Ai for Autonomous Systems – CS814 Assignment.

Monte Carlo Tree Search & Mean Average Simulation-based Search applied to the popular web game '2048'.

By: Frank Mitchell
Course: MSc Artificial Intelligence and Applications.

Contents

Abstract

The following research paper implements a traditional Monte Carlo Tree Search (MCTS) and a Mean Average Simulation-based (MASb) artificial intelligence agent to the popular internet game ‘2048’. The aim is to maximise the agent’s individual tile values, as well as the total score at the end of the game.

Findings conclude that a vanilla implementation of MCTS does not perform well. The algorithm cannot seem to accurately predict ‘high-scoring’ moves by using an Upper Confidence Bound to measure potentially high-value paths. This is presumably due to the random nature with which the system drops tiles on the board.

However, the scoring was increased by using a ‘Mean Average Simulation-based’ algorithm, which performs game simulations from each subsequent move after the start state. So, for every move from the current state, the system will simulate games and use the Mean Average Value of all the simulations to assign a value to that particular child node.

Due to simulating the game multiple times from the same leaf-node, all moves (including the random dropping of tiles) are explored and a more accurate estimation of that moves’ particular end-score can be achieved.

Throughout both algorithms it was necessary to implement a depth-limit while rolling out simulations. This cut down on computational overhead, allowing the agent to make decisions within a reasonable timescale.

It was also found that a combination of both techniques produced very good results, with high individual tile values and mean average total scores.

1. Introduction

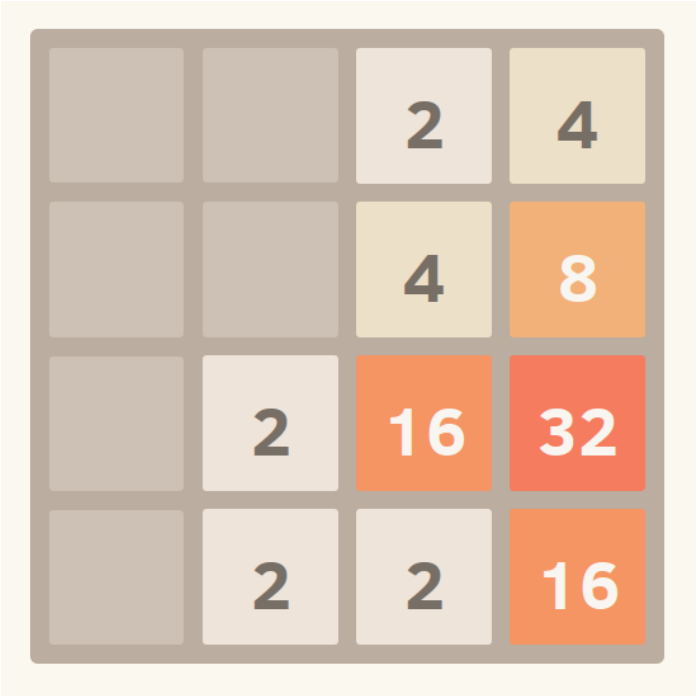

Figure 1 – Still of 2048 game

Description of 2048

2048 is a game created by Gabrielle Cirilli, an Italian web developer who was inspired by the 'game of three's' and created 2048 over one weekend in 2014 (Wikipedia, n.d.). The rules are relatively simple, the user is presented with a 4 * 4 grid-board, on which is initially dropped two random tiles (either a 2 or a 4 with an 80/20 chance). Throughout the game the user must decide whether to slide the tiles up, down, left or right, with transformations happening as outlined by the following rules:

i. If a tile collides with another tile of the same value, those tiles will be added together and this value is added to the total score;

ii. If three tiles collide that have the same value, the two tiles farthest from the direction of motion will be added together and this value is added to the total score;

iii. If four tiles that have the same value appear in the same column or row, if slid vertically or horizontally respectively, the tiles in positions 2 and 4 (0 & 2 in computer science parlance) will be added together and the remaining tiles will become blank. The total value of all tiles added will be added to the total score.

Once the user has made her move the system will randomly drop another 2 or 4 onto the board. The aim of the game is primarily to achieve a tile with the value “2048” or higher, and secondarily to score as many points as possible.

Description of Monte Carlo Tree Search

In 2006, Monte Carlo Tree Search (MCTS) was created and applied to the game of Go (Gelly, et al., 2006). Since then the algorithm has come to dominate the game (Bradberry, 2015), famously used as part of an ensemble method to beat champion Go player Lee Sedol in March 2016.

It is a probabilistic algorithm which automatically explores potentially high-scoring areas of the game-tree by assigning an Upper Confidence Bound value to each node. MCTS is a statistical anytime algorithm that can be used with little or no domain knowledge, and has succeeded on difficult problems where other techniques have failed (Browne, et al., 2012).

Description of Mean Average Simulation-based Search

While experimenting with varying parameters in MCTS, it was discovered that higher scores and individual tiles can be achieved by running simulations from only the children of the current state. I’m calling this technique Mean Average Simulation-based Search (MASb).

In this implementation no ‘tree’ is ever built, and simply the mean average of all simulations is used to assign a value to the child node. The system will then choose the move with the highest mean average over multiple simulations.

Experiments using a combination of both techniques (where a small portion of the game tree is built while still performing multiple rollouts on leaf nodes) also yielded successful results.

