INFO-H410
TECHNIQUES OF ARTIFICIAL INTELLIGENCE

PROJECT REPORT

EXPECTIMAX AND MONTE CARLO TREE SEARCH

Students
Sébastien DE VOS
Tarik KALAI
David SILBERWASSER

Professor
Bersini HUGUES

Assistants
Bono Rosselo LLUC

2021-2022

Contents

1 Introduction **2**

2 Expectimax **2**
2.1 Explanation of the choice 2
2.2 Heuristic 3
2.3 Results 3

3 *Monte Carlo Tree search* **5**
3.1 Choice Explanation 5
3.2 Results 6

4 *Expectimax* VS *Monte Carlo tree search* **8**

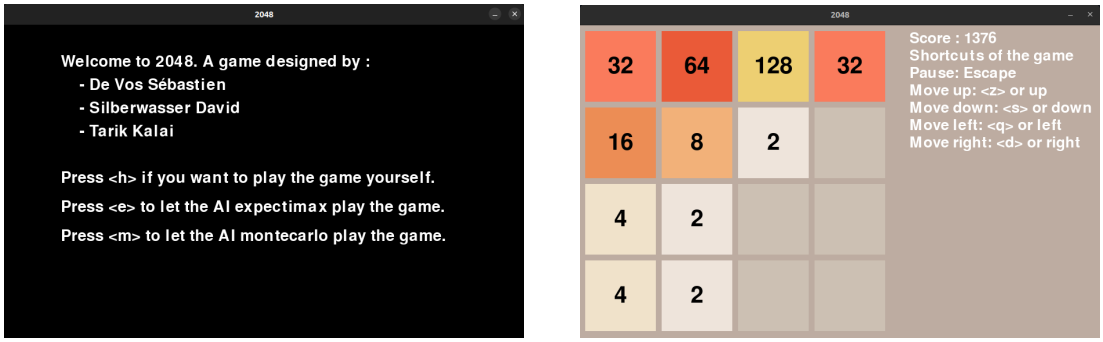
5 Conclusion **9**

List of Figures

1 Game Design 2
2 *Expectimax* performance 3
3 *Expectimax* cumulated performance on 100 trials 4
4 Box-Plot Expectimax 4
5 Monte-Carlo Tree Search [1] 5
6 Monte Carlo frequency histogram performance 6
7 Monte Carlo Cumulated Score 7
8 Monte Carlo Box-plot 7
9 Cumulated scores for both AI algorithms 8
10 Performance Comparison between the best of both AI algorithms 9

1 Introduction

This report will focus on the implementation of an AI for the game 2048. The game itself takes place on a 4 x 4 board, the player can move tiles using the arrow keys (left, right, up, down) if 2 tiles of the same value collide they will be merged and a new tile with twice the value of the previous tiles appears. After each turn a tile appears randomly on the board, it can have the value 2 (9 out of 10 times) or 4 (1 out of ten times).[1].
Two kinds of artificial intelligence will be implemented: *Expectimax* and *Monte Carlo Tree Search*. Details about the performance of the two techniques will be provided and in order to select the best kind of AI to use in this particular environment a benchmark will be implemented.



(a) Menu Interface

(b) Game Interface

Figure 1: Game Design

2 Expectimax

2.1 Explanation of the choice

This algorithm is a variant of the *MinMax* algorithm. The *MinMax* couldn't be implemented directly for two main reasons:

- The concept behind *MinMax* suggests to use it when the game confronts two players together, in this case it is a single-player game.
- There is a random component in the 2048 game (random appearances for the tiles at each turn) which makes the implementation of *MinMax* inefficient because making all the combination of moves with random apparitions would make the tree explode quickly.

For these reasons *Expectimax* was implemented instead. It works in the same way as *MinMax*, but instead of alternating between minimum and maximum, it will alternate between maximum and average.

It is worth to observe that Expectimax does not offer the possibility to use pruning[2] like MinMax as the value of a single unexplored instance can change the value of the following moves. As a

[1]For additional information: 2048-Game
[2]practice used to accelerate the algorithm by removing the useless or redundant solutions

consequence, it requires the full search tree to be explored and it can be slow. This can intuitively limit the computational capacities for the algorithm.

