**ESS101 Python Project- Othello**

Othello is a [strategy](https://en.wikipedia.org/wiki/Abstract_strategy_game) [board game](https://en.wikipedia.org/wiki/Board_game) for two players, played on an 8×8 chequered board. There are sixty-four identical game pieces called *disks* (often spelled "discs"), which are light on one side and dark on the other. Players take turns placing disks on the board with their assigned colour facing up. During a play, any disks of the opponent's colour that are in a straight line and bounded by the disk just placed and another disk of the current player's colour are turned over to the current player's colour.

The object of the game is to have the majority of disks turned to display your colour when the last playable empty square is filled.

**The Project**

The game has 3 parts, the menu, the GUI(the menu and the board) and the algorithm.

**The GUI**

* The menu
  + This file is run before the game starts
  + The game mode is decided from here
* The board
  + In the 2-Player mode, we’ve added a feature of displaying whose chance it is
  + The score display is common to all game modes.
  + Every mode has a certain endgame message
  + ‘Valid’ moves are displayed when one hovers the cursor over that particular square

**The Algorithm**

**1. The basic implementation**

* Checking for a valid move
  + This is necessary as if there is no valid move, the turn is passed to the next player.
  + If it’s black’s turn, take all the empty squares around white, and recursively check if it playing on the empty square is a valid move by going in the same direction as the white coin it was around.
  + Break the recursion if you come across the black square and store the indices of the white coins it passed through to get there. The ‘storing’ has been done with the help of a dictionary
  + That particular empty square is now a valid move.
  + This has been implemented with the help of the functions checkValidity(),recur() and check()
* Changing the coins
  + If the input move is valid, search the value of the key in the dictionary
  + Iterate through the list and turn the corresponding white coins into black
* Endgame
  + The game ends
    - When both the players don’t have valid moves.
    - When the board is full
    - When one player has a score of 0

**2. The AI**

* The AI has been divided into 3 parts – Easy , Medium and Hard
* Easy
  + This uses a greedy approach
  + A greedy approach is an approach which plays the best possible move at that point of time.
  + The computation for all the moves is already done by checkValidity().
* Medium
  + Edges and corners have a very high utility in this game. An edge can only be ‘sandwiched’ in one direction. A corner has an even higher utility as once you get a corner, it’s yours. It can also be used to ‘sandwich’ edges which in turn can be used to sandwich other pieces.
  + The greedy approach may not choose a corner every single time as corners have a potential utility.
  + This algorithm is also greedy if we give a potential ‘score’ to the corners and edges.
* Hard
  + Hard is actually hard to beat.
  + Hard takes things one step further. It ‘thinks’ one step ahead.
  + It uses an algorithm called the minimax algorithm.
    - The minimax algorithm has optimizations such as alpha-beta pruning and progressive deepening.
    - The above, combined with end game analysis, parallel computing and uneven tree development was used in the Deep Blue which beat Gary Kasparov in 1997.
  + The hard level algorithm however, uses the minimax at a depth of 1 level.
  + It also uses the medium algorithm to give the scores of each board on which minimax is then used and the move is played.