2. Implementation

Traditional Monte Carlo Tree Search

Outlined below are the steps taken by the agent to search for the next best move using traditional MCTS:

i. A random tile is dropped on the board by the system;
ii. The agent creates the root node of the tree with the current state of the board (stored as a dictionary where the parent is the key and the children are the values stored in that key position);
iii. Next, the agent will perform a pre-determined number of rollouts (a constant variable that can be tuned by the user);
iv. During these rollouts the agent will explore the tree until a leaf node is found. If the current state is not a leaf node the agent will calculate the Upper Confidence Bound of each of its children and choose the highest value UCB to further explore the tree.
v. When a leaf node is discovered, the agent next evaluates how many times this leaf node has been sampled. If the node has never been sampled the agent performs a rollout (random moves until the end of a game) and back-propagates the final score from the leaf node, to each parent, back up the tree.

vi. If the node has been sampled more than once the agent will add this node (as a key) and its children (as that key's values) to the tree and choose the first child to perform a rollout, then back-propagate the total end score from the leaf node, back up the tree;
vii. The current node is reset back to the start state and the algorithm performs another iteration;
viii. Once complete the agent will choose the node with the highest back-propagated value as the next best move;
ix. The move is completed, and the system goes back to step 1 until there are no more moves left on the board.

In the context of the game '2048' there were two adjustments I made to the vanilla MCTS implementation. The first was adjusting the traditional Upper Confidence Bound formula (UCB1) (Kocsis & Szepesveri, 2006).

$$UCB1 = V_i + 2\sqrt{\frac{\ln N}{n_i}}$$

Figure 2 – Traditional UCB1 formula

The normalising constant of 2 is designed to be used in zero-sum games where the score is (0, 1) and no prior knowledge of reward distributions is known (Browne, et al., 2012). As such, I experimented with different values to encourage exploration of potentially higher scoring areas of the game tree. I eventually settled on a value of 20 as the optimum square root coefficient for this version of MCTS, the reasons for which are outlined in the results section below.

The other variable, which had a beneficial effect on the time taken for the agent to make a decision, was limiting the depth to which rollouts can simulate a game. This meant that most rollouts didn't actually complete a game simulation but rather returned the value of the board after reaching a certain depth limit. This had two major effects on the agent:

i. It dramatically reduced search times by giving the agent a shallower view of the search space.
ii. It also increased the mean average total score, slightly counter acting the effects of random tiles dropping throughout the game.

Mean Average Simulation-based Search (MASb)

As mentioned above, the traditional approach to MCTS was not producing good results, and so I began to explore some different approaches the result of which was the following 'Mean Average Simulation-based Search'.

Here are the steps I used when applying MASb to 2048:

i. The system drops a random tile.
ii. The agent takes the current state and calls the next states function to obtain a list of all the current states children.
iii. A pre-determined number of rollouts (a variable set by the user) is then performed on each of the children.
iv. When all rollouts have completed for a particular child the mean average total score is used to assign a value to that particular child node.
v. This is repeated for every child from the current state.
vi. The system then picks the move with the highest mean average total node value.
vii. We then go back to step 1 and continue until no more moves can be made.

The results section below demonstrates the dramatic improvement in both individual tiles and mean average total score when applying this technique, with further improvements made when combining both approaches.

One point to note is that combining the two techniques involves a careful balancing of parameters, as the number of total game simulations taking place will be:

*total rollout value * the number of rollouts from individual nodes*

If not careful this factor can cause the system to take an unacceptable amount of time to decide on a move.

Also demonstrated below is the requirement to limit the depth rollout simulations for MASb. Although the depth limit has less of an impact than on MCTS, it shows that decreasing the depth at which the tree can search in MASb will drastically improve computational speeds while having a negligible effect on overall total score.

3. Results

Vanilla MCTS - Adjusting UCB formula normalising constant

The following results demonstrate the agent playing 20 games per constant value change. The following parameters were maintained throughout;

