2048 solver

Michelle Fried, Yaacov-Yonatan Goldberger, Zuzana Gavorova

The Hebrew University of Jerusalem
Jerusalem, Israel

***Abstract*—We present an artificial intelligence to the game 2048. In our solution we used Expectimax and AlphaBeta agents with various heuristics. For the heuristics we used score evaluation, the value of the highest tile, the number of empty spaces, the closeness of the values of the tiles, a heuristic assigning higher values to tiles in one corner and a heuristic assigning ascending values in a folded-snake shape. Comparing the agents and the heuristics results in the Expectimax with snake-shape heuristics as the most successful single-heuristic approach. However, some combinations of the remaining, less successful heuristics result in better performance than each of those heuristics individually. A combination of the corner and the score evaluation heuristic is even better than the snake heuristic.**

I. INTRODUCTION

In our project we decided to solve 2048 by using artificial intelligence. A year ago, like everyone else, we managed to get addicted to the 2048 game. This addiction made us choose that game as our final project and to try different approaches and methods for solving it as we describe next.

II. GAME DESCRIPTION

The goal in this project was to solve 2048 game in different ways. 2048 is a single-player game, in which the objective is to slide numbered tiles on a grid to combine them and create a tile with the number 2048. "2048 is played on a gray 4×4 grid, with numbered tiles that slide smoothly when a player moves them using the four arrow keys. Every turn, a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move… A scoreboard keeps track of the user's score. The user's score starts at zero, and is incremented whenever two tiles combine, by the value of the new tile."[1]

III. APPROACH AND METHODS

In order to solve the game we used the following methods:

A. AlphaBeta pruning:

Alpha-Beta pruning[2] is just an optimized version of MiniMax algorithm which solves a game with two players. The algorithm finds the best move for our player assuming that the opponent plays his best move. In a zero-sum game the best move of our player is the move with the highest heuristic value, while the best move of the opponent is the one with the lowest heuristic value (hence the name MiniMax).

In order to use Alpha-Beta pruning we must first identify the two players. The first player is our agent who plays the game. The second player is the 'computer agent' which randomly inserts valued tiles in the cells of the board. Our agent tries to maximize the heuristic value. On the other hand the 'computer agent' in the original game is not specifically programmed to block our agent but rather randomly inserts values on the empty cells. So the question is why we use an AI technique which solves zero-sum games and which specifically assumes that both players select the best possible move for them? The answer is simple, despite the fact that it is only our agent who always plays his optimal move, the choices of the 'computer agent' can block the progress and stop our agent from completing the game. By modeling the behavior of the 'computer agent' as an optimal player we ensure that our choice will be a solid one independently from what the 'computer agent' plays.

Zero-sum game approach gives rise to a game tree. Depending on how far into the future we want to look when deciding the current move, the tree is expanded to several levels, where one level is one move of our agent followed by one move of the 'computer agent'. The branching factor at the move of our agent is 4, corresponding to sliding in the maximum of 4 possible directions. (If the current game state is such that sliding in some direction would not alter it, this direction is not available.) The 'computer agent' during his move inserts a tile of value 2 or 4 into any empty place so the branching factor at his move is twice the number of empty cells. The overall branching factor of the tree is thus of the

order of $8l^2$ where l^2 is the size of the game board. This large branching factor justifies the use of Alpha-Beta pruning, which avoids expanding useless nodes.

B. Expectimax search:

The problem with the AlphaBeta agent is that it is unnecessarily careful when choosing the next move. We know that the 'computer agent' is not optimal, but random. We would like our agent to make the best move while being subjected to a random event (the random tiles being added). Knowing the probabilities of the 'computer agent's' moves we can use Expectimax algorithm[2] to model the opponent's behavior more realistically. What this algorithm does is to recursively predict the future possible moves and to choose the next immediate one which will probably lead to the best outcome.
In the Expectimax algorithm we expand the full game tree as described for MiniMax, but based on this tree we choose the next move differently. Moving from the leafs upwards, we take heuristic-value expectations over the possible 'computer agent' moves and maxima over our agent's moves. The expectations over the 'computer agent's moves are with respect to the new-tile probabilities taken from the source code of the original 2048 game. Since expectations are taken over all possible moves, there are no useless nodes that can remain unexpanded and the branching factor cannot be reduced.