2.2 Heuristic

A single heuristic was used for this algorithm: it concerns the positions on the board. Indeed, a well-known trick to beat the *2048* game is to position the biggest tiles in the corners. As such, a weight matrix was used to make the AI understands where is the most important position. Here is the matrix which is currently being used:

$$\begin{bmatrix} 0.135759 & 0.121925 & 0.102812 & 0.099937 \\ 0.0997992 & 0.0888405 & 0.076711 & 0.0724143 \\ 0.060654 & 0.0562579 & 0.037116 & 0.0161889 \\ 0.0125498 & 0.00992495 & 0.00575871 & 0.00335193 \end{bmatrix}$$

This weight matrix was computed in order to obtain the best adapted weighted matrix[3]

2.3 Results

To asses the performance of the AI, 100 runs have been made with depth[4] ranging from 2 to 6 and the final score of each game has been kept. The results can be seen in the following figures[5].

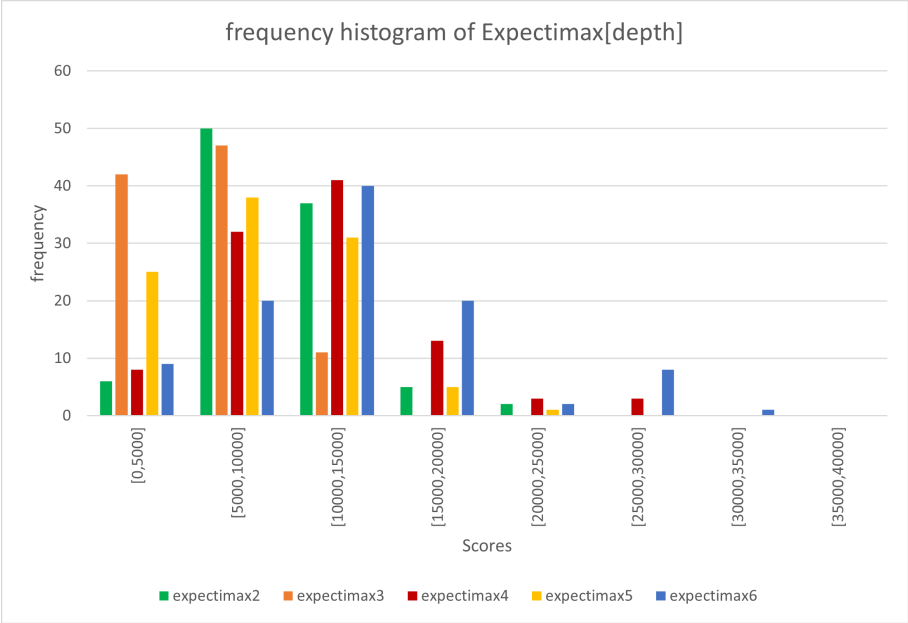


Figure 2: *Expectimax* performance

[3]Source: Weight matrix computation

[4]Depth corresponds to the level of prediction of the algorithm: if depth is 2 it means that the prediction of the Expectimax will stop after computing the possibilities of the random apparitions of the tiles once.