i. Total Rollouts – 1000
ii. Search Depth – 5

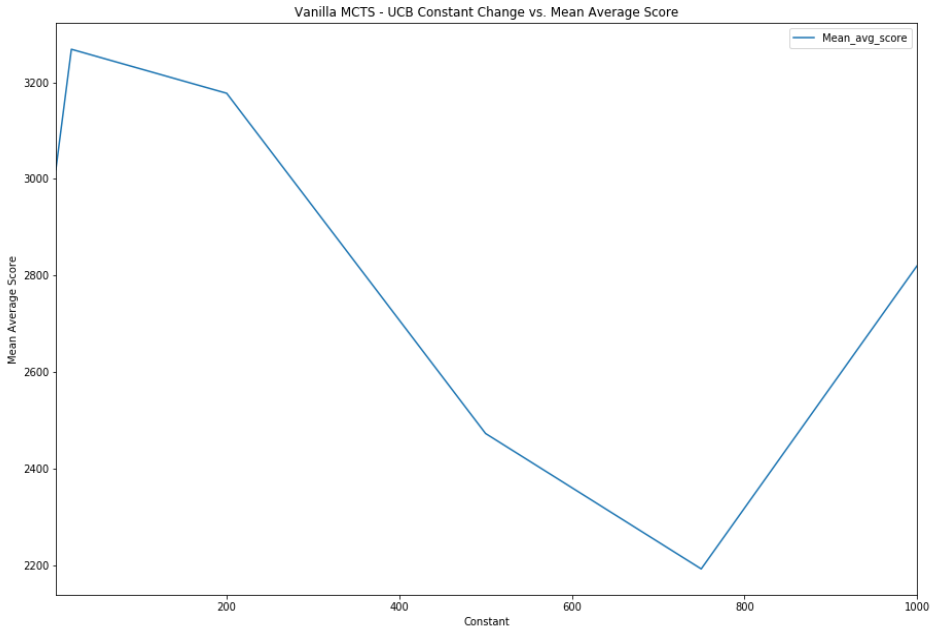

Figure 3 – Vanilla MCTS – UCB Constant vs. Mean Average Score

As is clear from figure 3, changing the traditional normalising constant of 2 had a negative effect on overall mean average total score. As such, the above graph indicates that a coefficient of 2 will maintain high mean average scores.

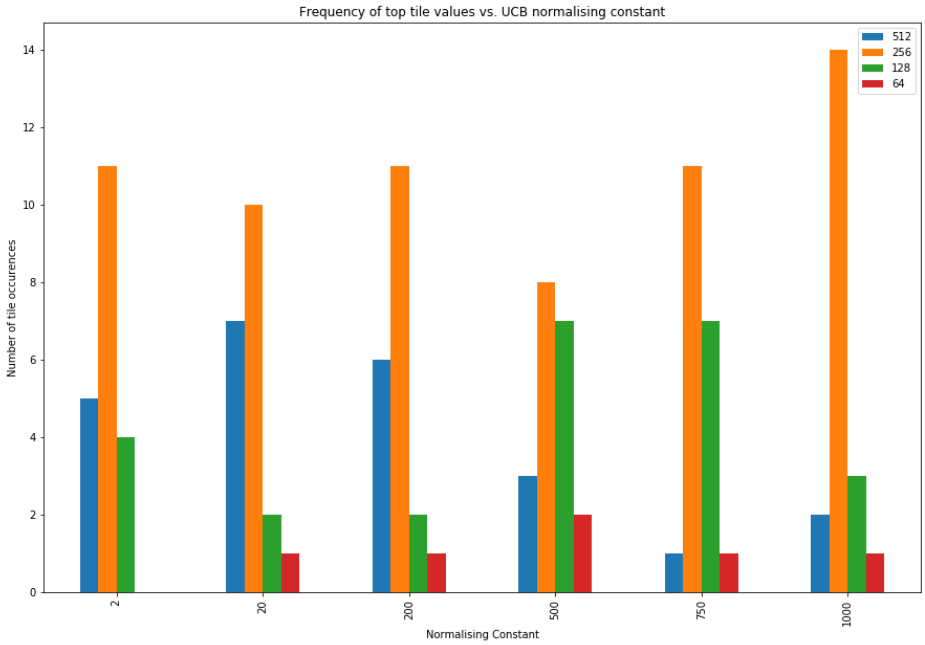

Figure 4 – Vanilla MCTS, Tile Frequency vs. UCB Constant

However, an interesting consequence of adjusting the constant value was the effect on the occurrences of high-value tiles. Figure 4 shows that by maintaining a constant value of 20, the agent has a higher likelihood of achieving the '512' tile (the highest tile achieved throughout this iteration of 20 games).

This constant value of 20 is supported in figure 3, only showing a shallow decline in average total value, indicating this would be a good balance to optimise scores and high value tiles over a number of games.

The mean average score achieved using this constant value was 3268 with a high score of 6100.

Vanilla MCTS - Adjusting Depth Limit

The following results demonstrate the agent playing 20 games per depth limit value, with the following parameters maintained throughout;

i. UCB Normalising Constant – 20
ii. Total Rollouts - 1000

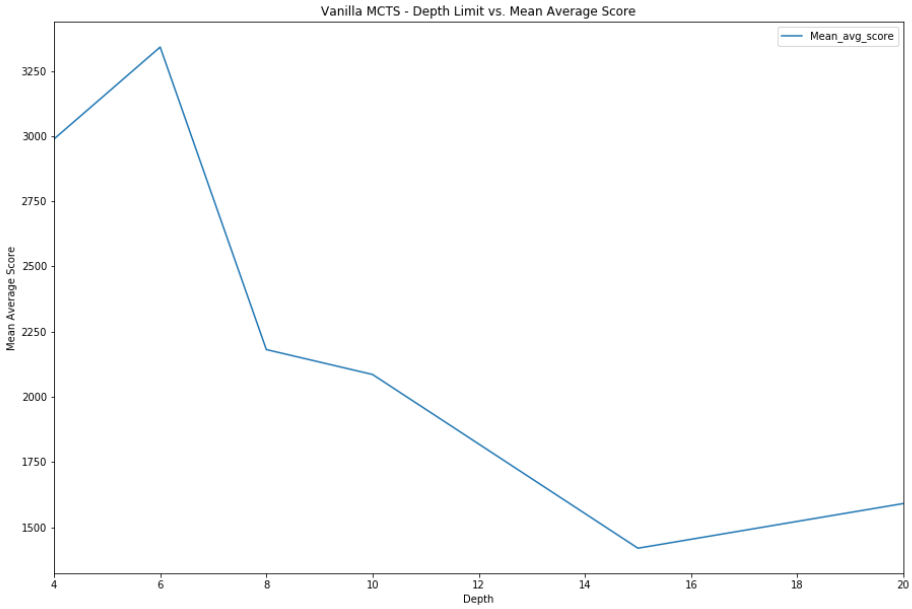

Figure 5 – Vanilla MCTS, Depth Limit vs. Mean Average Score

In an effort to increase total scores I experimented with different depth limits. As figure 5 shows, the optimum depth-limit to achieve maximum mean average total score was 6. I can only surmise that as the search doesn't take into account the multitude of random moves that can occur throughout the game, the error margin is increasing proportional to the depth at which the search explores.

