**Othello game**

These functions are provided for you in order to help you in implementing Othello game strategy:

**GameBoard.java**  contains several methods:

- public GameBoard(): creates and initializes the board with the four initial coins at the center positions.

- public GameBoard clone(): clones a GameBoard

- public void setCoin(Move m, int playerID): associates the playerID with the position on the GameBoard (where a coin is placed by Move).There are two constructors of the class Move: (1) public Move () and (2) public Move (int i, int j), where i and j represent the position on the board.

- public void addCoin(Move m, int playerID): adds the new coin for player playerID at the position defined in Move. It also reverses the intermediate positions (according the rules of the game) using method reverseCoins(m, playerID);

Methods to get current statistics:

- public int getCoinCount(int playerID): get number of coins of the current player

- public int getEdgeCoinCount(int playerID): get number of the edge coins of the current player (edge coins are important for the strategy of the game)

- public int getCornerCoinCount(int playerID): get number of corner coins of the current player (corner coins are stable: if obtained they can not be taken away)

- public int getPlayerIDAtPos(int line, int column):gets the playerID at a certain position

Method to display game board:

- public void displayGameBoard(): displays the Game Board

Method to get all possible moves:

- public ArrayList<Move> getPossibleMoves(int playerID): gets all possible moves of playerID

Methods to reverse coins:

- private void reverseCoins(Move move, int currentID): reverses the coins after the Move according to the rules of the game

- private void testDirection(int line, int column, int currentID, int dLine, int dColumn, boolean ennemyFound):

- private boolean reverseDirection(int line, int column, int currentID, int dLine, int dColumn, boolean ennemyFound): dLine and dColumn are increments (not absolute coordinates).

Methods to check if move is possible:

- private boolean isMovePossible(int line, int column, int playerID): checks all 8 directions (using the method checkDirections) from the current position to see if the move is possible, returns true or false.

- private boolean checkDirection(int line, int column, int playerID, int dLine, int dColumn, boolean ennemyFound): check if move is possible by considering only one direction. Parameters dLine and dColumn are increments and they are usually set to represent change of one step/position such as (-1,1), (-1,0), (1,1) etc...The function recursively calls itself by incrementing one step in the specified direction until the same color/player coin is found and then it returns true, otherwise returns false.

**Move.java** contains the class Move with two constructors: (1) public Move () and (2) public Move (int i, int j), where i and j represent the position on the board.

**Node.java** contains a game state with his evaluation and his child nodes (possible actions at the current state). It also contains the following methods:

* setEvaluation(int evaluation)
* getEvaluation()
* setMove(Move move)
* getMove()
* public ArrayList<Node> getChildNodeList()
* public void addChildNode(Node childNode): this method is used for adding a movement as child node to the current state
* public boolean isLeaf()

**Joueur.java**: contains the class Joueir with the following variables: (1) protected int playerID;

(2) protected int depth (this is the depth of alpha-beta) ; (3) protected int size = 8 (this is the width of the search).

It has two constructors: (1) public Joueur() and (2) public Joueur(int depth, int playerID).

It also has the method: public Move nextPlay(Move move) which returns null.