C. Heuristics:

C.1. ***Score heuristic***

This heuristic returns the current score. The goal is to reach higher score so we would like to maximize the value returned by this heuristic. To fit the range of the values of the other heuristics the returned value is actually the score divided by *4*.

C.2. ***Empty space heuristic***

This heuristic calculates the number of empty tiles (Figure 1). Its goal is to maximize the number of empty tiles on the board. However, every turn, the board generates a new tile. The intuition behind this heuristic is that the only way to move towards a board with more empty tiles is by merging blocks together. Merging blocks aligns with the main objective of the game. To fit the range of the other heuristics, the value returned is the number of empty tiles multiplied by *100*.

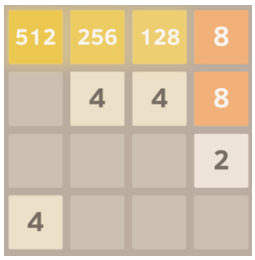

Fig.1. *Empty space heuristic returns 100 times the number of empty spaces.*

C.3. ***Differnce heuristic***

In this Heuristic we compute the difference between every two adjacent tiles and sum the results. We return the negative value of the sum because we are trying to minimize the differences.

C.4. ***Max Tile heuristic***

This heuristic returns the maximum value in the board. The goal of this heuristic is to reach higher values in the game (to reach 2048 and more). This is done by updating the max tile in each move and to use its value in the evaluation function.

C.5. ***Corner heuristic***

This heuristic of a game grid is computed via a set of weights. The weights increase either towards the top-left, top-right, bottom-right or bottom-left. We have defined a set of gradient weights for each of these directions (Figure 2 left). The highest values appear in one of the corners and descend along the diagonal. The tile value in each cell (an empty cell has value 0) is multiplied with its corresponding weight, and the product is summed up over all the tiles in the game grid to give the value in that particular weight gradient direction. We repeat this for all four directions, and take the highest value.

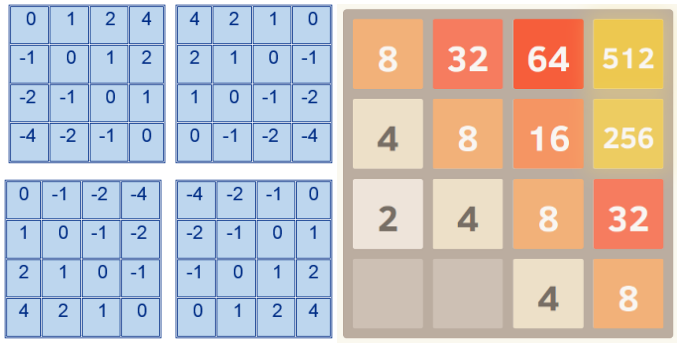

Fig.2.: *Corner heuristic returns the maximum between the four weight grids (left) multiplied with the actual tile values on the board (right).*

C.6. ***Snake Heuristic***

This heuristic of a game grid is computed by weights ascending along a snake shape. The weights can increase

along 8 different directions. We have defined a set of weights for each of these directions (Figure 3). The tile value in each cell (an empty cell has value 0) is multiplied with its corresponding weight, and the product is summed up over all the tiles in the game grid to give the 'snake' heuristic value in that particular direction. We repeat this for all eight directions, and take the highest value divided by 2^{13}.

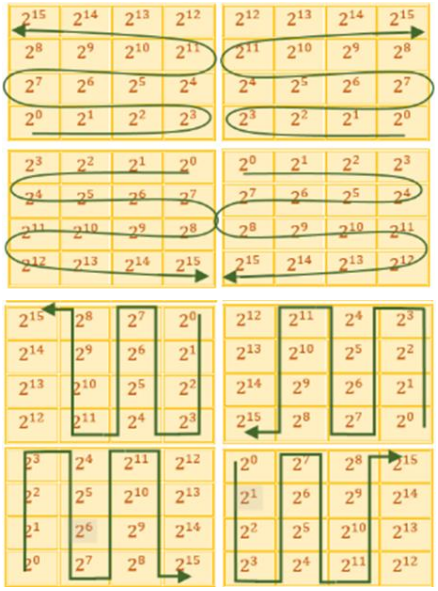

Fig. 3: *Snake heuristic returns the maximum bertween the eight grids above multiplied with the tile values of the board.*