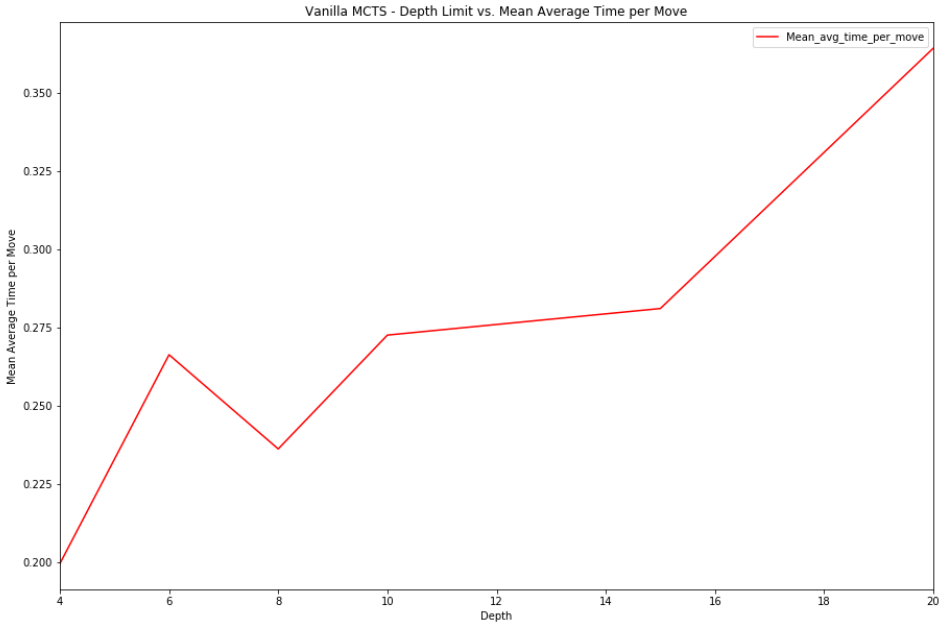

Figure 6 – Vanilla MCTS, Depth Limit vs. Time Per Move

Figure 6 above works in conjunction with Depth vs. Mean Average Score, in that it visualises the increase in total time per move as we increase the depth limit. Depth limit 6 appears to be the perfect balance between optimum high score and time taken to make a move. Also interesting is that after depth level 14 the time taken to decide starts to increase rapidly, indicating that to achieve reasonable decision times in 2048 a depth-limit should be applied to rollout simulations.

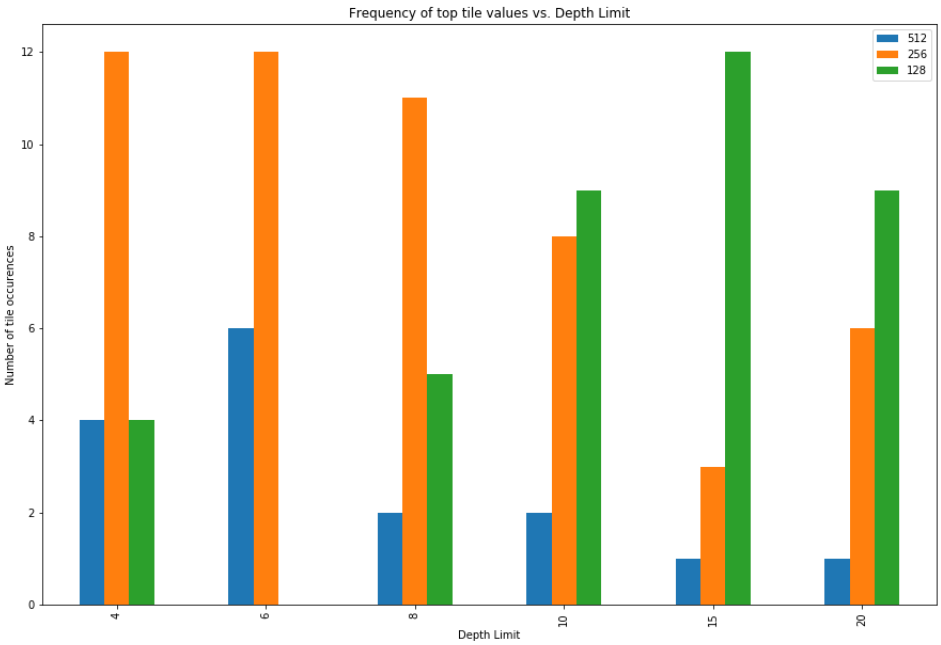

Figure 7 – Vanilla MCTS, Tile Frequency vs. Depth Limit

As further evidence that depth level 6 is the optimum setting we can see in figure 7 that this implementation will also have the greatest probability of scoring a high value tile (512 being the highest value tile achieved throughout this grouping of games).

