number of ai draughts and subtracts the number of opponent draughts. Ideally if I had more time I would like to include if a draught was crowned or not in the calculation for example crowned draughts are worth 1.5 or 2 and going further it is also worth including non crowned draughts position up the board in the calculation although that would require a lot of fine tuning to ensure that the function does not overvalue position over pieces.

2.1.7. Alpha Beta Pruning

I implemented the minimax alpha beta pruning algorithm in the AI class. It is used as a part of the evaluateMoves() method.

2.1.8. Variable AI difficulty

Variable AI difficultly is provided in the form of of a text box on the program where the user is able to choose a difficulty between 0-7 this represents the depth that the minimax function works to. I found that when I set the value to 8 or 9 the turns would take too long and the program would reach the 1gb JVM heap size limit therefore I have restricted the number to 7. If the user attempts to put a value higher than 7 then the program truncates it to 7.

2.1.9. Valid AI Moves

As mentioned earlier the AI class selects a move from moves provided from the successor function therefore are only limited to legitimate moves.

2.1.10. Multi-step User Moves

If the user has the ability to continue performing moves the application will keep the selected draught highlighted and display only the potential move. Even if there is only one move available the user still needs to select it. This has been done so that the user does not get confused when many pieces moves around.

The multistep moves are checked for after every jump as a part of the selectMove () method in the game class section C.3. Then when the moves are requested as a part of the findAllPossibleMoves() method the method restricts the moves to capturing moves of the selected draught i.e. the one which completed the last jump. Multi-step moves are also stopped if a draught gets crowned this is done by only continuing the multi-step jumps if the draught isn't on kings row or is crowned the code for this can be found as a part of the selectMove() method in the game class section C.3.

2.1.11. Multi-step AI moves

This is implemented in a similar way to the user multi-step. To continue the game the user needs to press the complete AI move button. This was done intentionally so that the user is able to see each of the moves made at their own pace.

2.1.12. Forced Takes

When a user has the option to make a jump they are not allowed to complete a normal move. Therefore the program only displays the jumps they are able to complete as a move if there is only one jump then they are only provided the one option this is preferred over automatically completing the move to avoid confusion on the users part. This works as a part of the findAllPossibleMoves() method which gets all of the possible moves as mentioned earlier it then runs it through the removeNonCapturingMovesIfCapturingMoveExists(ArrayList<Move> moveList) method. which if there is a capturing move then it restricts the moves to just capturing moves otherwise it returns all the moves.

2.1.13. Automatic King Conversion

Once a king reaches the king's row it gets crowned. This is done with the crownDraughtIfPossible() method in the draught class section C.2. This is called on a draught after it moves. Therefore will always convert a draught correctly to display that a draught is a king a smaller circle is added to the top of the draught.

2.2. HCI/GUI

2.2.1. GUI Updates

The GUI is refreshed after each turn and after the control panel is updated this is done through the updateDisplay() method which redraws the display with information from the game class variable.

2.2.2. Full Graphical Board Display

The board displayed on the screen is made up of a gridPane each of the checkerboard boxes are a coloured square these are generated in the generateBoard() method in the main class section A. The draughts are then added on top of them in the displayDraughtsOnBoard() method which makes use of the generateDraughtVisual() method. Then possible moves are only displayed on the correct conditions.

2.2.3. Mouse Interaction

The user interaction with the program is almost entirely mouse based except from setting the AI difficulty. To make a move the user needs to select a draught and then select a square to move it to. This is done by making use of the setOnMouseClicked() method to set an event handler for when the user clicks on a draught or a square the draught can move to. When the user clicks on a draught with possible moves the moves are displayed to the user the user can then click on any of the available moves for that draught. Once the move is selected it is passed on to the game to handle the changes, once it returns the program refreshes the display.

2.2.4. GUI pauses on multi-step moves

The GUI will only continue with the game when the user selects an action either by selecting a move or clicking the complete AI move button this was done intentionally so that the user is able to play at their own pace. This means that the program will always stop during multi-step moves and the user will have to initiate the continuation.

