CSE 4082

Artificial Intelligence Project 2 – Connect Four Game

Barış Hazar – 150118019

Ulaş Deniz Işık – 150118887

Implementation Details

1. Classes

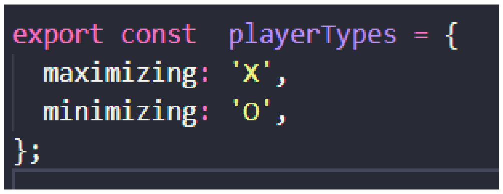
1.1 GameBoard Class

In the GameBoard class, we only have a board 2d array which represents the board of our game. And it has several methods to check for draw, winning conditions etc. Also, it has a method for generating children boards from it by having a player type as input.

1.2 Player Class

Player class, is the superclass of the HumanPlayer and AIPlayer classes, it has GameBoard class and a playerType as its attributes. GameBoard is necessary because the players need to interact with the game board to play their turn.

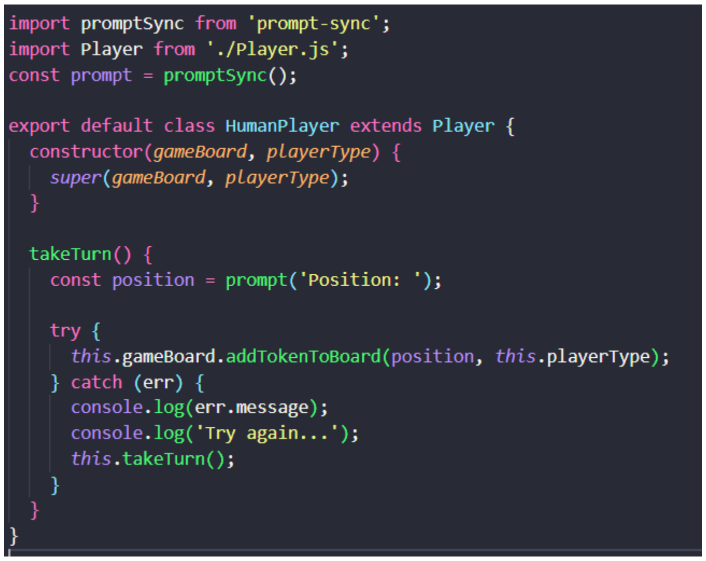
Note: In our program, maximizing player always has the token 'X', and the minimizing player always has the token 'O'.



```
export const  playerTypes = {
  maximizing: 'X',
  minimizing: 'O',
};
```

1.3 HumanPlayer Class

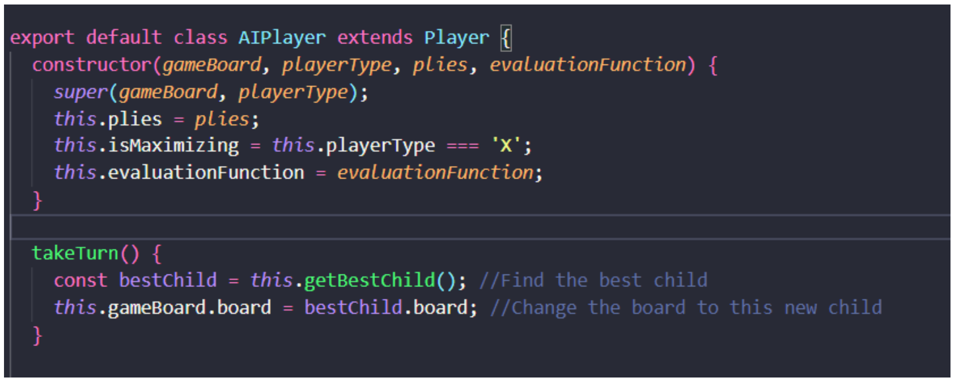
HumanPlayer Class has a simple takeTurn method which asks a user for the position to play, and calls its gameBoard's addTokenToBoard method to add its token to the board.



```
import promptSync from 'prompt-sync';
import Player from './Player.js';
const prompt = promptSync();

export default class HumanPlayer extends Player {
  constructor(gameBoard, playerType) {
    super(gameBoard, playerType);
  }

  takeTurn() {
    const position = prompt('Position: ');

    try {
      this.gameBoard.addTokenToBoard(position, this.playerType);
    } catch (err) {
      console.log(err.message);
      console.log('Try again...');
      this.takeTurn();
    }
  }
}
```