The mean average high score at this depth was 3341 with highest scoring game of 6304, with a 30% chance of scoring the 512 tile.

Mean Average Simulation-based Search (MASb) – Depth limit increase

Following on from experimenting with different parameters using MCTS, I began to examine the scoring and tile values for MASb, noting that the total score and tile values increased immediately. Throughout these experiments I maintained constant variables:

i. Total Rollouts – 4 (given the user only has four possible moves from the current state this means that all node children are sampled once).
ii. Leaf-Rollouts – 200 (number of simulation games played from a single child node).

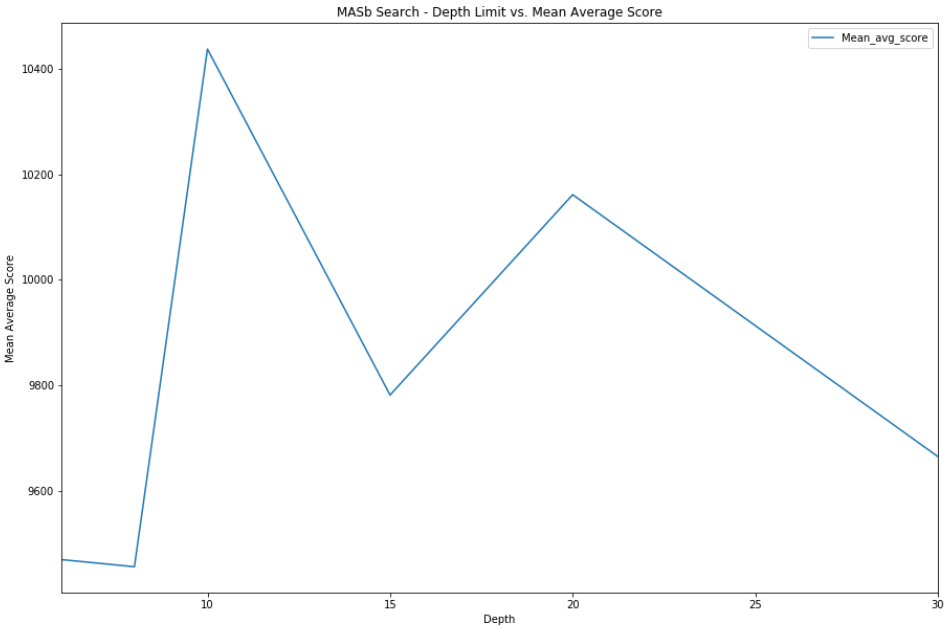

Figure 8 – MASb Search, Depth Limit vs. Mean Average Score

First thing to note is the increased mean average score, particularly at depth 10. Throughout this round of games, the highest maximum score achieved was 13128, with a mean average score of 10438.

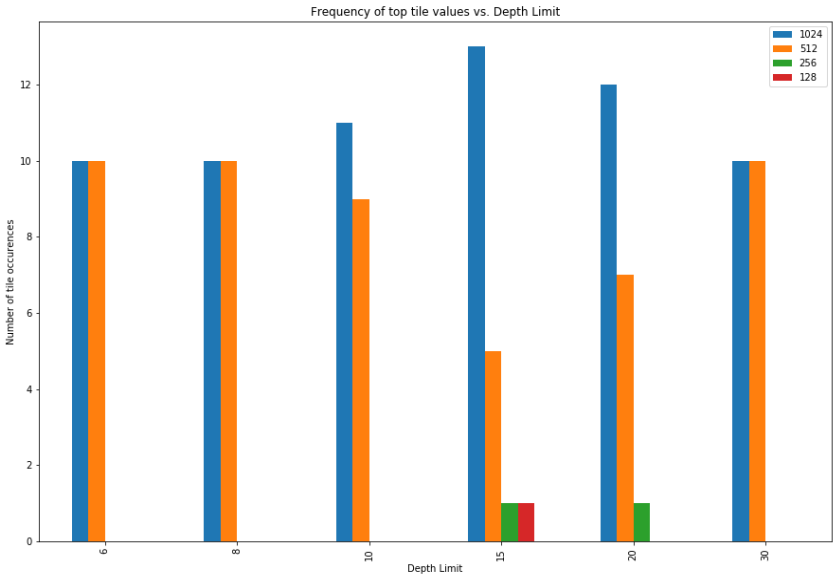

Figure 9 – MASb Search, Tile Frequency vs. Depth Limit

Perhaps not surprisingly, a greater value of tile was achieved when searching at a deeper level (figure 9 showing a 65% chance of achieving the 512 tile at depth level 15). At this depth the highest score achieved was 14220, but the average score over all 20 games drops to 9781. However, due to the considerable increase in the highest value tile achieved throughout this round of games I'd consider depth level 15 as optimum, providing an excellent score vs. top tile ratio using vanilla MASb.