2.2.5. Display of basic game rules

Due to time constraints I was not able to create a page with the rules of the game however I was able to add a button to the control panel which opens a webpage with the game rules on it. This has been done in the generateControlPanel() method.

2.2.6. Possible moves display

This is done by firstly showing the draughts which can be moved then, after a draught is selected, the possible squares it can move to have indicators. This is done by adding code to the generateDraughtVisual() method so that the draughts have yellow borders if they have moves. then when a draught is selected possible moves are displayed using the displayPossibleMoves() method.

Appendices

A. Main

```
1  import ai.AI;
2  import game.Colour;
3  import game.Draught;
4  import game.Game;
5  import javafx.application.Application;
6  import javafx.geometry.HPos;
7  import javafx.geometry.Insets;
8  import javafx.geometry.Pos;
9  import javafx.geometry.VPos;
10 import javafx.scene.Node;
11 import javafx.scene.Scene;
12 import javafx.scene.control.*;
13 import javafx.scene.layout.*;
14 import javafx.scene.paint.Color;
15 import javafx.scene.shape.Circle;
16 import javafx.stage.Stage;
17 import moves.Move;
18
19 import java.io.IOException;
20 import java.util.ArrayList;
21 import java.util.Optional;
22
23 public class Main extends Application {
24
25     private static final int BOARDSIZE = 8;
26     private static final Background WHITEBACKGROUND = new Background(
       new BackgroundFill(Color.WHITE, CornerRadii.EMPTY, Insets.EMPTY));
27     private static final Background BLACKBACKGROUND = new Background(
       new BackgroundFill(Color.BLACK, CornerRadii.EMPTY, Insets.EMPTY));
28     private Game game;
29     private Stage primaryStage;
30     private AI ai;
31
32
33     @Override
34     public void start(Stage primaryStage) {
35         game = new Game();
36         game.setUpNewGame();
37         ai = new AI(2, Colour.LIGHT);
38         this.primaryStage = primaryStage;
39         updateDisplay();
40     }
41
```

```
42
43      public static void main(String[] args) {
44          launch(args);
45      }
46
47      public GridPane generateDisplay(GridPane root){
48          GridPane board = generateBoard();
49          board = displayDraughtsOnBoard(board);
50          if (game.getSelectedDraught() != null && game.getCurrentTurn()
        == Colour.DARK) {
51              board = displayPossibleMoves(board, game.getSelectedDraught
        ());
52          }
53          root.add(board, 0, 0);
54          VBox controlPanel = generateControlPanel();
55          root.add(controlPanel, 1, 0);
56          return root;
57      }
58
59      public VBox generateControlPanel() {
60
61          VBox controlPanel = new VBox();
62          HBox aiDifficulty = new HBox();
63          Label aiDifficultyLabel = new Label("AI Difficulty Level (0-7):
         ");
64          TextField aiDifficultyTextField = new TextField(String.format("
        %d",ai.getDifficulty()));
65          aiDifficulty.getChildren().addAll(aiDifficultyLabel,
        aiDifficultyTextField);
66          controlPanel.getChildren().add(aiDifficulty);
67
68          Button updateValues = new Button("Update");
69          updateValues.setOnMouseClicked(event -> {
70              String str = aiDifficultyTextField.getText();
71              int temp = Integer.parseInt(str);
72              ai.setDifficulty(temp > 7 ? 7 : temp);
73              updateDisplay();
74          });
75          controlPanel.getChildren().add(updateValues);
76
77          Button competeAIMove = new Button("Complete AI Move");
78          competeAIMove.setOnMouseClicked(event -> {
79              if (game.getCurrentTurn() == Colour.LIGHT) {
80                  game.selectMove(ai.selectMove(game));
81                  updateDisplay();
82              }
83          });
84
85          controlPanel.getChildren().add(competeAIMove);
```

```
86
87          Button rules = new Button("Rules");
88          rules.setOnMouseClicked(event -> {
89              try {
90                  java.awt.Desktop.getDesktop().browse(java.net.URI.create(
    "http://www.indepthinfo.com/checkers/play.shtml"));
91              } catch (IOException e) {
92                  e.printStackTrace();
93              }
94          });
95          controlPanel.getChildren().add(rules);
96          controlPanel.setAlignment(Pos.CENTER);
97          controlPanel.setSpacing(10);
98          return controlPanel;
99      }
100
101     public GridPane generateBoard() {
102         GridPane board = new GridPane();
103         for (int row = 0; row < BOARDSIZE; row++) {
104             for (int column = 0; column < BOARDSIZE; column++) {
105                 StackPane square = new StackPane();
106                 square.setBackground((row + column) % 2 == 0 ?
    WHITEBACKGROUND : BLACKBACKGROUND);
107                 board.add(square, column, row);
108
109             }
110         }
111         for (int i = 0; i < BOARDSIZE; i++) {
112             ColumnConstraints columnConstraints = new ColumnConstraints
    ();
113             columnConstraints.setPercentWidth(50);
114             board.getColumnConstraints().add(columnConstraints);
115             RowConstraints rowConstraints = new RowConstraints();
116             rowConstraints.setPercentHeight(50);
117             board.getRowConstraints().add(rowConstraints);
118         }
119
120         board.setPadding(new Insets(15, 15, 15, 15));