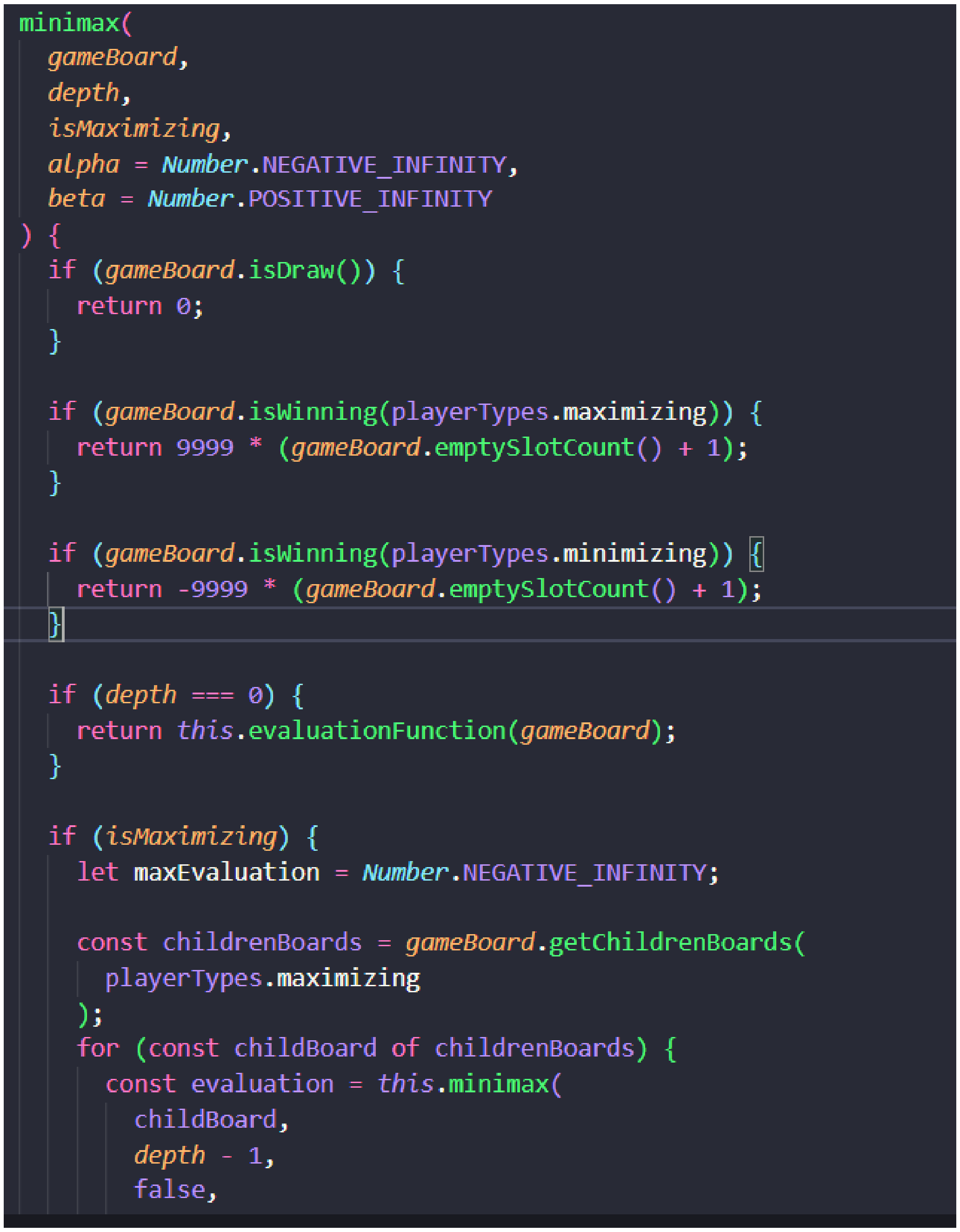
1.4 AIPlayer Class

AIPlayer class shares the takeTurn method with HumanPlayer class. It has evaluation Function and number of plies attributes in addition to the HumanPlayer class .



```
export default class AIPlayer extends Player {
  constructor(gameBoard, playerType, plies, evaluationFunction) {
    super(gameBoard, playerType);
    this.plies = plies;
    this.isMaximizing = this.playerType === 'X';
    this.evaluationFunction = evaluationFunction;
  }

  takeTurn() {
    const bestChild = this.getBestChild(); //Find the best child
    this.gameBoard.board = bestChild.board; //Change the board to this new child
  }
```

And this is the class where we implemented the minimax function. In the takeTurn method, it gets the best child by calling minimax on each of them and then changes the gameboard to the best child (child with max or min score according to isMaximizing property).



```
minimax(
  gameBoard,
  depth,
  isMaximizing,
  alpha = Number.NEGATIVE_INFINITY,
  beta = Number.POSITIVE_INFINITY
) {
  if (gameBoard.isDraw()) {
    return 0;
  }

  if (gameBoard.isWinning(playerTypes.maximizing)) {
    return 9999 * (gameBoard.emptySlotCount() + 1);
  }

  if (gameBoard.isWinning(playerTypes.minimizing)) {
    return -9999 * (gameBoard.emptySlotCount() + 1);
  }

  if (depth === 0) {
    return this.evaluationFunction(gameBoard);
  }

  if (isMaximizing) {
    let maxEvaluation = Number.NEGATIVE_INFINITY;

    const childrenBoards = gameBoard.getChildrenBoards(
      playerTypes.maximizing
    );
    for (const childBoard of childrenBoards) {
      const evaluation = this.minimax(