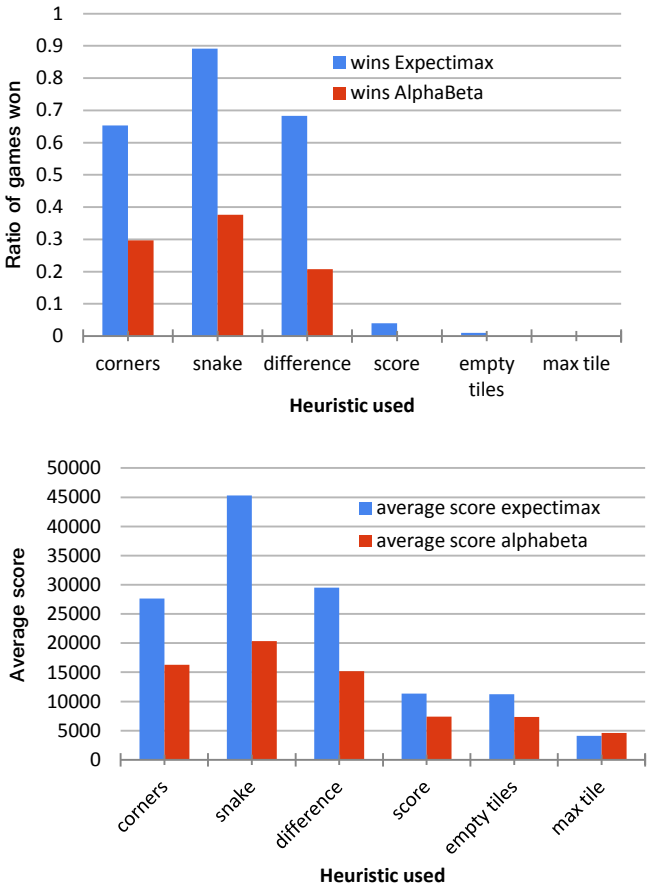

Fig. 4: *Performance of each heuristic with Expectimax and AlphaBeta agent with 2-moves lookahead.*

IV. PERFORMANCE ANALYSIS

A. Individual Heuristic:

To compare all the heuristics used by both Expectimax and AlphaBeta agents we averaged over 100 runs of the game for each heuristic-agent combination. The results are shown in Figure 4. Both agents were set to expand the game tree to depth 2. This corresponds to the lookahead of two steps when deciding the best move.

The Expectimax agent models the opponent or the 'computer agent' according to the true probabilities of inserting the next tile. The Expectimax, being realistic, performs much better than the pessimistic AlphaBeta agent, which assumes that the opponent always makes the move which is the worst for us. The AlphaBeta agent's assumption simply does not correspond to the game we are playing and this is why its performance is worse for all the heuristics.

Both agents perform the best with the snake heuristics. Depending on the agent, the corners and the difference heuristics come the 2nd and 3rd. The snake heuristic performs the best because it avoids well being blocked by the 'computer agent', it directs the behavior of the agent the most and at the same time it offers a clear long path towards collapsing with the highest tile.

To illustrate why the snake heuristic cannot have its strategy spoiled by the 'computer agent' so easily, we can compare it to the corners heuristic. The strategy of the corners heuristic is to keep the high values in one corner. To make sure that the low values do not pop up at the chosen corner and get mixed up with the high values that the agent keeps there, the agent wants to perform one of only two particular slide-moves at any later stage of the game. There is a high chance that at some point these two moves are blocked for him and he has to take a bad move which might spoil his corner-strategy. The agent using the snake heuristic keeps his highest values in e.g. the top row (Figure 5 right) and most of the time that row is filled up and there are 3 particular moves that are essentially harmless to the agent's snake strategy. Three moves are much less likely to be all blocked.

The second advantage of the snake heuristic is that it is more directing, or hierarchical, than for example the corners heuristic, because the weight assigned to each tile is different (Figure 5 left). The corners heuristic on the other hand has multiple tiles with the same weights (Figure 6 left). The less hierarchical heuristics much more frequently result in situations when the returned heuristic value is the same for more than one choice of our agent's move. The agent then chooses the first move it loops over. Better heuristics avoid such situations.