121         return board;
122     }
123
124     public GridPane displayPossibleMoves(GridPane board, Draught
    draught){
125         ArrayList<Move> moves = game.findPossibleMoves(draught);
126         for (Move move : moves){
127             Circle moveVisual = generateMoveVisual(move);
128             board.add(moveVisual, move.getNewXPosition(), move.
    getNewYPosition());
129             GridPane.setHalignment(moveVisual, HPos.CENTER);
```

```
130             GridPane.setValignment(moveVisual, VPos.CENTER);
131         }
132         return board;
133     }
134 
135     public Circle generateMoveVisual(Move move){
136         Circle moveVisual = new Circle(20);
137         moveVisual.setFill(Color.TRANSPARENT);
138         moveVisual.setStrokeWidth(2);
139         moveVisual.setStroke(Color.WHITE);
140         moveVisual.setOnMouseClicked(event -> selectMove(move));
141         return moveVisual;
142     }
143 
144     public GridPane displayDraughtsOnBoard(GridPane board) {
145         for (Draught draught : game.getDraughtArrayList()) {
146             Node draughtVisual = generateDraughtVisual(draught);
147             board.add(draughtVisual, draught.getxPosition(), draught.
    getyPosition());
148             GridPane.setHalignment(draughtVisual, HPos.CENTER);
149             GridPane.setValignment(draughtVisual, VPos.CENTER);
150         }
151         return board;
152     }
153 
154     public StackPane generateDraughtVisual(Draught draught){
155         StackPane rtnPane = new StackPane();
156 
157         Circle draughtVisual = new Circle(20);
158         if (draught.getColour() == Colour.DARK) {
159             draughtVisual.setFill(Color.RED);
160         } else if (draught.getColour() == Colour.LIGHT) {
161             draughtVisual.setFill(Color.WHITE);
162         }
163 
164         if (game.draughtsWithPossibleMoves().contains(draught) &&
    draught.getColour() == Colour.DARK) {
165 
166             // If draught has possible moves
167             rtnPane.setOnMouseClicked(event -> selectDraught(draught));
168             if (game.getSelectedDraught() == null) {
169                 draughtVisual.setStroke(Color.YELLOW);
170             }
171         }
172         if (draught.equals(game.getSelectedDraught())) {
173             draughtVisual.setStroke(Color.CYAN);
174         }
175         draughtVisual.setStrokeWidth(2);
176         rtnPane.getChildren().add(draughtVisual);
```

```
177         if (draught.isCrowned()){
178             Circle crownVisual = new Circle(10);
179             crownVisual.setFill(Color.TRANSPARENT);
180             crownVisual.setStroke(Color.BLACK);
181             crownVisual.setStrokeWidth(2);
182             rtnPane.getChildren().add(crownVisual);
183         }
184 
185         return rtnPane;
186     }
187 
188     public GridPane setupRoot() {
189         GridPane root = new GridPane();
190 
191         root.setHgap(8);
192         root.setVgap(8);
193 
194         ColumnConstraints col0 = new ColumnConstraints();
195         col0.setPrefWidth(500);
196         ColumnConstraints col1 = new ColumnConstraints();
197         col1.setPrefWidth(400);
198         root.getColumnConstraints().addAll(col0, col1);
199 
200         RowConstraints row0 = new RowConstraints();
201         row0.setPrefHeight(500);
202         root.getRowConstraints().add(row0);
203         root.setBackground(new Background(new BackgroundFill(Color.
    LIGHTSTEELBLUE, CornerRadii.EMPTY, Insets.EMPTY)));
204 
205 
206         return root;
207     }
208 
209     public void selectMove(Move move){
210         game.selectMove(move);
211         updateDisplay();
212     }
213 
214     public void selectDraught(Draught draught){
215         if (!game.isCurrentMultiStepMove()) {
216             if (draught.equals(game.getSelectedDraught())) {
217 
218                 game.setSelectedDraught(null);
219             } else {
220                 game.setSelectedDraught(draught);
221             }
222         } else {
223             Alert alert = new Alert(Alert.AlertType.WARNING);
224             alert.setTitle("Unable to complete move");
```

```
225             alert.setHeaderText("Unable to complete move");
226             alert.setContentText("Is it not possible to select any other
    draught therefore you must complete the move with the selected
    tile");
227             alert.showAndWait();
228         }
229         updateDisplay();
230 
231     }
232 
233     private void updateDisplay() {
234         primaryStage.setScene(new Scene(
235             generateDisplay(setupRoot()),
236             900,
237             550)
238         );
239         primaryStage.show();
240         if (game.isGameComplete()) {
241             ButtonType reset = new ButtonType("Reset", ButtonBar.
    ButtonData.OTHER);
242             ButtonType close = new ButtonType("Close", ButtonBar.
    ButtonData.CANCEL_CLOSE);
243             Alert alert = new Alert(Alert.AlertType.NONE, "The Game has
    come to a conclusion the winner is " + game.getGameWinner().
    toString(), reset, close);
244             alert.setTitle("Game Complete");
245             alert.setHeaderText("Game Complete");
246             Optional<ButtonType> result = alert.showAndWait();
247             if (result.orElse(close) == reset){
248                 game.setUpNewGame();
249                 updateDisplay();
250             }
251 
252         }
253     }
254 }
```

