```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;

/*
Student Name: Ben Grandy
Student Number: 6090484

Program: This program was written for COSC 3P71 Artificial Intelligence: Assignment #1 October 18th, 2021 Fall Brock University
This program was written in java using the IDE Intellij.
The main goal for this program was to create an Artificial Intelligence for a game of Connect 4. The AI must run on minimax, an algorithm
for determining optimal plays in a turn based game. My minimax function follows pseudocode that was covered in lecture and tutorial.
My heuristic is a combination of a few things. If the AI can win, win the game. If the human player has a chance to win the game then set
the value to the lowest possible. If the extreme cases aren't found then calculate a value of how 'winning' the board is for the AI.
The Heuristic does this by first calculating how many 2 sequential pieces it can find. It then calculates how many 2's sequentially the
opponent has. It does the same thing with 3 sequential pieces. It then adds points for 3 sequential pieces and 2 sequential pieces and reduces points
for the opponents sequential pieces. If the value of the board is low the AI priorities the middle of the board. This helps the AI
choose a move at the beginning of the game.
*/

public class Connect4 {
    String[][] board = new String[6][7];
    boolean player = false;
    private static Scanner s = new Scanner(System.in);

```

```
26
27      public Connect4() {
28          fillBoard();
29          printBoard();
30          play();
31
32      }
33
34      //This method structures the program. Provides
    prompts for the user's columns and notifies the user
    when the column is full.
35
36      public void play() {
37          int depth;
38          System.out.println("How deep do you want the
    AI to search (how deep to ply): ");
39          depth = s.nextInt();
40
41          while (true) {
42
43              System.out.print("Enter a column: ");
44              int play = s.nextInt();
45              if (play < 0 | play > 6) {
46                  System.out.println("Column choices
    are between 0-6");
47                  play();
48              }
49              if (filledColumnCheck(play)) {
50                  System.out.println("This column is
    full.");
51                  play();
52              }
53              placePiece(play, player);
54              printBoard();
55
56              //if that play didn't win switch turns
57              if (checkForWin(player, board)) {
58                  break;
59              } else {
60                  if (player) {
61                      player = false;
62                  } else {
63                      player = true;
64                  }
```

```
65              }
66              //get all possible next moves
67              ArrayList<String[][]> possibleStates =
    new ArrayList<>();
68              possibleStates = getPossibleStates(board
    , player);
69
70              //for all possible next moves take max
    score from minimax
71              //since we pass the minimax algorithm
    all possible moves for the AI, the next set of moves
     would be the
72              //humans moves therefore, the original
    call will be passed as false.
73              int bestScore = Integer.MIN_VALUE;
74              int minimaxScore, bestAIMove = 0;
75
76              for (int i = 0; i < possibleStates.size
    (); i++) {
77                  minimaxScore = minimax(
    possibleStates.get(i), depth, false);
78                  if (minimaxScore > bestScore) {
79                      bestScore = minimaxScore;
80                      bestAIMove = i;
81                  }
82              }
83              placePiece(bestAIMove, player);
84              printBoard();
85
86              System.out.println("The AI has chosen
    column: " + bestAIMove);
87              //if that play didn't win switch turns
88              if (checkForWin(player, board)) {
89                  break;
90              } else {
91                  if (player) {
92                      player = false;
93                  } else {
94                      player = true;
95                  }
96              }
97
98          }
99          //if the while loop has broken then the game
```

```
 99  has been won
100          if (player) {
101              System.out.println("AI has won!");
102          } else {
103              System.out.println("You have won!");
104          }
105      }
106
107      //This method follows the minimax algorithm
    covered in class and tutorial to find optimal moves.