[5]The tile 2048 is obtained when a score higher than 20000 is achieved

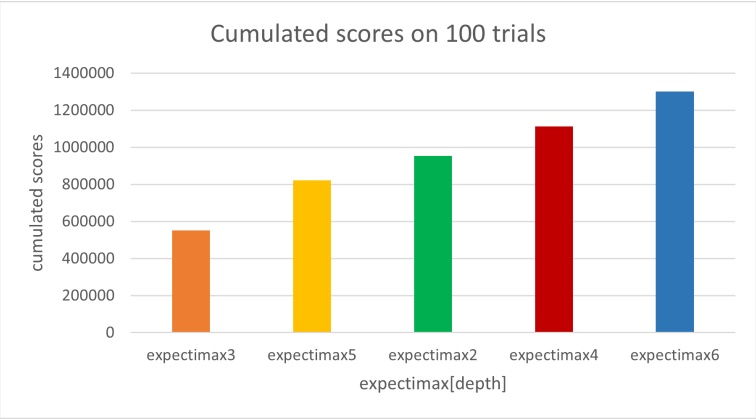


Figure 3: *Expectimax* cumulated performance on 100 trials

It can be noticed on these graphs (2, 3) that an even value for the depth will typically perform better than its direct superior odd depth. This can be explained by the fact that when the depth is odd the last step of the algorithm will not compute the average results, thus omitting the randomness of the problem. Whereas when the depth is even the algorithm will compute the mean of the last component, this will mitigate the randomness impact and greatly improve the performance of the AI. This effect is even more marked on the figure 4 where depth 2 performed better than depth 5 but worse than depth 4.

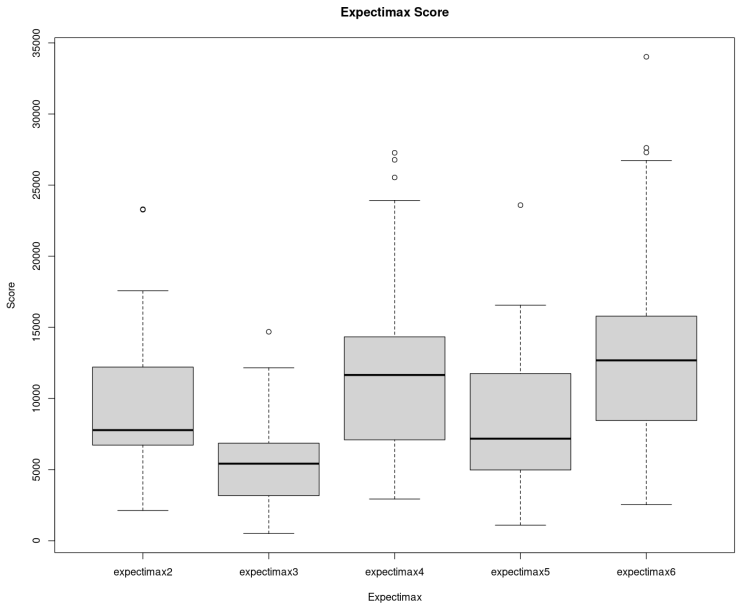


Figure 4: Box-Plot Expectimax

The best performing depth corresponds to the highest even number which is understandable as the AI computes its own move while taking into account the randomness of the game. By this logic a

depth of 8 will be better than one of six[6].

3 *Monte Carlo Tree search*

3.1 Choice Explanation

The Monte Carlo tree search technique was chosen as it also contains, randomness in its computation. This will make the comparison with Expectimax interesting as one will have randomness but with some guidelines (i.e. the weight matrix), whereas the Monte Carlo tree search will play without a priori knowledge of the game (it will create its own way of playing the 2048 game).

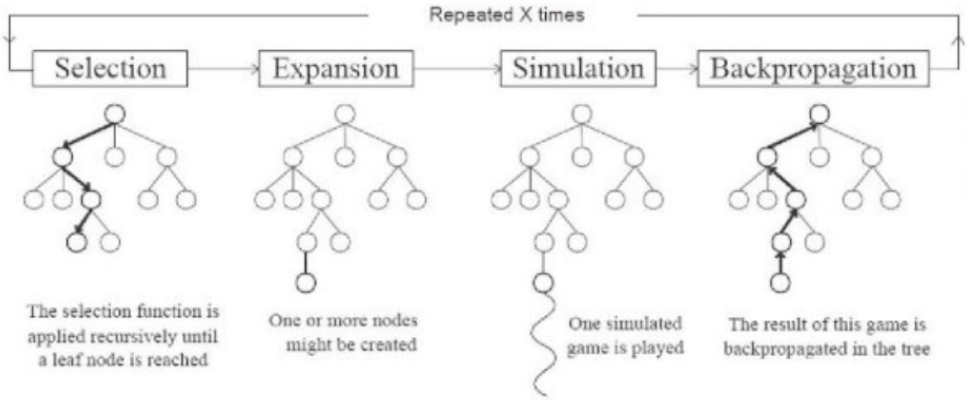


Figure 5: Monte-Carlo Tree Search [1]