        childBoard,
        depth - 1,
        false,
```

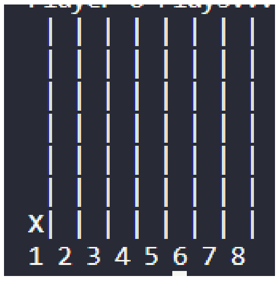
The reason for assigning 9999 * empty slot count for winning terminal nodes is to go for the closest winning state by giving it a higher value. (or -9999 for minimizing player)

2. Evaluation Heuristics

2.1 Completable Score

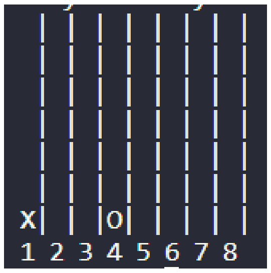
In this heuristic function, we consider the completability of each token to four. Because when you have combined tokens of four, you win the game. So, a token that is completable to four in all four directions (horizontal, vertical, diagonal and antidiagonal) is better than a token which is surrounded by enemy tokens and can not be a part of a solution.

For example, in the below figure, the completable score is 3. Because ‘X’ is completable in vertical, horizontal, diagonal but not the antidiagonal. So heuristic assigns the score of 3 to this position.



```
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
X| | | | | | | |
1 2 3 4 5 6 7 8
```

Now let’s say the minimizing player (‘O’) played for position 4, as you can see from the figure below.



```
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
X| | |O| | | | |
1 2 3 4 5 6 7 8
```

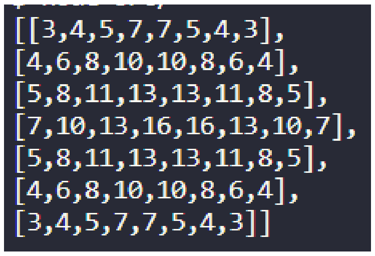
In this new position, the completable score for player X decreased to 2 from 3, because its completability in horizontal direction no longer exists. And the player ‘O’ has a completable score of 4, because it can be completed in all four directions. So, the total completable score becomes 2 – 4 = -2. Which favors the minimizing player (Player ‘O’).