B. AI

```
1 package ai;
2 
3 import game.Colour;
4 import game.Draught;
5 import game.Game;
6 import javafx.util.Pair;
7 import moves.Move;
8 
9 import java.util.ArrayList;
```

```
10 import java.util.stream.Collectors;
11
12 public class AI {
13
14
15     int difficulty;
16     Colour colour;
17
18     public AI(int difficulty, Colour colour) {
19         this.difficulty = difficulty;
20         this.colour = colour;
21     }
22
23     public int getDifficulty() {
24         return difficulty;
25     }
26
27     public void setDifficulty(int difficulty) {
28         this.difficulty = difficulty;
29     }
30
31     public Colour getColour() {
32         return colour;
33     }
34
35     public Move selectMove(Game game) {
36         ArrayList<Pair<Integer, Move>> evaluatedMoves = evaluateMoves(
       game);
37         evaluatedMoves.sort((lhs, rhs) -> lhs.getKey() > rhs.getKey() ?
        -1 : (lhs.getKey() < rhs.getKey()) ? 1 : 0);
38         return evaluatedMoves.get(0).getValue();
39     }
40
41
42     public ArrayList<Pair<Integer, Move>> evaluateMoves(Game game) {
43         ArrayList<Pair<Integer, Move>> evaluatedMoves = new ArrayList
       <>();
44
45         for (Move move : game.findAllPossibleMoves()) {
46             Game futureGame = game.successor(move);
47             evaluatedMoves.add(new Pair<>(minimaxAlphaBeta(futureGame,
       difficulty, false, Integer.MIN_VALUE, Integer.MAX_VALUE), move));
48         }
49
50         return evaluatedMoves;
51     }
52
53     private int minimaxAlphaBeta(Game gameBoard, int depth, boolean
       maxPlayer, int alpha, int beta) {
```

```
54          if (depth == 0 || gameBoard.isGameComplete()) {
55              return evaluateGameBoard(gameBoard);
56          }
57          if (maxPlayer) {
58              int value = Integer.MIN_VALUE;
59              ArrayList<Game> children = gameBoard.findAllPossibleMoves().
        stream().map(gameBoard::successor).collect(Collectors.toCollection
        (ArrayList::new));
60              for (Game game : children) {
61                  value = Math.max(value, minimaxAlphaBeta(game, depth - 1,
         false, alpha, beta));
62                  alpha = Math.max(alpha, value);
63                  if (alpha >= beta) {
64                      break;
65                  }
66              }
67              return value;
68          } else {
69              int value = Integer.MAX_VALUE;
70              ArrayList<Game> children = gameBoard.findAllPossibleMoves().
        stream().map(gameBoard::successor).collect(Collectors.toCollection
        (ArrayList::new));
71              for (Game game : children) {
72                  value = Math.min(value, minimaxAlphaBeta(game, depth - 1,
         true, alpha, beta));
73                  beta = Math.min(beta, value);
74                  if (alpha >= beta) {
75                      break;
76                  }
77              }
78              return value;
79          }
80
81      }
82
83
84      private int evaluateGameBoard(Game gameBoard) {
85          ArrayList<Draught> draughts = gameBoard.getDraughtArrayList();
86          int aiDraughts = Math.toIntExact(draughts.stream().filter(
        draught -> draught.getColour() == getColour()).count());
87          int opponentDraughts = Math.toIntExact(draughts.stream().filter
        (draught -> draught.getColour() != getColour()).count());
88          return aiDraughts - opponentDraughts;
89      }
90  }
```

