List of Classes with Behaviors and Method Descriptions

Flow Chart

Board
+ Sets array of 9 spaces
+ Gets board state
+ Sets state of squares
+ Resets board

Player
+ Takes a marker (gets)
+ Holds a marker (sets)

Runner
+ Starts new game with two players and new board
+ Calls square setting method from Board class
+ Returns true if marker can be placed
    + Checks if there is any available square on the board
        + Returns true if a particular square on board is empty
        + Returns true if there is a winning combo on board
            + Returns the winning marker type in addition to (or instead of?) true
            + Change empty square integers to strings to allow for comparison
+ Returns false if board is full and game is tied
+ Gets turn(board) (Returns marker type whose turn it is)
        + Randomly assigns first move to X or O if board is empty
+ Restarts game
        + Queries human to play again
            - if input == 1, return start_game
            - elsif input == 2, exit_game
            -else, return “That is not a 1 or 2”
- Starts the game
        while true:
        - print board(@board)
        - place marker(io.get_square, whose_turn?(board))
        - check board for winner(board)

Input/Output
+ Returns (Need to change to puts) “It is __’s turn”
+ Gets restart input
+ Gets first player marker type
        - Ensures that first marker type is a marker type
        - Re-gets marker type if it is not

- def get_marker_type Queries Player to determine marker, either X or O
    - “What sort of marker would you like to use? (i.e. X or O)”
    - if input.length == 1 return input
    - elsif input.length != 1
        - puts “Your marker should be only one character” get_marker_type
    - end
+ Prints current board in 3 x 3 pattern
+ Gets (input) next square on board to be marked
    + Queries human for which square to place piece into on board
+ Prints winner

AI
- Determines best move
    + Complete winning triple
        + Looks for triples with two of the same plus one empty
        + Fills in empty with X if there are 2 X’s
        + Returns board position of blank to be filled
    + Blocks opponent’s winning move (inverse of “Determines best move”)
        + Looks for triples with two of the same plus empty for opponent
        + Fills in empty with X if there are 2 O’s
    - Creates a fork
        - Gets possible forks (opportunity to make two threats to win)
    - Blocks a fork (inverse of “Creates a fork”?)
    + Marks the center of board
        + If square 5 is empty, mark with marker
        + If square 5 is full, return nil (or something else?)
    + Marks opposite corner
        + If square 1 (if this evaluates to true square 1.to_i == 0, and @board.square_empty?(board, 9)) is filled, mark square 9, 3->7, 7->3, 9->1
    + Marks empty corner
        + If 1,3,7,9 empty, fill empty with marker
    + Marks empty side
        + If 2, 4,6,8 empty, fill empty with marker

Describing Runner

startup:

```
def start_game
        io.ask_for_width_of_board # Enter 3 for a 3x3 board or 4 for a 4x4 board
        board_size = io.get_size_of_board
        runner.set_width_of_board(board_size)
        @board.reset_any_size_board

-       io.ask_for_number_of_human_players
-       io.get_number_of_human_players

        runner.play_game

end

def play_game

        winner = board.winner_on_board?(board) #Maybe need to rename @board in runner?
        while winner == false
                it “should call:
                marker = whose_turn?(runner.number_of_empty_squares(@board))
                io.puts_turn(runner.whose_turn?) # It’s #{marker}’s turn
                output_board = @board.output_board(board)
                io.put_to_console(output_board)
                io.ask_for_square_to_mark? # Enter a square to mark

                square = io.get_square_to_mark
                if runner.square_empty?(square) == false
                        io.marker_error
                        io.ask_for_square_to_mark
                        io.get_square_to_mark
                else
                        runner.place_marker(square, marker)
                        runner.play_game

        io.puts_winner(winner)
        io.ask_to_restart? # Enter 1 to restart or any key to exit
        choice_to_restart = io.get_input
        if restart?(choice_to_restart) == 1
                runner.start_game

end
```