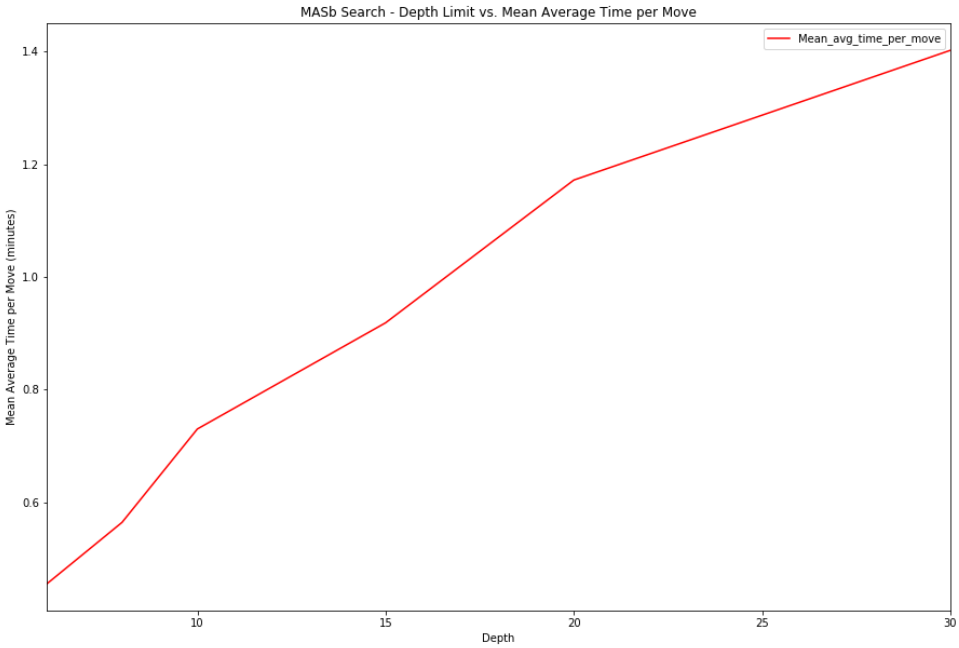

Figure 10 – MASb Search, Depth Limit vs. Time Per Move

This hypothesis is further supported when examining the Depth vs. Time per Move graph above. The difference between depth level 10 and depth level 15 is 0.003 seconds (0.2 minutes / 60) and as such has an arbitrary effect on the time taken for the search to make a decision.

Mean Average Simulation-based Search (MASb) – Leaf-rollout increase

Further experimentation was undertaken to maximise scores and tile values by adjusting the number of rollouts from a child node. The following represents 20 games per rollout iteration change and maintained the following parameters:

i. Total Rollouts – 4 (so that the children of the current state are all sampled once);
ii. Depth Limit – 7

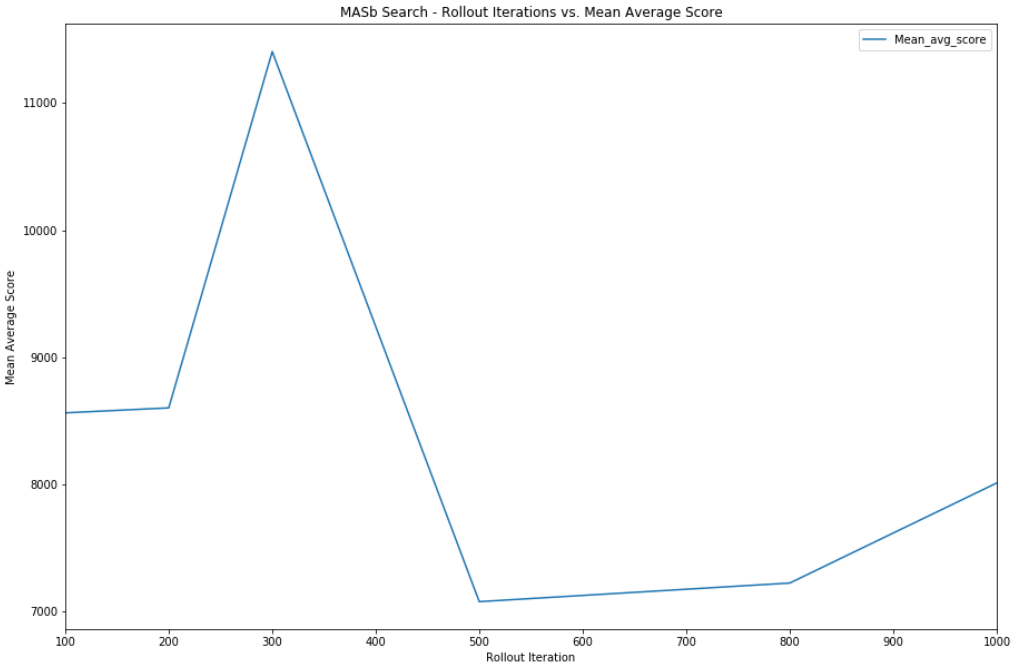

Figure 11 – MASb Search, Rollout Iterations vs. Mean Average Score

The figure above shows the continuance of this technique to provide much better results. Interestingly, the rollout iterations reach a convergence point (at 300 iterations) after which the accuracy of the agent to predict high scoring moves drops dramatically.