The Monte Carlo tree search works in 4 phases:

1. It will select the node to expand by computing the UCB[7] formula (inspired by [2]) :

$$UCB = \frac{s}{n} + c * \sqrt{\frac{\ln N}{n}} \tag{1}$$

   where:

   - s = score of the child node
   - n = number of visit of the child node
   - N = total current number of visit
   - c = a constant to make a trade-off between exploitation and exploration. It has been decided to put it to 2.

2. Then it will expand this node.

   - If this node has not been visited it will make a rollout/simulation.

[6]This could not be demonstrated here due to limited computing power
[7]Upper Confidence Bound built to maximize our score

- If it already has been visited it will create d childs, where d is the number of possible move for a given grid. Then it will select the first child and make a rollout.

3. The rollout/simulation consists in playing randomly until the game is done.
4. Finally, it back-propagates the game's score after the rollout. This score will be the nodes score.

This process can be repeated multiple times. Once the number of iterations allowed has been exceeded, the move that will be selected is the move corresponding to the root node's child with the highest UCB (see 1) value.

3.2 Results

Like Expectimax, to asses the performance of the AI, 100 runs have been made with iterations ranging from 10 to 50 and the final score of each game has been kept. The results can be seen in in the following figures.

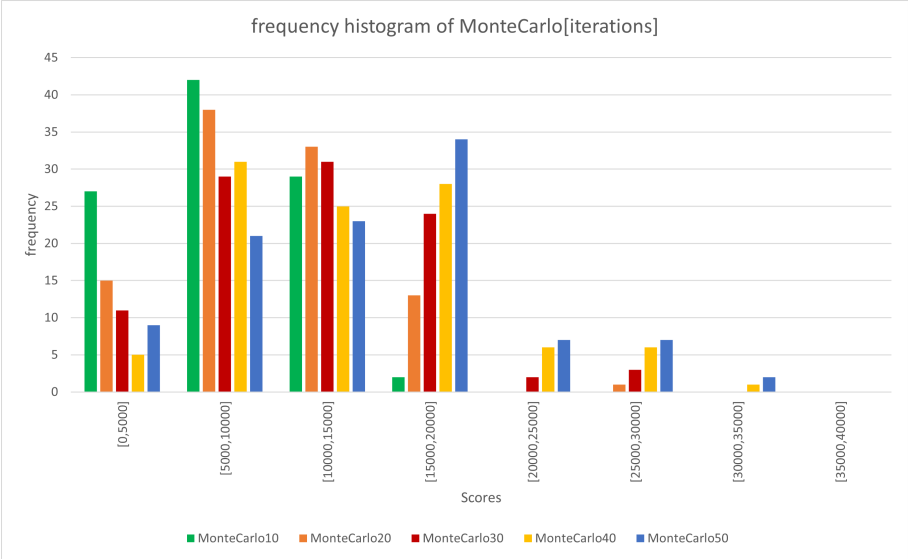


Figure 6: Monte Carlo frequency histogram performance

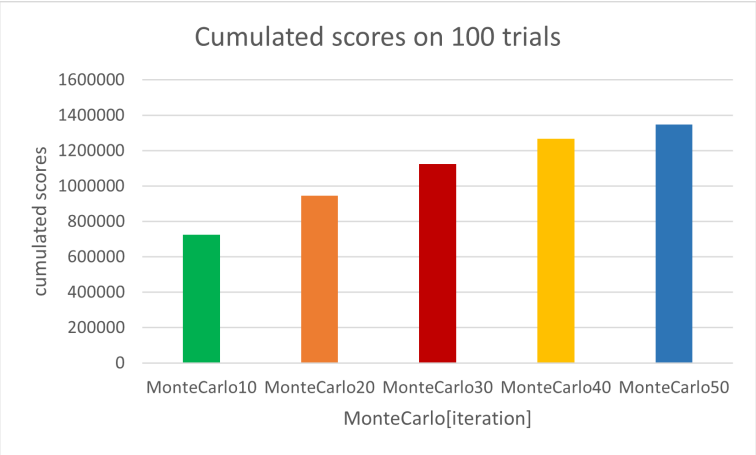


Figure 7: Monte Carlo Cumulated Score

As indicated by the graphs (see 6, 7), the results improve when considering a higher number of iterations, which was expected. What cannot be seen on these graphs is the way it played the game. Instead of heading for the corners, it decided that keeping higher numbers in the center of the board was better.

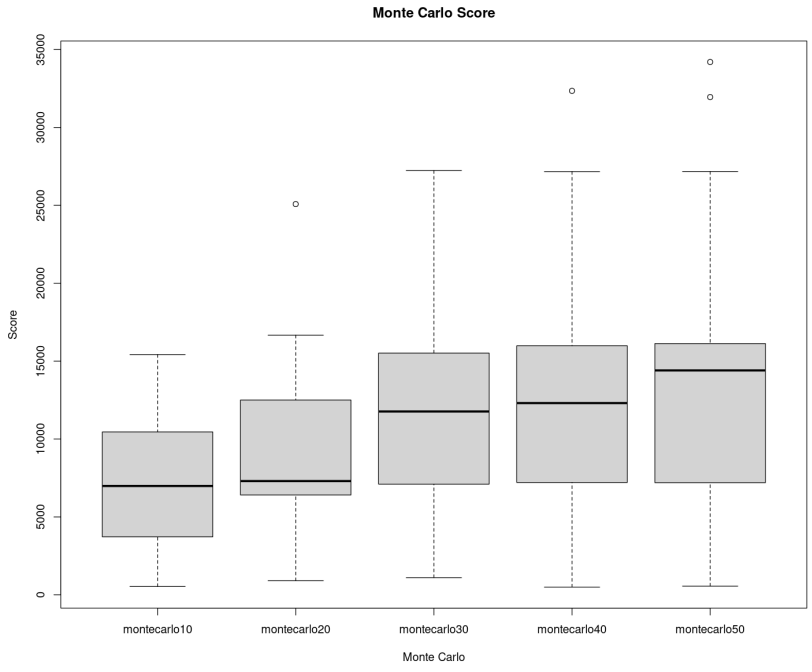


Figure 8: Monte Carlo Box-plot

Based on the graph 8, it is seen that the mean of the scores increases with the number of itera-

tions. Whereas the overall performance doesn't improve much, especially in the last 3 families of iterations. Indeed, the variance is almost null for these specific scores.

4 *Expectimax* VS *Monte Carlo tree search*

The following graph represents the respective cumulated runs for Expectimax and Monte-Carlo. It can be seen that Monte-Carlo tends to perform better in general.