C. Game

C.1. Colour

```
1  package game;
2
3  public enum Colour {
4     LIGHT, DARK
5  }
```

C.2. Draught

```
1  package game;
2
3  import moves.Move;
4  import sun.reflect.generics.reflectiveObjects.NotImplementedException
       ;
5
6  import java.util.ArrayList;
7
8  public class Draught implements Cloneable{
9     int xPosition;
10    int yPosition;
11    boolean crowned;
12    Colour colour;
13
14
15    public int getxPosition() {
16       return xPosition;
17    }
18
19    public void setxPosition(int xPosition) {
20       this.xPosition = xPosition;
21    }
22
23    public int getyPosition() {
24       return yPosition;
25    }
26
27    public void setyPosition(int yPosition) {
28       this.yPosition = yPosition;
29    }
30
31    public boolean isCrowned() {
32       return crowned;
33    }
34
35    public Colour getColour() {
```

```
36          return colour;
37      }
38
39
40      public Draught(Draught draught) {
41          xPosition = draught.xPosition;
42          yPosition = draught.yPosition;
43          crowned = draught.crowned;
44          colour = draught.colour;
45      }
46
47      public Draught(int xPosition, int yPosition, Colour colour)
48      {
49          this.xPosition = xPosition;
50          this.yPosition = yPosition;
51          crowned = false;
52          this.colour = colour;
53      }
54
55      public ArrayList<Move> listPossibleMoves() {
56
57          ArrayList<Move> possibleMoves = new ArrayList<>();
58          int yDirection;
59
60          if (colour == Colour.LIGHT) {
61              yDirection = -1;
62          } else if (colour == Colour.DARK) {
63              yDirection = 1;
64          } else throw new NotImplementedException();
65
66          if (crowned) {
67              possibleMoves.add(new Move(this, xPosition - 1, yPosition -
    yDirection));
68              possibleMoves.add(new Move(this, xPosition + 1, yPosition -
    yDirection));
69          }
70
71          possibleMoves.add(new Move(this, xPosition - 1, yPosition +
    yDirection));
72          possibleMoves.add(new Move(this, xPosition + 1, yPosition +
    yDirection));
73
74          possibleMoves.removeIf(move -> !move.moveInBounds());
75
76          return possibleMoves;
77      }
78
79      public void crownDraughtIfPossible() {
80          if (!crowned) {
```

```
81              crowned = isDraughtOnKingsRow();
82          }
83      }
84
85
86      public boolean isDraughtOnKingsRow() {
87          return colour.equals(Colour.LIGHT) ? yPosition == 0 : yPosition
        == 7;
88      }
89
90      @Override
91      protected Draught clone(){
92          Draught clone = null;
93          try {
94              clone = (Draught) super.clone();
95              clone.colour = getColour();
96              clone.crowned = crowned;
97              clone.xPosition = xPosition;
98              clone.yPosition = yPosition;
99
100         } catch (CloneNotSupportedException e) {
101             e.printStackTrace();
102         }
103         return clone;
104     }
105 }
```