The third advantage of the snake heuristic is its directionality, i.e. it exactly specifies the path along which the values of the tiles should increase. This is the snake-shaped path. Once a state of the board is reached where every next tile on the path is a multiple of two of the previous one (Figure 5 right), the snake heuristics will choose precisely the sequence of moves which makes all those tiles gradually collapse into the highest tile in the corner. As oppose to that, the corners heuristics offers multiple equally-rated paths when starting from the corner-gradient board (Figure 6 right). Most importantly, the paths offered by the corner heuristics are shorter, than the snake path. Thus in the later stages of the game when the board gets filled up with high-value tiles, the agent using the corner heuristic gets blocked sooner.

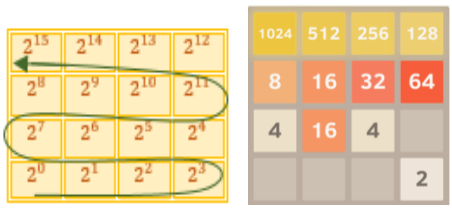

Fig.5. *board-snake*: *Optimal arrangement of tiles according to the snake heuristic.*

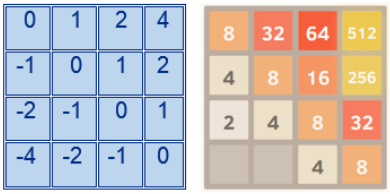

Fig.6. corner heuristic weights: *Optimal arrangement of tiles according to the corner heuristic.*

The difference heuristic also tries to get the high-valued tiles to the edges of the board rather than keeping them in the middle. This is because tiles arranged in a row in a monotonically increasing order have a lower sum of the differences between neighbours than if they are arranged so that the maximum tile is in the middle. Thus when minimising the neighbour-tile differences in both horizontal and vertical directions we also get preference towards putting the high-valued tiles into the corner. This directionality causes the difference heuristic to also perform better than the remaining three: the empty tiles, score and max tile heuristics. The three poor heuristics are greedy in collapsing the tiles fast (especially if the lookahead of the agent is small) but they are insensitive to the positioning of the highest-value tile.

While Expectimax agent outperforms the AlphaBeta agent in terms of the percentage of the wins and the score reached, it is approximately twice as slow as AlphaBeta agent. Figure 7 plots the average number of nodes expanded in a single lookahead tree. AlphaBeta agent is faster because it does not need to use all the branches of the tree and so it avoids expanding the useless ones.

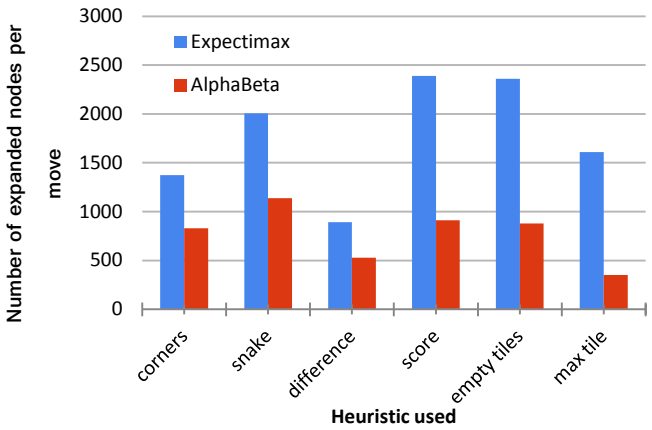

Fig.7: *Average number of expanded nodes in one lookahead tree for Expectimax and AlphaBeta agent.*

Note that the average lookeahead tree of the three worst heuristics is larger simply because the game does not stay for long in the later stages when the game board is more filled up and the branching factor of the 'computer agent' proportional to the number of empty tiles is smaller. However, because of their greediness, the score, empty tiles and the max tile heuristics stay relatively long in the empty-board stages. Figure 7 teaches us that the snake heuristic manages to maintain an empty board for a high fraction of time.

Because of the huge branching factor, increasing the game tree to three levels slows down the computation too much. In Figures 8,9 we compare the performances of the agents with 2-moves lookahead to their performances if the lookahead is increased to 3-moves only at the later stages of the game, when the number of empty tiles decreases to 6 or less. The number of expanded nodes increases exponentially with the depth of the tree. A large increase can be seen in Figure 8 for both Expectimax and AlphaBeta agents respectively. Note that the corners heuristic improves the most with a further lookahead at the later stages of the game (Figure 9). Comparing to only a small improvement in the snake heuristics, it can be argued that the lookahead is already partially

incorporated in the snake-weights increasing with every single position. However for the corners heuristic, the 3-step lookahead might resolve exacly those cases where the 2-step lookahead returned the same value for multiple options. In all of the remaining analysis 2-step lookahead was used.