2.2 Centrality Score

The focus of centrality score is not the completability of the tokens like the first heuristic, but the positions of the tokens on the board. We wrote a function which returns a 2d array calculating the total number of different solutions that includes each position.

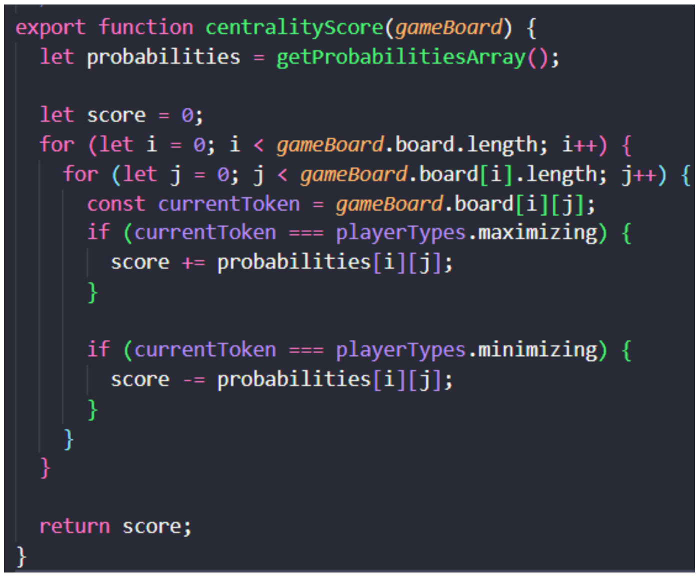
The functions we used to calculate these values are so similar to the functions for checking the winning condition. But unlike checking for winning, they increment the positions that they pass by 1. So, the positions which are closer to the center have a higher score than those on the sides.

When we use this technique, we get the following probabilities array for a 7X8 board. As you can see, a total of 16 different solutions passes through the center of the board. And only 3 possible solutions include the corners. And these numbers are calculated using the functions we wrote, we didn't statically write the below array ourselves, so our functions can calculate for different board sizes too.



```
[[3,4,5,7,7,5,4,3],
[4,6,8,10,10,8,6,4],
[5,8,11,13,13,11,8,5],
[7,10,13,16,16,13,10,7],
[5,8,11,13,13,11,8,5],
[4,6,8,10,10,8,6,4],
[3,4,5,7,7,5,4,3]]
```

The calculation of the scores is straightforward. For each token of the maximizing player, we increment the score by the corresponding probability on the above array. And for each of the minimizing tokens, we do the opposite by subtracting.



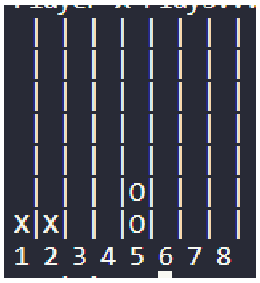
```
export function centralityScore(gameBoard) {
  let probabilities = getProbabilitiesArray();

  let score = 0;
  for (let i = 0; i < gameBoard.board.length; i++) {
    for (let j = 0; j < gameBoard.board[i].length; j++) {
      const currentToken = gameBoard.board[i][j];
      if (currentToken === playerTypes.maximizing) {
        score += probabilities[i][j];
      }

      if (currentToken === playerTypes.minimizing) {
        score -= probabilities[i][j];
      }
    }
  }

  return score;
}
```

2.3 Completable Centrality Score (Combination of the First Two)

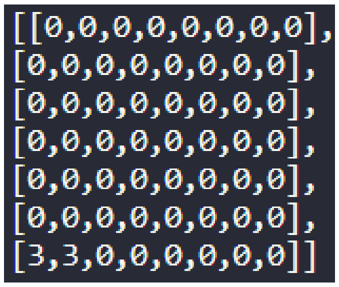
The final evaluation heuristic we have is the combination of the first two heuristics.

We get the completable array for both maximizing and minimizing players which is a 2d array that has the same dimensions as the game board. And for each of the tokens, it has their completable scores.

