The game of Othello Design Doc

Goal:

The goal of this design doc is to provide data model, algorithm and pseudo code for Othello Game.

Basic Information:

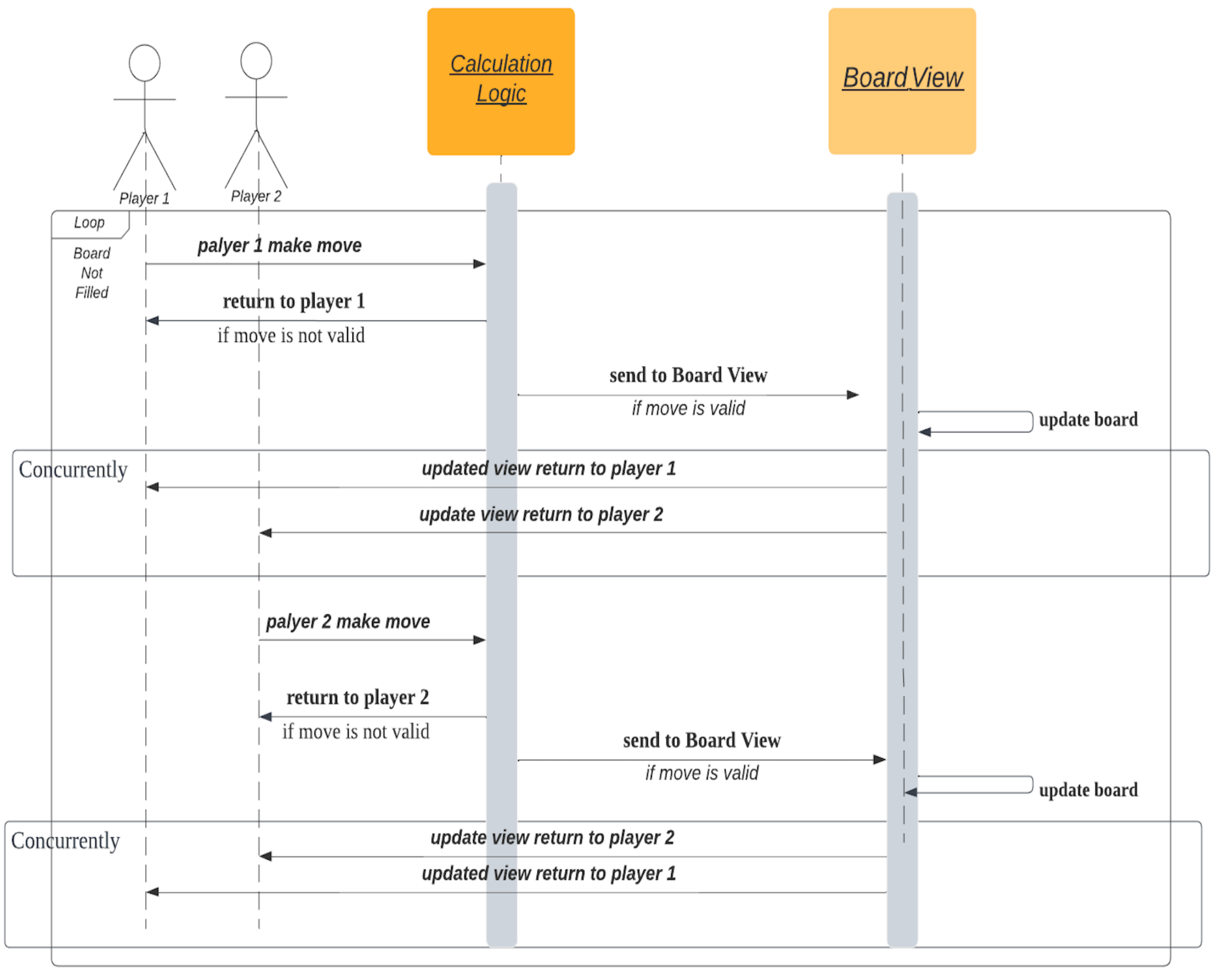
In this section, basic rules of the game will be discussed. Game is played on an 8 x 8 board. Player will have to play in turns by putting either a \`black\` or \`white\` tiles on the board, until eventually all board is filled up.

- Legal Moves:
  Legal move can be determined by rule as follows: a tile must be put in a position such that at least on opposing tile will be flipped.
  - a legal move needs to be treated as a starting point, we need to check the row of tiles starts from the starting point for up, down, left and right four directions. The move is legal when there is at least one row, we can flip an opposite tile.
  - in order to flip an opposite tile, the opposite tile has to be in the middle of two opposite tiles. For example, when we are to put a \`white\` tile, the \`X\` position is a legal position.

	White	Black	X
Black	Black	Black	
	X		

- Winning:
  After all board has been filled up, the player has more tiles of their color win.