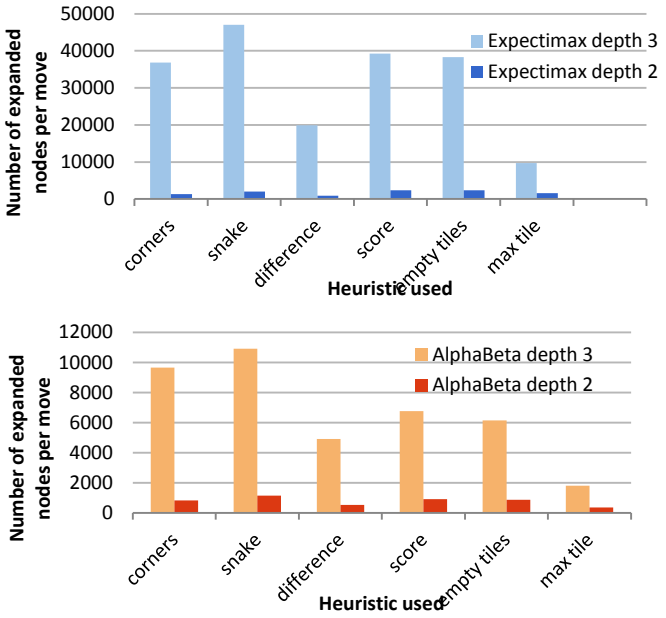

Fig.8: *Increase in the average size of the lookahead tree forExpectimax (top) and AlphaBeta (bottom) agent if a third level is added for the trees which start from a state with 6 or fewer empty spaces on the board.*

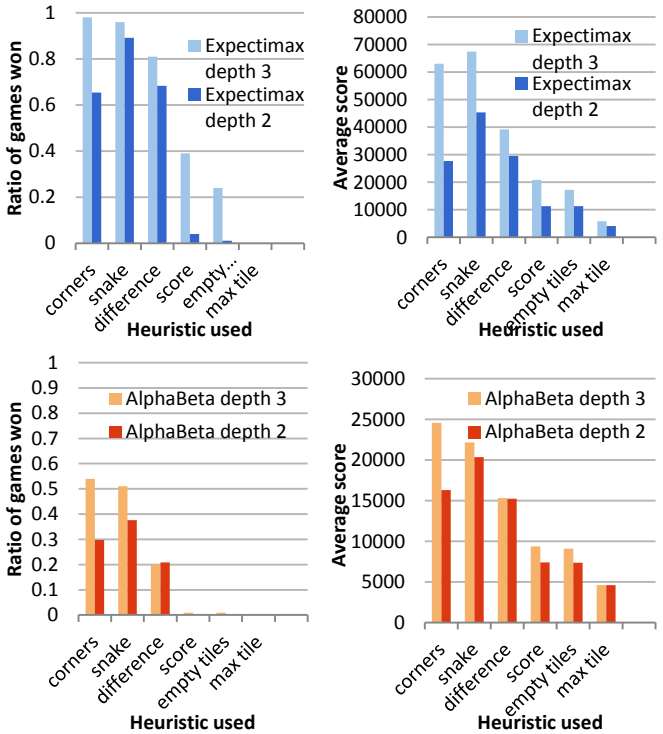

Fig.9: *Improvement of the performance of each heuristics and agent if for the situations with 6 or fewer empty spaces on the board the agent's lookahead increases from 2 to 3 moves.*

B. Combinations of heuristics

Having identified that some of our good heuristics may sometimes return an heuristic value that corresponds to more than one "best" move, we would like to make the heuristic more restrictive. This can be achieved by expanding the lookahead, which is computationally very expensive, or by combining the good heuristic with a second heuristic (Figure 10).