C.3. Game

```
1  package game;
2
3  import moves.CapturingMove;
4  import moves.Move;
5
6  import java.util.ArrayList;
7  import java.util.Collection;
8  import java.util.Optional;
9  import java.util.stream.Collectors;
10
11 public class Game implements Cloneable {
12     private ArrayList<Draught> draughtArrayList;
13     private Colour currentTurn;
14     private Draught selectedDraught;
15     private boolean isCurrentMultiStepMove;
16     private Colour gameWinner;
17     private boolean isGameComplete;
18
19     public ArrayList<Draught> getDraughtArrayList() {
20         return draughtArrayList;
```

```
21      }
22
23      public Colour getCurrentTurn() {
24          return currentTurn;
25      }
26
27      public Draught getSelectedDraught() {
28          return selectedDraught;
29      }
30
31      public void setSelectedDraught(Draught selectedDraught) {
32          this.selectedDraught = selectedDraught;
33      }
34
35      public Colour getGameWinner() {
36          return gameWinner;
37      }
38
39      public boolean isGameComplete() {
40          return isGameComplete;
41      }
42
43      public Game() {
44      }
45
46      public Game(Game game) {
47          draughtArrayList = new ArrayList<>();
48          for (Draught draught : game.draughtArrayList) {
49              draughtArrayList.add(new Draught(draught));
50          }
51
52          currentTurn = game.currentTurn;
53          selectedDraught = game.selectedDraught;
54          isCurrentMultiStepMove = game.isCurrentMultiStepMove;
55          isGameComplete = game.isGameComplete;
56      }
57
58      public void setUpNewGame() {
59          draughtArrayList = new ArrayList<>();
60          for (int i = 0; i < 8; i++) {
61              if (i % 2 == 1) {
62                  draughtArrayList.add(new Draught(i, 0, Colour.DARK));
63                  draughtArrayList.add(new Draught(i, 2, Colour.DARK));
64                  draughtArrayList.add(new Draught(i, 6, Colour.LIGHT));
65              } else {
66                  draughtArrayList.add(new Draught(i, 1, Colour.DARK));
67                  draughtArrayList.add(new Draught(i, 5, Colour.LIGHT));
68                  draughtArrayList.add(new Draught(i, 7, Colour.LIGHT));
69              }
```

```
70          }
71          currentTurn = Colour.DARK;
72          isCurrentMultiStepMove = false;
73          isGameComplete = false;
74      }
75
76      public ArrayList<Move> findAllPossibleMoves() {
77          if (isCurrentMultiStepMove) {
78
79              return findPossibleMoves(selectedDraught)
80                      .stream()
81                      .filter(move -> move instanceof CapturingMove)
82                      .collect(Collectors.toCollection(ArrayList::new));
83          }
84
85          ArrayList<Move> moveList = draughtArrayList
86                  .stream()
87                  .filter(draught -> draught.colour == currentTurn)
88                  .map(this::findPossibleMoves)
89                  .flatMap(Collection::stream)
90                  .collect(Collectors.toCollection(ArrayList::new));
91
92          return removeNonCapturingMovesIfCapturingMoveExists(moveList);
93
94      }
95
96      private ArrayList<Move>
        removeNonCapturingMovesIfCapturingMoveExists(ArrayList<Move>
        moveList) {
97          ArrayList<Move> captureMoveList = moveList
98                  .stream()
99                  .filter(move -> move instanceof CapturingMove)
100                 .collect(Collectors.toCollection(ArrayList::new));
101
102         return !captureMoveList.isEmpty() ? captureMoveList : moveList;
103     }
104
105     public ArrayList<Move> findPossibleMoves(Draught draught) {
106         ArrayList<Move> possibleMoves = new ArrayList<>();
107
108         for (Move move : draught.listPossibleMoves()) {
109             if (isMoveBlocked(move)) {
110                 Optional<Draught> capturedDraught = draughtArrayList.
111                     stream()
112                     .filter(draught1 -> draught1.xPosition == move.
        getNewXPosition()
113                             && draught1.yPosition == move.getNewYPosition()
114                             && draught.colour != draught1.colour)
115                     .findFirst();
```

```
116             if (capturedDraught.isPresent()) {
117                 Move captureMove = new CapturingMove(
118                     move.getDraught(),
119                     move.getNewXPosition() + move.xPositionMovement(),
120                     move.getNewYPosition() + move.yPositionMovement(),
