**CS6613 Final Project –Checker Game**

Youxing Gao yg1497

**Instruction:**

In order to play this game, the only thing you need to do is opening the test.html file in any browser. I would recommend you use google chrome, in this case you can inspect every computer move with google console tool.

**Design and Program:**

There are three files in total for this project, one is test.html, another is checkerGame.js, and the last one is checkerGame.css. I used css and html to implement the UI part conveniently, and used javascript top manipulate the DOM and implement the alpha-beta search algorithm.

UI: after open the html file in chrome you can see the interface as shown below:

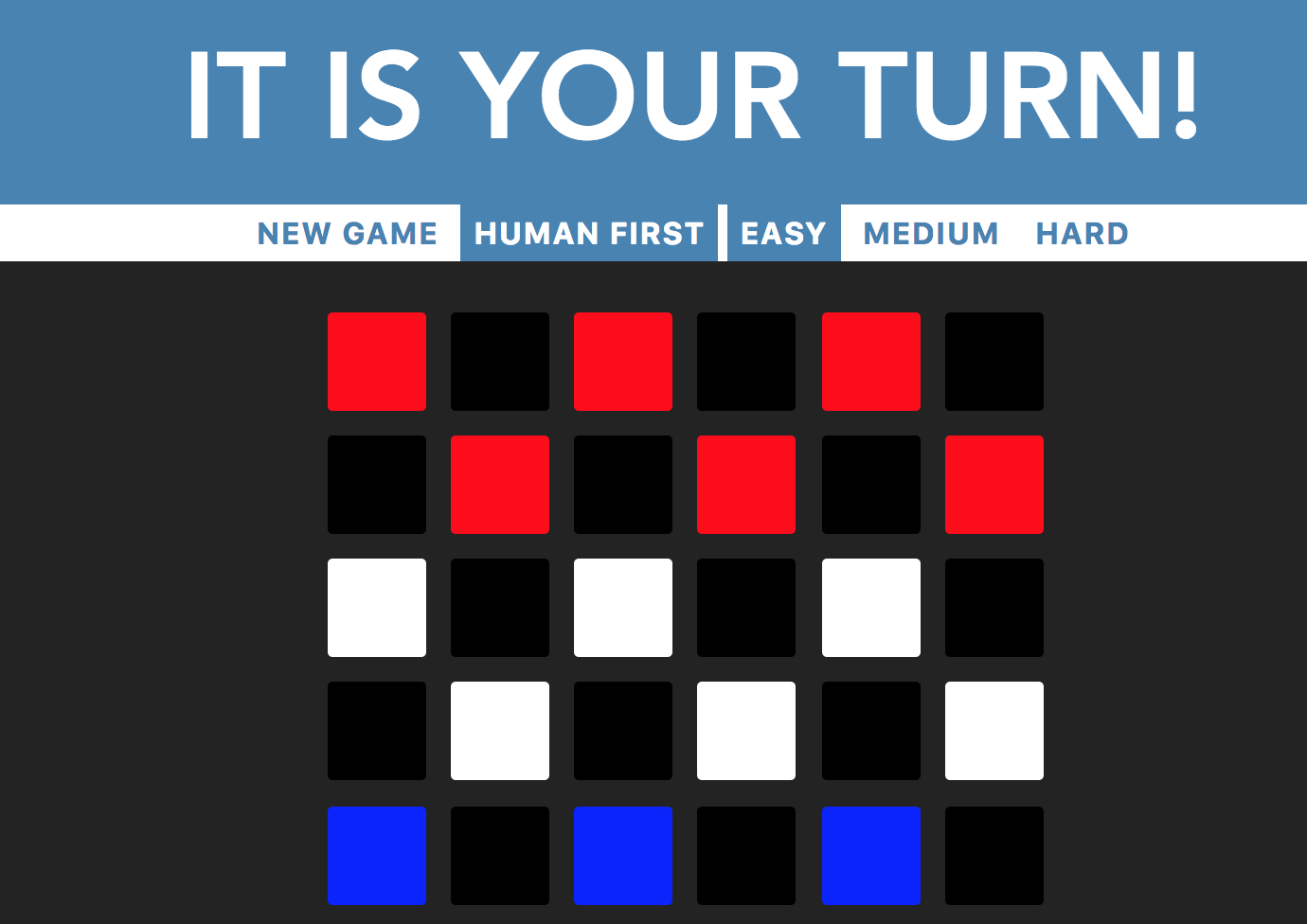


figure1

The first line is a message window, which is used to tell human player what he needs to do, and the result of the game. And the next line contains five buttons, clicking ‘new game’ button means restart the game, selecting ‘human first’ button means human goes first otherwise computer goes first, and the last three buttons can be used to choose hard level.

For the board part, red means human player’s pieces, blue means computer’s pieces, white means empty places, and black means unreachable squares. After human selected a red piece, the orange squares as shown below are the available moves for that piece. After clicked any of these five buttons, the game will restart.