User action Flow Chart:



- Player_1 makes move -> Controller.make_move( ) → Board.check_legal_move() -> return checking result to Controller
  → controller.update() if valid
  → controller.prompt_invalid_move() if invalid

- Player_2 makes move
  → if it's a user
  -> same logic as player_1
  → if it's a computer
  -> search for available slot for next move
  -> pick random slot if there is one (simplest solution, can apply algorithm to make most effective move. Some BFS or DFS algorithm needs to be used)

- Game Over

→ count tiles for each player
→ prompt user for winer

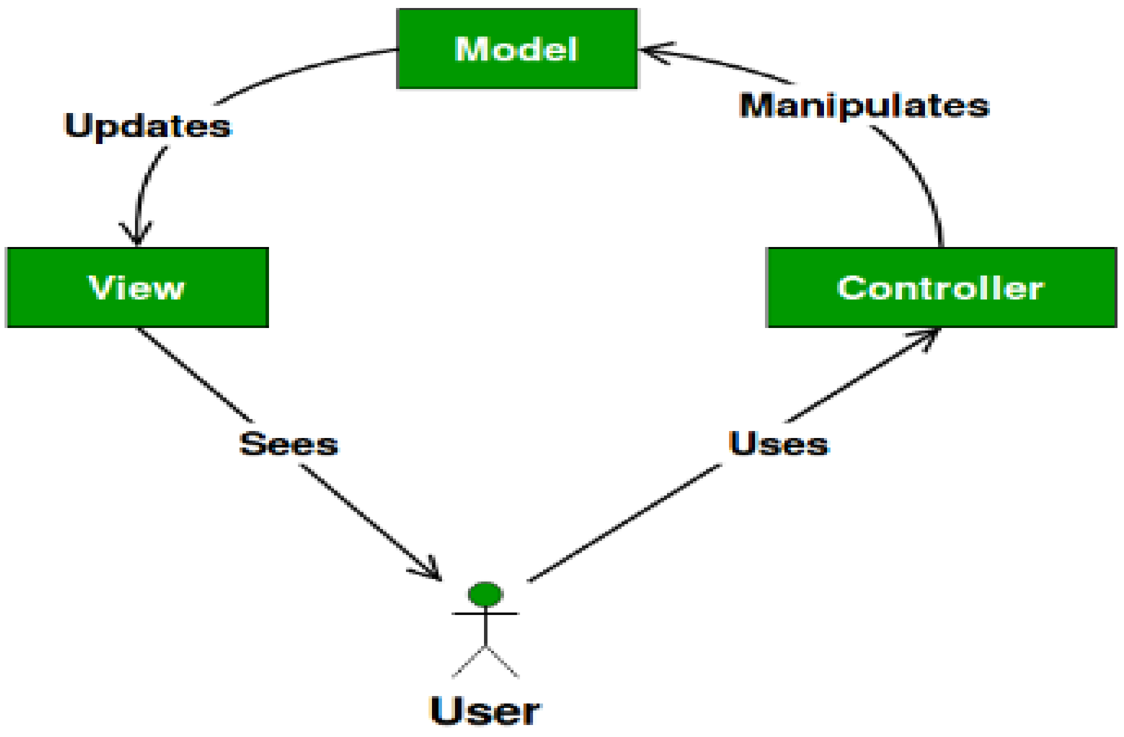
Objects:

Tile	
- tile_id	<string>
- color	<string>
- row_index	<Integer>
- col_index	<Integer>

Board	
- background_color	<string>
- width	<Integer>
- height	<Integer>
- update(tile,row, col)	void
- draw_board(width, height)	void
- calculate(row, col, tile)	[Integer,Integer]

Game	
- move_count	<Integer>
- winner	<Player>
- player_1	<Player>
- player_2	<Player>

Player	
- id	<string>
- name	<string>
- tile_count	<Integer>
- side	<string> enum: <white,black>



Classes / Functions pseudocode

```
class Player:
 self.id;
 self.name;
 self.side;
 self.tile_count;
 self.move_count;

class Game:
 self.player_1;
 self.player_2;
 self.winner;
 self.total_available_move;

  def is_game_end(self):
     return self.total_available_move ==
self.player_1.move_count+self.player_2.move_count;


# defined Game Controller
class  Game_Controller:
      self.Tile[ ];
      self.Game;
      self.Board;
      self.Player_1;
      Self.player_2;

      def make_move(self, row, col)
      def udpate_game(self)
      def update_Board(self, Tile)
      def update_player(self)
```

Data Structure

Array [ ] :
Storing Tiles by rows and cols, used for checking if the current move is a legal move
E.g.

```
      for tile_item in row_0:
            # check if there's a tile needs to be changed,
            # if no, then it's not a valid move.
            # if yes, it's a valid move and make updates
```

HashMap/ Dictionary

Used to associate Tile object information to a specific row column index.