I have three classes to represent my game:

Class1: Board

The Board object takes two arguments, the dimensions of the board, x coordinate and y coordinate, when there are no arguments passed it takes the dimensions as 5*9

The board class has 5 initial variables,

- X : The x dimension of the board
- Y : The y dimension of the board
- No_of_b: Keeps track of the number of black pieces on the board
- No_of_w: Keeps track of the number of white pieces on the board
- My_board: array of Slot objects, it stores information about each point on the x*y board.

The moment you create a board object the board gets initialized with all the initial values, sets where each array point can move to, place all the pieces on the board based on board dimensions

Class2: Slot

The slot object is used to represent each point on the board.

It has two initial variables:

- Possible_moves: an array of points where each point can move to on the Fanorona board.
- Color: The color of the piece on the board at that particular point

Class3: Move

The move object represents a coordinates.

It has two initial variables:

- X: the x-coordinate of any point
- Y: the y-coordinate of any point

The strategy class is the part of the code that contains the main logic,

The get_best_move() method of the strategy class gets all the pieces of a particular player that can be moved into an empty space and then calculates the best move.

The best move is calculated using the evaluation function.

The evaluation function: The evaluation function returns the maximum number of pieces of the opposite player that can be captured through withdrawal or approval. It calculates the number of pieces that have been captured and keeps a track of the maximum number of opponents pieces that can be captured by making a particular move.

Not keeping track of the maximum number of pieces a move can capture will give us a non-optimal solution.

My code does not take the coordinates from the player, instead it calculates the best move for each player at any point in time and updates the board.

Using a smaller depth value would never cause any player to win, in such a case we assume it to be a draw. Hence using a higher depth value is better.

Using an evaluation function always produces a better and an optimal solution.