Report for 2048 game

1. Abstract:

In this report , our approach for building the Artificial Intelligence Solver of Game 2048 is briefly discussed. We have used 'Minimax Search', 'Alpha Beta pruning' and 'heuristic functions' approach for implementation of the game. This report first describes '2048 game'. Then it explains our approach to solve it using above mentioned algorithms & techniques. It also gives brief information about classes in the implementation. Besides we have tested average time for 'findBestMove ()' to tuning or changing different parameters as described in reading table.

2. The game 2048 -works as following:

Initial State:
It has 16 tiles (4x4) and almost all of them are empty. Two of the tiles have either a 2 or a 4 in their places.
Iteration:
With each round, a 2 or a 4 is generated by the computer in one of the empty tiles.
Rules:
We can slide the board towards left, right, up or down and all the values on the board accumulate towards that side. If two tiles of same value are adjacent while sliding, then they sum up and the tile closest to the slided end holds the sum while the other tile disappears and all the tiles behind that tile queue up behind the first tile. The tiles closest to sliding end are added first.
Score calculation:
Whenever two tiles combine, then the sum or the tiles add up to the current score. Initially score is zero.
Goal State:
win: Form a tile that holds the value of “2048” i.e., the score generated in that round should be 2048 and you win the game.
Loss: If you have no legal moves left to slide the board.

3. Using AI to play & solve 2048:

High Level description of implementation:
ConsoleGame Class: The ConsoleGame is the main class which allows us to play the game and test the accuracy of the AI Solver.

AIsolver Class: The AIsolver is the primary class of the Artificial Intelligence module which is responsible for evaluating the next best move given a particular Board.

Utilities:

Board: The board class contains the main code of the game, which is responsible for moving the pieces, calculating the score, validating if the game is terminated etc.

Action and Direction Enum : The Action and the Direction are 2 essential enums which store the outcome of a move and its direction accordingly.

Heuristic Function:
h = curr_score + (empty_cells * log(curr_score)) – clustering_score;

where:
curr_score* = The current score
empty_cells* = Number of empty cells
clustering score* = How far away tiles of high values are present.

Significance of contributing factor & the logic of the heuristic function is as follows:

1. curr_score : is linearly added since Boards with higher scores are generally preferred in comparison to boards with lower scores to attain winning state for maximizing player.
2. (empty_cells * log(curr_score)) : The lower the score, the less important it is to have many empty cells (This is because at the beginning of the game combining the cells is fairly easy). The higher the score, the more important it is to ensure that we have empty cells in our game (This is because at the end of the game it is more probable to lose due to the lack of empty cells.
3.clustering_score: This score is subtracted from the previous two scores and acts like a penalty that ensures that clustered boards will be preferred. For example if we scatter the large values in remote cells, it would be really difficult for us to combine them. Additionally if we have no empty cells available, we risk losing the game at any minute.

Minimax, Alpha-Beta Pruning :

Despite the fact that it is only the first player who tries to maximize his/her score, the choices of the computer can block the progress and stop the user from completing the game. By modeling the behavior of the computer as an orthological non-random player we ensure that our choice will be a solid one independently from what the computer plays.

For each turn of Computer, it chooses a location from empty cells and inserts 2 or 4 at that cell intentionally, to make it harder for user to maximize his score.

We played game with different values of search depths using Minimax & Alpha beta pruning algorithms to find best moves separately.We found some observations cum verification as mentioned below.

(PTO)

4. Observations:-

Time analysis for findBestMove() for both Minimax & Alphabeta:

Depth of Search	1- AlphaBeta 0- Minimax (Algorithm)	N – Number of runs of findBestMove to compute avg. bestMove time	T – Time is milliseconds			
			1	2	3	4
7	1	50	80.4	73.08	147.21	75.62
		100	68.34	78.89	60.67	79
	0	100	80.45	79.32	71.34	83.23
4	1	50	89.94	80.44	81.12	85.47
		100	45.35	55.59	68.46	68.02
	0	100	61.31	74.4	45.39	67.18

From above table we can clearly observe following:

1. Average time to find next best move decreases, as we decrease depth of search.

2. Average time to find next best move for Minimax is greater than that of Alpha-Beta pruning algorithm, as expected.

5. References:

1. Stackoverflow: http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048

2. Stackexchage: http://codereview.stackexchange.com/questions/92488/simple-2048-ai-in-python

3. Rodgers, Philip, and John Levine. "An investigation into 2048 AI strategies." Computational Intelligence and Games (CIG), 2014 IEEE Conference on. IEEE, 2014.

Name of Students:

Ganesh (B14CS017)

G. Shravan(B14CS018)