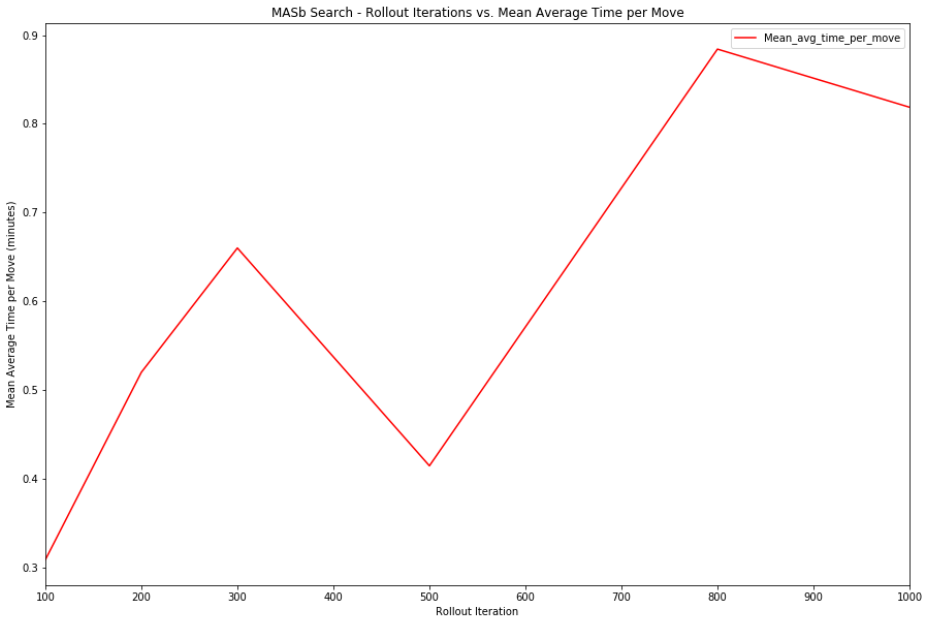

Figure 12 – MASb Search, Rollout Iterations vs. Time Per Move

This evidence is further supported in figure 12, indicating that the rollout value of 300 provides the best time per move / mean average score ratio, taking 0.011 seconds to decide on the next best move. Throughout the round of 20 games with the rollout value set to 300 the total mean average score was 11405 with a highest scoring game of 15500.

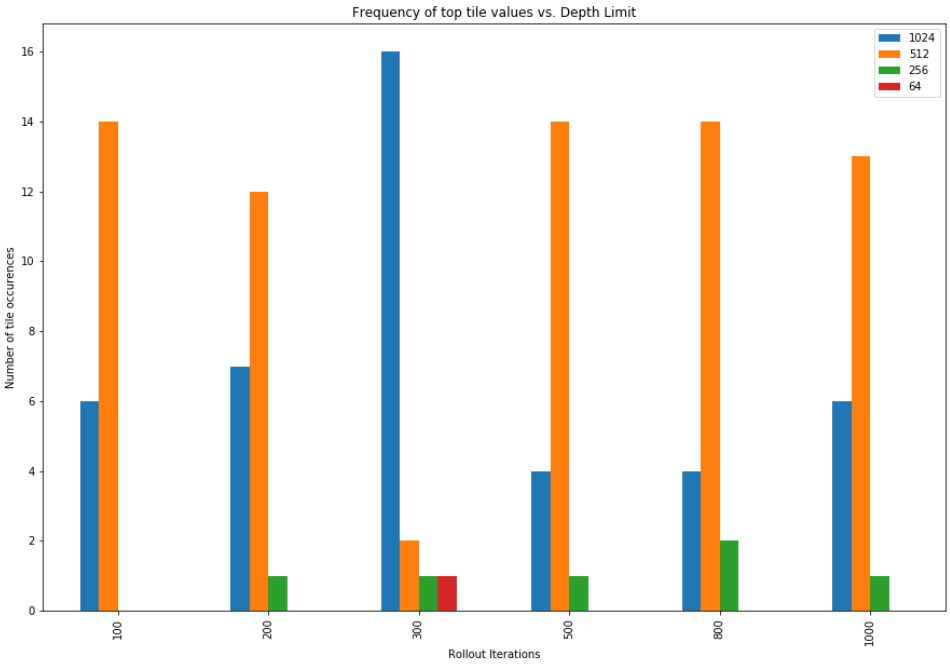

Figure 13 – MASb Search, Tile Frequency vs. Depth Limit

The final point to note regarding the rollout iterations also supports the hypothesis that 300 is the optimum number of simulations to play from each child node of the current state. Figure 13 shows that the agent has an 80% chance of achieving the highest value tile (1024 being the highest tile during this round of games).

Hybrid Models – MCTS & MASb Combination

After much experimentation I achieved the best results when combining both approaches, using a combination of multiple leaf rollouts, and building the tree like a traditional MCTS approach.

