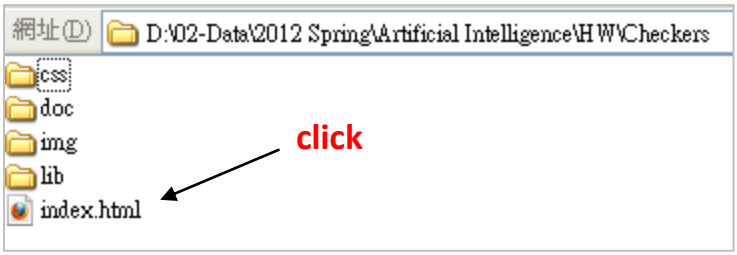
Final Project of Artificial Intelligence

Checkers Game

Student: Ching-Che Chang **Poly ID: 0435353**

1. **Instructions on how to compile and run the Checkers Game program.**

   This program is built by JAVASCRIPT with jQuery library, so it can be executed on all kinds of browsers without compiling. Therefore, what you need to do is open the Checkers folder, and then click the file "index.html". See the figure as below.



2. **A high level description of design for this Checkers Game.**

   To build this game, I use a matrix to store the tiles of both players. The matrix looks like the figure as below. All procedures of this game are based on the value change in the matrix.

1	-1	1	-1	1	-1	1	-1
-1	1	-1	1	-1	1	-1	1
1	-1	1	-1	1	-1	1	-1
-1	0	-1	0	-1	0	-1	0
0	-1	0	-1	0	-1	0	-1
-1	2	-1	2	-1	2	-1	2
2	-1	2	-1	2	-1	2	-1
-1	2	-1	2	-1	2	-1	2

1: tiles of Max-playerer

2: tiles of Min-player

0: empty square

-1: forbidden square

In addition, I have labeled every square with an index number ranging from 0 to 64, so that these indexes can help me analyze all possible moves in the Checkers Game, which means all moves can be represented by the changes in these indexes regardless of which player's tiles. See the following figure.

0	1	**2**	3	**4**	5	**6**	7
8	**9**	10	**11**	12	**13**	14	**15**
16	17	**18**	19	**20**	21	**22**	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	**41**	42	**43**	44	**45**	46	**47**
48	49	**50**	51	**52**	53	**54**	55
56	**57**	58	**59**	60	**61**	62	**63**

Here, I will show two cases to describe how to analyze the possible moves for both players.

The first one is that Max-player moves one of his own tiles from top-side to bottom-side. Therefore, the index of the tile always increases by 7 or 9 in a regular move while increasing by 14 or 18 in a jump move.

The second one is that Min-player moves one of his own tiles from bottom-side to top-side. Obviously, the index of the tile always decreases by -7 or -9 in a regular move while increasing by -14 or -18 in a jump move. So far, I find out the difference between the moves of two players is the change in the sign of indexes.

Finally, I use these rules to implement the moves of tiles for two players. Moreover, Alpha-Beta search algorithm is in charge of finding a best move for one of two players.

3. Explanations on the choice of utility values for terminal states.

For this game, I have made zero meaning draw, one Max-player wins, and minus one Min-player wins, since the choice can result in more pruning when Alpha-Beta search algorithm is running.

4. Explanations on the design of evaluation function

To design the evaluation function for this game, I have categorized all possible moves into seven groups, including perfect jump, safe jump, dangerous jump, perfect move, safe move, dangerous move, and no moves; each has different meaning as follows:

Move group	Description	The impact on Score
Perfect jump	You can take several opponent's tiles and score two, which means your tile will land on the either end of the checker board after jumping.	2 + (n_tiles – 1)
Safe jump	You can take several opponent's tiles and your tile won't be taken after your jumping.	1 + (n_tiles – 1)
Dangerous jump	After jumping, your tile will be taken.	0 + (n_tiles – 1)
Perfect move	Your tile will move into the either end of the checker board, and you score two.	2
Safe move	Your tile won't be taken after you move it.	0
Dangerous move	Your tile will be taken after you move it.	-1
No moves	No legal moves can be taken by players	0

* n_tiles: it means the number about how many tiles you take after you move one of your own tiles.
* (n_tiles -1): it means the score you got before you take the last one jump.

Based on the above definition of the move groups, I can use them to predict whether one move of either player will make himself undefeated or not. The idea is separately computing the number of tiles alive on the board for two players and then choosing one best move available for the player in the next turn to see if it will make the number of the opposite player's tiles decrease or increase. At last, I will combine the two numbers of tiles for two player by the number of the player' tiles minus the number of the opposite player’s tiles as a value of evaluation for a non-terminal node. Now, I will show the idea in the form of mathematical model for two cases when a cutoff happens and it is a non-terminal node.

If it is Max-player's turn, I will consider as below:
Eval = Num_max_tiles – (Num_min_tiles - Max_M_factor)
, where Max_M_factor is decided by move group.

If it is Min-player's turn, I will consider as below:
Eval = Num_min_tiles – (Num_max_tiles - Min_M_factor)
, where Min_M_factor is decided by move group.

To make program behave smarter, the evaluation function I proposed is considered with a move that either player take and the number of tiles alive for either side rather than only with the number of tiles for two players. Since one player's move can lead to the changes in the number of opponent's tiles, considering this point will help the program smartly choose the next move against its opponent.