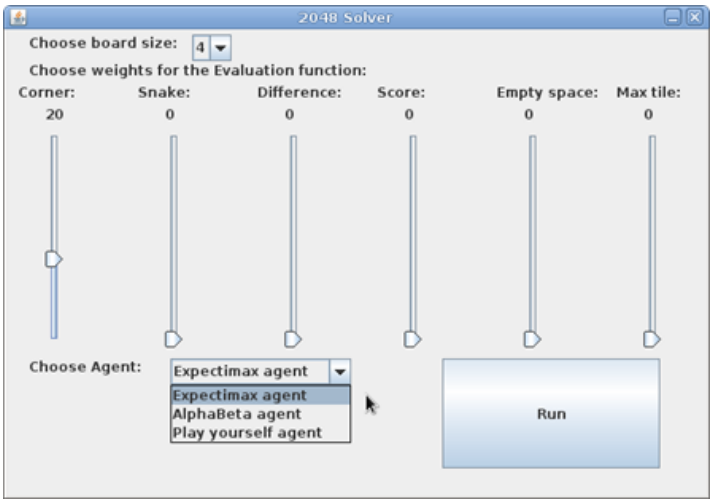

Fig.10: *When solving the game we can choose the boardsize, the agent and also the relative weight assigned to each heuristic in case we want to combine multiple heuristics.*

The role of the second heuristic is only to take care of the tie-cases. Calling the values returned by the first and the second heuristics *h1* and *h2*, the new heuristic value returned by such a combined heuristic is:

$$h = w_{h1}\, h1 + w_{h2}\, h2,$$

where w_{h1} and w_{h2} are the weights of the first and the second heuristic respectively.

Figure 11 shows the effect of adding the on its own not very successful score heuristic to each of the good heuristics. Not surprisingly, the snake heuristic combined with this bad heuristic only worsens. On the other hand, corners heuristic and difference heuristic improve, which makes us believe that an addition of the score heuristics corrects for their small drawbacks. Note that adding the score heuristic especially improves the difference heuristic. The fact that there was so much room for improvement indicates that the difference heuristics on its own does not maximize the score enough. This is similar to how we earlier concluded that the corners heuristic does not have enough lookahead incorporated in it.