108      public int minimax(String[][] possibleState, int
     depth, boolean maximizingPlayer) {
109
110          int value;
111          ArrayList<String[][]> childStates;
112          boolean won = false;
113          if (possibleState == null)
114              return Integer.MIN_VALUE;
115
116          //if the depth is 0 or the node is a
    terminal node
117          if (depth == 0 | checkForWin(
    maximizingPlayer, possibleState)) {
118              //return the value the heuristic returns
119              value = heuristic(possibleState,
    maximizingPlayer);
120              return value;
121          }
122
123          if (maximizingPlayer) {
124              value = Integer.MIN_VALUE;
125              childStates = getPossibleStates(
    possibleState, maximizingPlayer);
126              for (int i = 0; i < childStates.size();
    i++) {
127                  if (childStates.get(i) != null) {
128                      //find the max value of the
    nodes children
129                      value = Math.max(value, minimax(
    childStates.get(i), depth - 1, false));
130                  }
131              }
132              return value;
133
```

```
134         } else {
135             value = Integer.MAX_VALUE;
136             childStates = getPossibleStates(
    possibleState, maximizingPlayer);
137             for (int i = 0; i < childStates.size();
    i++) {
138                 //find the min value of the nodes
    children
139                 value = Math.min(value, minimax(
    childStates.get(i), depth - 1, true));
140             }
141             return value;
142         }
143     }
144 
145     //this method calculates the score of the board
    state for the AI. Higher the value the better the
    board state is for
146     //the AI
147     public int heuristic(String[][] possibleState,
    boolean maximizingPlayer) {
148         int value;
149         //if the board state is won for the human
    the board state is worst possible scenario
150         if (checkForWin(player, possibleState)) {//
    if the AI can win set it to the highest possible
    value.
151             value = Integer.MAX_VALUE;
152             return value;
153         } else if (checkForWin(!player,
    possibleState)) {//if the human has a chance to win
    set it to the lowest possible value
154             value = Integer.MIN_VALUE;
155             return value;
156         }
157 
158         int countThree = 0, countTwo = 0,
    opponentThree = 0, opponentTwo = 0;
159 
160         //count the amount of two and three in a row
     each person has and adjust the value accordingly.
161         countTwo = countInARow(player, possibleState
    , 2);
162         countTwo *= 250;
```

```
163         opponentTwo = countInARow(!player,
    possibleState, 2);
164         opponentTwo *= -10;
165
166         countThree = countInARow(player,
    possibleState, 3);
167         countThree *= 1000;
168         opponentThree = countInARow(!player,
    possibleState, 3);
169         opponentThree *= -100;
170
171         //if the value is very low prioritize the
    middle of the board,
172         value = (countThree + countTwo +
    opponentThree + opponentTwo);
173         if (value < 30 & value > -30) {
174             value += ((countMiddle(player,
    possibleState)) * 10);
175
176         }
177
178         return value;
179     }
180
181
182     //this method counts how many middle tiles the
    AI could obtain and assigns a value accordingly.
    This method returns the value
183     public int countMiddle(boolean player, String
    [][] board) {
184         int count = 0;
185         for (int i = 5; i >= 3; i--) {
186             for (int j = 2; j < 5; j++) {
187                 if (board[i][j].equals("O")) {
188                     if (i == 5 & j == 3) {
189                         count += 50;
190                     } else if (i == 5) {
191                         count += 30;
192                     } else if (i == 4) {
193                         count += 5;
194                     } else {
195                         count++;
196                     }
197                 }
```

```
198             }
199         }
200         return count;
201     }
202 
203     //this method gets a board and a player and
    returns all possible moves for the player/.
204     public ArrayList<String[][]> getPossibleStates(
    String[][] board, boolean player) {
205         ArrayList<String[][]> possibleStates = new
    ArrayList<>();
206         String[][] tmpBoard = new String[6][7];
207 
208         for (int i = 0; i < 7; i++) {
209             tmpBoard = copyArray(board);
210             if (!filledColumnCheck(i)) {
211                 possibleStates.add(placeTmpPiece(i,
    player, tmpBoard));
212             } else {
213                 possibleStates.add(null);
214             }
215         }
216         return possibleStates;
217     }
218 
219     //Make a copy of the board for possibleStates
220     public String[][] copyArray(String[][] board) {
221         String[][] tmpBoard = new String[6][7];
222         for (int i = 0; i < board.length; i++) {
223             for (int j = 0; j < board[i].length; j
    ++) {
224                 tmpBoard[i][j] = board[i][j];
225             }
226         }
227         return tmpBoard;
228     }
229 
230     //check for a win horizontally and return true
    if found
231     public boolean checkForHorizontal(boolean player
    , String[][] board) {
232         int count;
233         boolean won = false;
234         for (int i = 5; i >= 0; i--) {
```