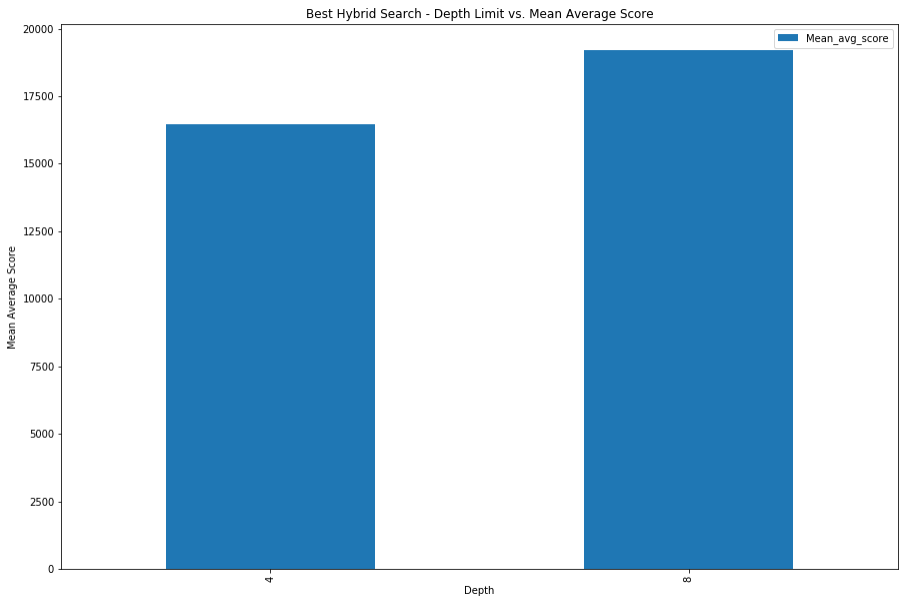

Figure 14, Hybrid Model, Depth vs. Mean Average Score

It's clear to see that, using the hybrid approach, the mean average total scores took another jump upwards. Figure 14 represents two groups of 20 games with different hybrid parameters. The highest mean scoring set of games (on the right) had parameters thusly:

i. Leaf rollouts (MASb) – 200
ii. Total rollouts (MCTS) – 6
iii. Depth limit – 8

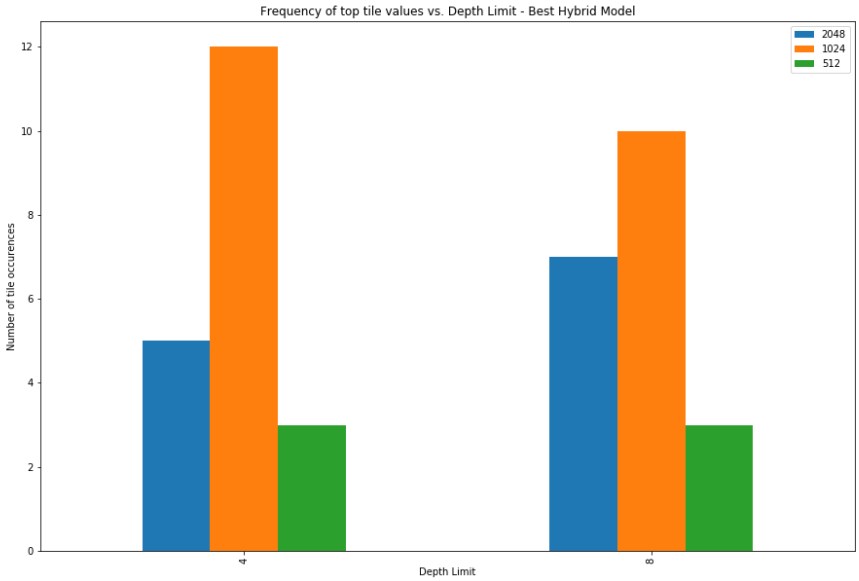

Figure 15, Hybrid Search, Tile Frequency vs. Depth Limit

Throughout this series of games, the highest score achieved was 35196, with a mean average total score of 19193, and a 35% chance of achieving the winning 2048 tile (figure 15). Further evidence of the benefit of a hybrid model can be found in the top tile value frequency for each set of games in the figure above. Considering the purpose of the game is to achieve the 2048 tile, the graph above indicates that the second set of game parameters will result in a higher likelihood of the agent achieving success.

4. Conclusions

From the evidence outlined above it is clear that traditional Monte Carlo Tree Search is an ineffective way to navigate the game tree of 2048. MCTS itself cannot make good decisions in a reasonable time. It has been shown that, when using MCTS in combination with Deep Reinforcement Learning, the search can be more effective at achieving high scores and high individual tiles. (Amar & Dedieu, 2017)

Our experimentation with Mean Average Simulation-based Search shows that by simulating games from only the level one nodes you can cut down on compute time and also increase top scores and top individual tiles. This technique appears to help mitigate the effects of random tiles being dropped by the system.

With regards to 2048, an ensemble method, combining both approaches outlined above (MCTS & MASb) provided the best results.

Further experimentation with simulation and probability-based techniques combined with Deep Reinforcement Learning would be a natural next step.

Acknowledgements

It must be acknowledged that the core idea behind MASb was conceived by Barry Smart, with contributions from myself regarding the optimum value to assign a child node. We both experimented extensively with the MASb technique, as well as traditional MCTS.

Works Cited

Amar, J. & Dedieu, A., 2017. *www.mit.edu.* [Online]
Available at: http://www.mit.edu/~amarj/files/2048.pdf
[Accessed 04 January 2020].
Browne, C. et al., 2012. *A Survey of Monte Carlo Tree Search Methods.* [Online]
Available at:
http://www.incompleteideas.net/609%20dropbox/other%20readings%20and%20resources/MCTS-survey.pdf
[Accessed 05 January 2020].
Rodgers, P. & Levine, J., n.d. *An investigation into 2048 AI strategies,* Glasgow: University of Strathclyde.
Wikipedia, 2019. *www.wikipedia.org.* [Online]
Available at: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
[Accessed 4 January 2020].
Wikipedia, n.d. *www.wikipedia.org.* [Online]
Available at: https://en.wikipedia.org/wiki/2048_(video_game)
[Accessed 05 January 2020].