Reversi using Minimax and Aplha-Beta Pruning

CS621 Project

Anshul Gupta (16305R001)
Himanshu Agarwal (163050001)

High Level Structure

The code is divided into 3 files: `client.py`, `board.py` and `ai.py`.

client.py

`client.py` is responsible for establishing the connection with the server and interacting with it. It has two modes: *Human v/s Computer* and *Computer v/s Computer*.

If the former is chosen, it gives a prompt for entering the desired moves. If the latter is chosen, it asks for an *Intelligence Level* which can take values from $1 - 4$ (4 being the highest).

It automatically terminates the game if the board becomes full or both server and client runs out of moves.

board.py

`board.py` manages all the operations on the board. These include *printing the board*, *updating the board*, *finding the legal moves* and *validating the moves*.

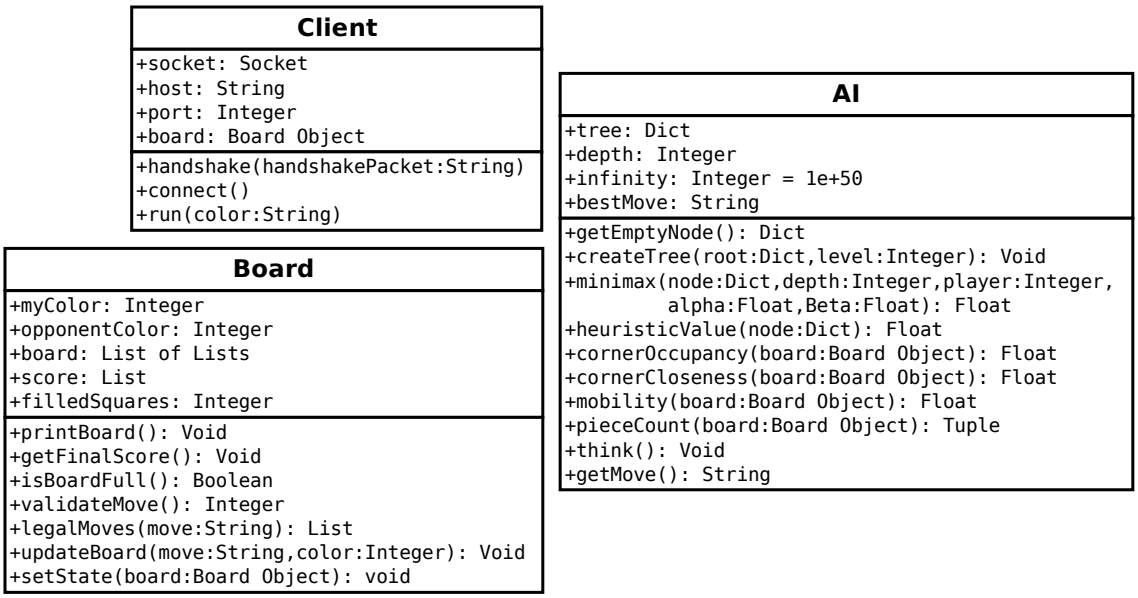
This file cannot be executed directly but imported in `client.py` and `ai.py` to perform board operations on the objects of this class.

ai.py

`ai.py` handles all the intelligent operations. AI class takes input as the current state of the board and enumerates all possible moves for the player and opponent for the specified level (default = 4).

It then evaluates the situation of the game and finds the best possible move for the player using a heuristic function. It optimizes the searching using *Aplha-Beta Pruning*.

This file also cannot be executed directly but imported in `client.py` to generate the tree and get the best move.



Heuristic Used

Heuristic is calculated based on six factors some of which are static and some are dynamic.

pieceValue

It is calculated by counting the pieces of both players. If a player has more pieces than other, it has more advantage.

```
if myPieces > opponentPieces:
  pieceValue = (100 * myPieces) / (myPieces + opponentPieces)
elif myPieces < opponentPieces:
  pieceValue = (-100 * opponentPieces) / (myPieces + opponentPieces)
```

diskSquares

It is a static factor. Each square of the board has a integer value (+ve or -ve). If a player's piece is on a square, the value of that square is added to it's points, if opponent's piece is on a square, the value of that square is subtracted from the points.

frontierValue

Frontier pieces are those which are on the perimeter (having alteast one empty neighbor). This is a negative factor.

```
if myFtrPieces > opponentFtrPieces:
  FrontierValue = (-100 * myFtrPieces) / (myFtrPieces + opponentFtrPieces)
elif myFtrPieces < opponentFtrPieces:
  FrontierValue = (100 * opponentFtrPieces) / (myFtrPieces + opponentFtrPieces)
```

cornerValue

Corner piece has an advantage that if it is occupied, it can never be flanked. So if a player has pieces on corners, they will have a positive effect on the player.

```
cornerValue = 25 * (myCornerPieces - opponentCornerPieces)
```

cornerClosenessValue

The pieces which are the immediate neighbors of the corner, they have a disadvantage that they can be flanked easily and the opponent will gain a corner position. This is a negative factor.

```
cornerClosenessValue = -12.5 * (myPieces - opponentPieces)
```

mobilityValue

Mobility is the number of moves a player has. More the number of moves, more advantage a player will have.

```
if myMoves > opponentMoves:
  return (100 * myMoves) / (myMoves + opponentMoves)
elif myMoves < opponentMoves:
  return (-100 * opponentMoves) / (myMoves + opponentMoves)
```

Heuristic Value

The final heuristic value is calculated using:

```
heuristicValue = (10 * diskSquares)
  + (10 * pieceValue)
  + (74.396 * frontierValue)
  + (801.724 * cornerValue)
  + (382.026 * cornerClosenessValue)
  + (78.922 * mobilityValue)
```