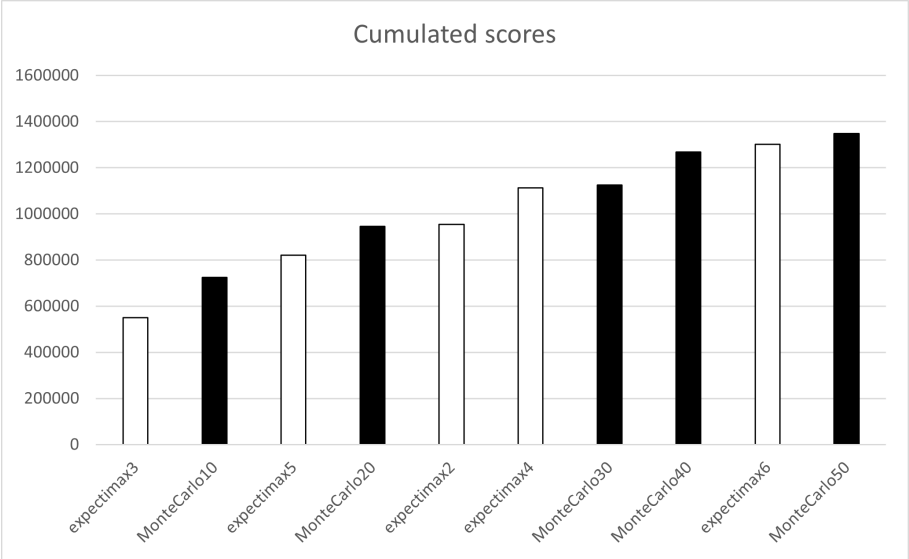


Figure 9: Cumulated scores for both AI algorithms

Moreover, Expectimax grows exponentially with the depth whereas Monte Carlo grows linearly with the number of iterations. When conducting these experiments, it was made so that the running time of the longest experiment of Monte Carlo would take a similar time to that of Expectimax. Thus, Monte Carlo was constrained to the running time needed by Expectimax. Nevertheless, Monte Carlo wielded better results than Expectimax. As such it can be anticipated that it will also beat it for longer running time.
On figure 10, it can be seen that the Monte Carlo curve is more on the right side (in higher values of scores), thus leading to better played game. Whereas Expectimax has its peak performance on the lower scores.

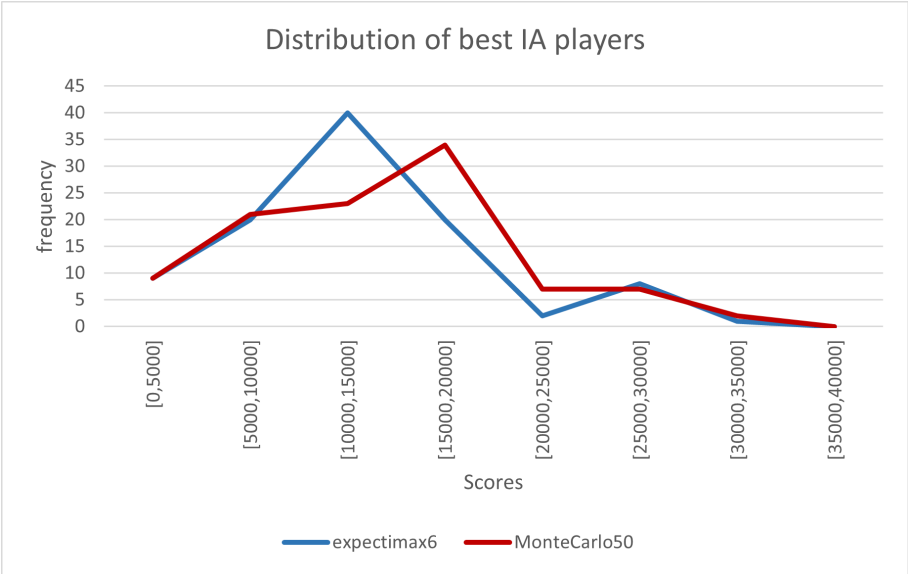


Figure 10: Performance Comparison between the best of both AI algorithms

5 Conclusion

To conclude, it can be said that Monte-Carlo performs generally better than Expectimax. Nevertheless, Expectimax still offers faster computation time for smaller depths and is still capable of reaching decent scores. It is still worth to take into account that for this particular algorithm, the odd values for the depth greatly decreases the performance. It was supposed that it is due to the random component inherent to the 2048 game. Finally, one of the possible reasons that can be considered as a valid reason for the fact that Expectimax did not give as good results as Monte Carlo is the fact that Expectimax is guided by human knowledge (use of heuristic) whereas Monte Carlo considers its possible moves by itself (only uses the score of the game).

References

[1] Mark Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. "Monte Carlo Tree Search in Lines of Action". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2 (Dec. 2010), pp. 239 –250. DOI: 10.1109/TCIAIG.2010.2061050.

[2] John Levine. *Monte Carlo Tree Search: class "CS310: Foundations of Artificial Intelligence" at the University of Strathclyde.* en-EN. URL: https://www.youtube.com/watch?v=UXW2yZndl7U. (accessed : 11.04.2022).