121                     capturedDraught.get()
122                 );
123                 if (!isMoveBlocked(captureMove) && captureMove.
    moveInBounds()) {
124                     possibleMoves.add(captureMove);
125                 }
126             }
127         }else{possibleMoves.add(move);}
128     }
129 
130     return removeNonCapturingMovesIfCapturingMoveExists(
    possibleMoves);
131 }
132 
133 private boolean isMoveBlocked(Move move) {
134     for (Draught draught : draughtArrayList) {
135         if (draught.xPosition == move.getNewXPosition() && draught.
    yPosition == move.getNewYPosition()) {
136             return true;
137         }
138     }
139     return false;
140 }
141 
142 public ArrayList<Draught> draughtsWithPossibleMoves() {
143     return findAllPossibleMoves()
144         .stream()
145         .map(Move::getDraught)
146         .distinct()
147         .collect(Collectors.toCollection(ArrayList::new));
148 }
149 
150 public void selectMove(Move move){
151     Draught moveDraught = getDraughtFromPosition(move.getDraught().
    xPosition, move.getDraught().yPosition);
152 
153     moveDraught.setxPosition(move.getNewXPosition());
154     moveDraught.setyPosition(move.getNewYPosition());
155 
156     if (move instanceof CapturingMove) {
157 
158         Draught capturedDraught = getDraughtFromPosition(((
    CapturingMove) move).getCapturedDraught().getxPosition(), ((
    CapturingMove) move).getCapturedDraught().getyPosition());
```

```
159             draughtArrayList.remove(capturedDraught);
160
161             ArrayList<Move> moves = findPossibleMoves(moveDraught);
162             if (moves.stream().anyMatch(move1 -> move1 instanceof
    CapturingMove)) {
163                 if (!moveDraught.isDraughtOnKingsRow() || moveDraught.
    isCrowned()) {
164                     selectedDraught = moveDraught;
165                     isCurrentMultiStepMove = true;
166                     checkForWinner();
167                     return;
168                 }
169             }
170
171         }
172         moveDraught.crownDraughtIfPossible();
173
174         if (currentTurn == Colour.DARK){
175             currentTurn = Colour.LIGHT;
176         }else{
177             currentTurn = Colour.DARK;
178         }
179         selectedDraught = null;
180         isCurrentMultiStepMove = false;
181         checkForWinner();
182     }
183
184     private Draught getDraughtFromPosition(int x, int y) {
185         return draughtArrayList.stream().filter(draught -> draught.
    xPosition == x && draught.yPosition == y).findAny().get();
186     }
187
188     public Game successor(Move move) {
189         Game rtnGame = this.clone();
190
191         rtnGame.selectMove(move);
192         return rtnGame;
193     }
194
195     private boolean checkForWinner() {
196
197         if (draughtArrayList.stream().noneMatch(draught -> draught.
    getColour() == Colour.LIGHT)) {
198             isGameComplete = true;
199             gameWinner = Colour.DARK;
200             return true;
201         }
202         if (draughtArrayList.stream().noneMatch(draught -> draught.
    getColour() == Colour.DARK)) {
```

```
203             isGameComplete = true;
204             gameWinner = Colour.LIGHT;
205             return true;
206         }
207         if (findAllPossibleMoves().isEmpty()) {
208             isGameComplete = true;
209             if (currentTurn == Colour.LIGHT) {
210                 gameWinner = Colour.DARK;
211             } else if (currentTurn == Colour.DARK) {
212                 gameWinner = Colour.LIGHT;
213             }
214             return true;
215         }
216         return isGameComplete;
217     }
218 
219 
220     public boolean isCurrentMultiStepMove() {
221         return isCurrentMultiStepMove;
222     }
223 
224     @Override
225     protected Game clone() {
226         Game clone = null;
227         try {
228             clone = (Game) super.clone();
229             clone.draughtArrayList = new ArrayList<>();
230             for (Draught draught : draughtArrayList) {
231                 clone.draughtArrayList.add(draught.clone());
232             }
233             clone.isGameComplete = isGameComplete;
234             clone.gameWinner = gameWinner;
235             clone.currentTurn = currentTurn;
236             clone.isCurrentMultiStepMove = isCurrentMultiStepMove;
237             clone.selectedDraught = selectedDraught == null ? null :
    selectedDraught.clone();
238         } catch (CloneNotSupportedException ignored) {}
239         return clone;
240     }
241 
242 }
```