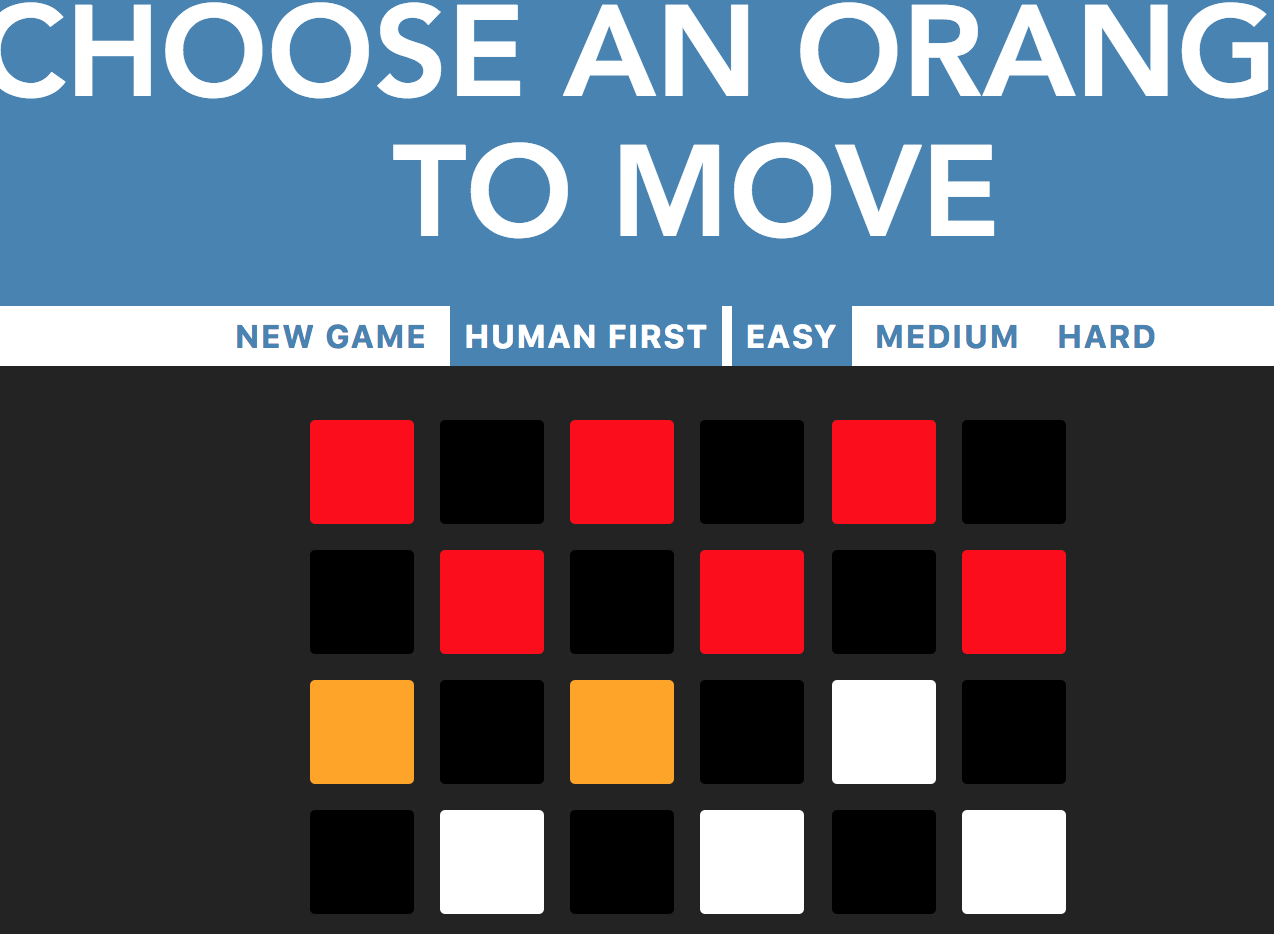


figure 2

I used a two-dimensional array to store the board information, and for each computer move I will copy the current board and use this copy to do the alpha-beta search.

**Terminal states and utility values:**

1. The number of red piece is 0, and the number of blue piece is not zero, return -1, which means computer win the game, and game over immediately.
2. The number of blue piece is 0, and the number of red piece is 0, return 1, which means human win the game, and game over immediately.
3. Both of players have no more moves, who left more pieces, who win. If they left the same number of pieces, the game is draw. If human win return utility value -100000, if computer win return utility value 100000.

**Evaluation function:**

When the alpha-beta search tree reaching its limit level, if there is no more jump moves for the computer, then use evaluation function to return the utility value.

1. Position value difference: For each computer piece return its position values as 6 minus its row number, for each human piece return its position value as its row number plus 1. Sum up all the position value for red pieces and blue pieces respectively, and get the get the difference position value by using sum of blue (computer player) pieces position value minus sum of red(human) pieces position value.
2. Pieces difference: The number of blue(computer) piece minus the number of red (human) pieces.
3. King difference: The number of blue(computer) kings minus the number of red (human) kings.
4. The weight for position value difference, king difference, and pieces difference is 1, 10, and 100 respectively.

**Hard Level:**

I used a variable called hard level to store the difficulty information. The easy level which is a default setting and represented by -1, is implemented by selecting the available next move randomly. The medium level is represented by number 2, which means when doing the alpha-beta search, the computer will search 2 levels down in the tree. The hard mode is represented by number 8, which means when doing the alpha-beta search, the computer will search 8 levels down.

**Special Explanation:**

For the computer player, when there is jump move available for it, it cannot choose a ‘slide’ move. For human player, I gave him one more option which means human player can choose a ‘slide’ move instead of selecting a ‘jump’ move. Because, defeating a computer player even in easy level is not easy for me, if I am forced to do ‘jump’ move. So I allowed human to cheat, but still I cannot win the computer in the hard mode.