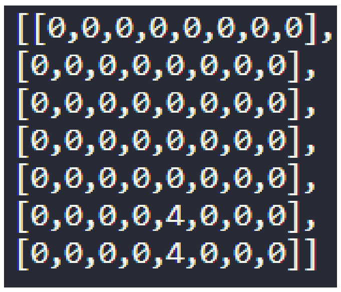
You can see from the below figures that for the board state in the first figure the completability array for maximizing players is shown in the middle. And we have a similar array for minimizing player on the right to calculate the total score.



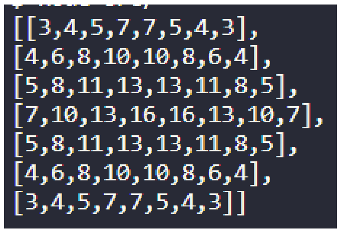
```
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | | | | | |
 | | | |O| | | |
x|x| | |O| | | |
1 2 3 4 5 6 7 8
```



```
[[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[3,3,0,0,0,0,0,0]]
```



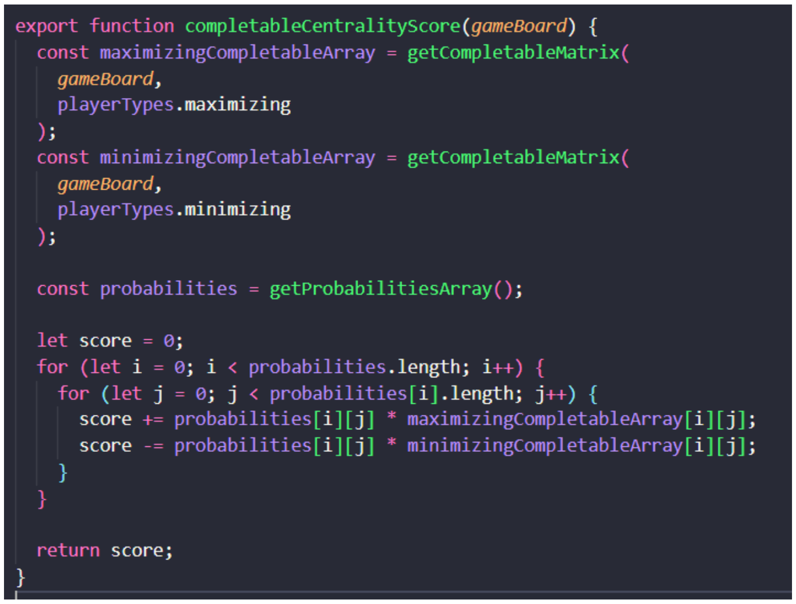
```
[[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,4,0,0,0],
[0,0,0,0,4,0,0,0]]
```



```
[[3,4,5,7,7,5,4,3],
[4,6,8,10,10,8,6,4],
[5,8,11,13,13,11,8,5],
[7,10,13,16,16,13,10,7],
[5,8,11,13,13,11,8,5],
[4,6,8,10,10,8,6,4],
[3,4,5,7,7,5,4,3]]
```

For each element in the maximizing completable array, we multiply them by the corresponding index of the probabilities array we introduced in 2.2 and add the result to the score. And similarly, for each element in the minimizing completable array we multiply and subtract from the score.

So, the score for the above board would be (3 * 3) + (3 * 4) – (4 * 7) – (4 * 10) = -47. As you might guess -47 is a score which favors the minimizing player (player ‘O’). Overall, this is the best evaluation heuristic among the three, because it combines the strong parts of the first two heuristics.



```
export function completableCentralityScore(gameBoard) {
  const maximizingCompletableArray = getCompletableMatrix(
    gameBoard,
    playerTypes.maximizing
  );
  const minimizingCompletableArray = getCompletableMatrix(
    gameBoard,
    playerTypes.minimizing
  );

  const probabilities = getProbabilitiesArray();

  let score = 0;
  for (let i = 0; i < probabilities.length; i++) {
    for (let j = 0; j < probabilities[i].length; j++) {
      score += probabilities[i][j] * maximizingCompletableArray[i][j];
      score -= probabilities[i][j] * minimizingCompletableArray[i][j];
    }
  }

  return score;
}
```