D. Move

D.1. Move

```
1 package moves;
2 
```

```
3  import game.Draught;
4
5  public class Move {
6     Draught draught;
7     int newXPosition;
8     int newYPosition;
9
10    public Draught getDraught() {
11       return draught;
12    }
13
14    public void setDraught(Draught draught) {
15       this.draught = draught;
16    }
17
18    public int getNewXPosition() {
19       return newXPosition;
20    }
21
22    public void setNewXPosition(int newXPosition) {
23       this.newXPosition = newXPosition;
24    }
25
26    public int getNewYPosition() {
27       return newYPosition;
28    }
29
30    public void setNewYPosition(int newYPosition) {
31       this.newYPosition = newYPosition;
32    }
33
34
35
36    public Move(Draught draught, int newXPosition, int newYPosition){
37       this.draught = draught;
38       this.newXPosition = newXPosition;
39       this.newYPosition = newYPosition;
40    }
41
42    public boolean moveInBounds() {
43       return 0 <= newXPosition && newXPosition < 8 && 0 <=
     newYPosition && newYPosition < 8;
44    }
45
46    public int xPositionMovement(){return newXPosition - draught.
     getxPosition();}
47    public int yPositionMovement(){return newYPosition - draught.
     getyPosition();}
48 }
```

D.2. Capturing Move

```
1  package moves;
2
3  import game.Draught;
4
5  import java.util.ArrayList;
6
7  public class CapturingMove extends Move {
8
9     Draught capturedDraught;
10    ArrayList<Move> possibleFollowingMoves;
11
12    public Draught getCapturedDraught() {
13       return capturedDraught;
14    }
15
16    public void setCapturedDraught(Draught capturedDraught) {
17       this.capturedDraught = capturedDraught;
18    }
19
20    public ArrayList<Move> getPossibleFollowingMoves() {
21       return possibleFollowingMoves;
22    }
23
24    public void setPossibleFollowingMoves(ArrayList<Move>
      possibleFollowingMoves) {
25       this.possibleFollowingMoves = possibleFollowingMoves;
26    }
27
28    public CapturingMove(Draught draught, int newXPosition, int
      newYPosition, Draught capturedDraught) {
29       super(draught, newXPosition, newYPosition);
30       this.capturedDraught = capturedDraught;
31    }
32
33
34 }
```