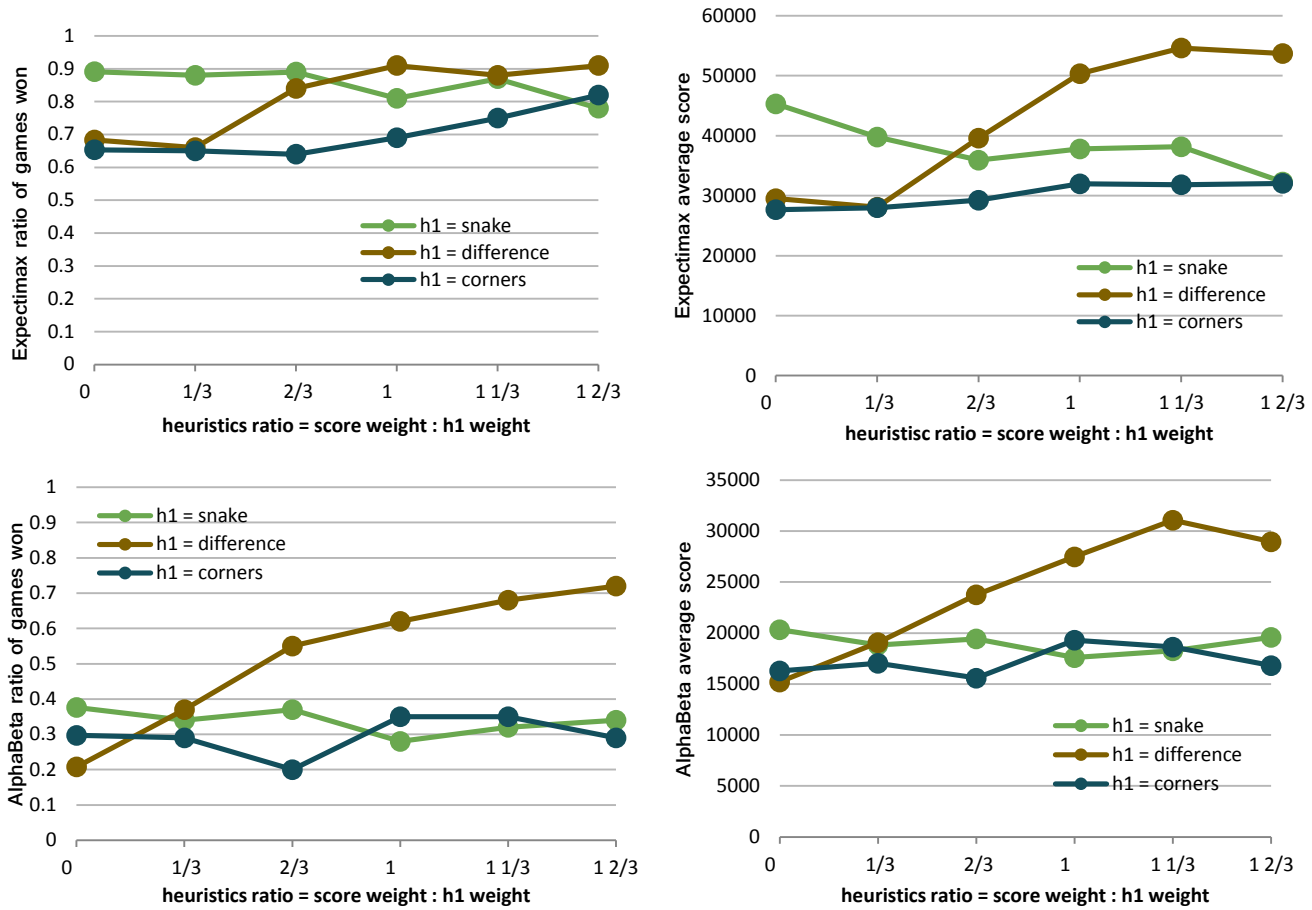

Fig.11: *The fraction of the games won (left) and the average score (right) for both Expectimax (top) and AlphaBeta (bottom) agents using each of the good heuristics combined with the score heuristics. The larger values on the horizontal axis correspond to more score heuristics being mixed in.*

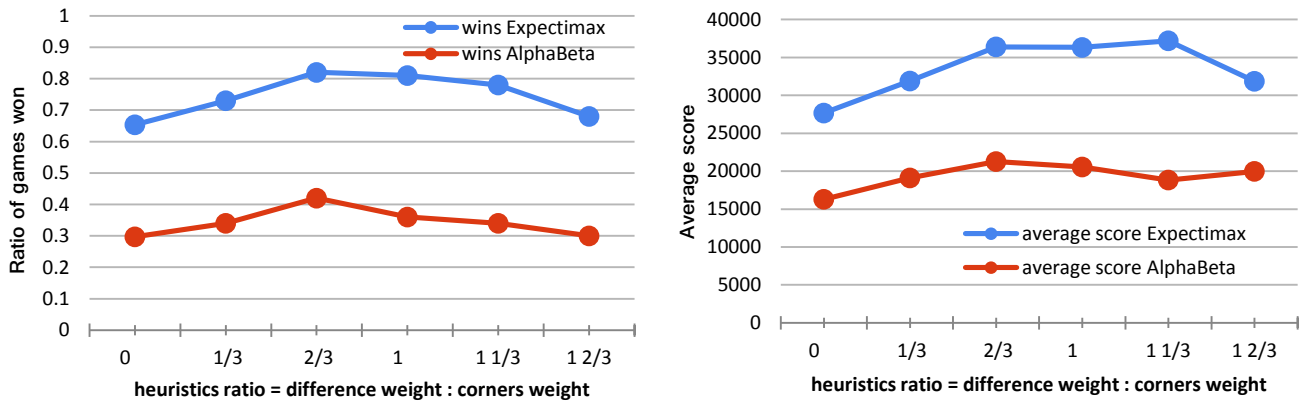

Fig.12: *The fraction of the games won (left) and the average score (right) for Expectimax or AlphaBeta agent using a combination of difference heuristic and corners heuristic. The larger values on the horizontal axis correspond to more corners heuristic being mixed in.*

Since the corners heuristic and the difference heuristic are both sensitive to the same 'corrections' but they each have a different dominant drawback, it might be interesting to combine the two. Figure 12 shows that the combination of the two indeed performs better than each of them separately.

V. CONCLUSION

We have shown that the Expectimax agent is more appropriate than the AlphaBeta agent for solving the game 2048, because the addition of the new tiles into the 2048 board is random. The best single heuristics to be used by the agent are the snake heuristics achieving the

probability 0.9 of winning and the average score (over 100 games) of 45000. However, the same winning probability and the average score of 55000 can be achieved by a combination of corners and score heuristics.

REFERENCES

[1] 2048 (video game) - Wikipedia, the free encyclopedia. Retrieved from http://en.wikipedia.org/wiki/2048_(video_game) on 26.03.2015.

[2] J. Rosenschein: Introduction to Artificial Intelligence - Course 67842 course notes, Fall Semester 2014/15, Hebrew University of Jerusalem.