

# General game-playing applied to 2048

Paul Lisker and Ahmed Ahmed

December 17, 2016

## 1 Introduction

General Game Playing (GGP) is a sub-field of Artificial Intelligence that seeks to develop agents that are capable of playing many games rather than being tailored and limited to particular game. GGP has been a popular subject of AI research, particularly over the past decade. The goal of GGP is to create programs that are capable of successfully playing more than one game successfully. As such, a successful GGP agent will not have algorithms tailored to a particular game, but rather generalized algorithms that can be successful when applied to various different games (though occasionally of the same domain—that is, games that are similar in nature). For an illustrative example on the difference between a regular game playing agent and a GGP agent, we can look at IBM’s famous Deep Blue computer. This computer ran an AI program that was tailored to the goal of winning a game of chess; however, this program would have been unusable for other games. On the other end of the spectrum, successful GGP programs can play—albeit perhaps less effectively as a dedicated algorithm—more games than just one, or play games that would otherwise be intractable for specialized algorithms.

The requirements of General Game Playing, then, necessitate the use of algorithms that are not domain-specific. In other words, it must use algorithms that can operate with a broad spectrum of games without being dependent on shared rules between said games. Among this class of algorithms is the notable GGP algorithm of Monte Carlo Tree Search (MCTS), a class of algorithms for finding optimal choices in a decision space by taking random sample simulations and building a search tree according to their results. We explore this algorithm with more in Section 2. Among other benefits, MCTS provides an alternative to the traditional approach to combinatorial games—MiniMax algorithms—especially in games of nontrivial size or for those in which no reliable heuristic has been developed, two situations in which MiniMax often fails.

In this project, we apply a version of a Monte Carlo Tree Search algorithm to the domain of games surrounding the popular puzzle game *2048*. We also analyze the algorithm’s

performance, and, in the case of the original *2048* game, compare its effectiveness to published heuristic AI approaches.

## 2 Background and Related Work

### 2.1 Introduction to 2048

*2048* is a single-player puzzle game developed by Gabrielle Cirulli that was a viral hit in 2014[?]. This game consisted of a  $4 \times 4$  grid in which each cell is either empty or contains a tile with a power of two number. In each move, the player chooses a direction in which to slide all the tiles with the goal of having tiles of the same number collide to merge and create a tile of their sum. After each move, a new tile is spawned randomly in any empty cell. The game's goal is to reach the eponymous tile; it continues until (1) there are no more available moves—no more merges possible or open tiles—or (2) the 2048 tile is reached, though at that point the user is given the option to continue beyond to form higher tiles.

For this project, we expand the domain of the game *2048* by creating what we refer to as *generalized-2048*. This class of games include modifications to the grid size of the original game and to the scoring policy. As such, though each game is indubitably similar to the original game, each has its own quirk and strategy that requires unique strategies for success.

### 2.2 Algorithm: Monte Carlo Tree Search

The Monte Carlo Tree Search is among the most important algorithms in the field of General Game Playing today; its use "has been widely adopted in GGP research" [?]. The main impetus for using MCTS as a principal algorithm in the building of GGP programs is its wide adaptability across games and the proven success in games that, until then, had proven intractable.

In 2006, Monte Carlo Tree Search came to prominence after it was successfully used to progress in the building of agents that could play in the game of Go. At the time, AI agents were competitive against humans with small boards, but not yet competitive at standard  $19 \times 19$  boards. However, since then, Google's AlphaGo initiative has created a program successfully integrating deep neural networks with an MCTS approach that, earlier this year, became the first AI winner against a professional, human player [?, ?]. Despite its clear and proven advantages, however, "the main disadvantage of MCTS is that it makes very limited use of game-related knowledge, which may be inferred from a

game description” [?].

MCTS is essentially a class of techniques for traversing game-trees. Such game trees are an especially convenient model for combinatorial games such as *2048*—that is, sequential, discrete, full-information games. Therefore, this project most directly draws off of the concepts of uninformed and heuristic search that we explored earlier in this course. Additionally, as its name implies, the MCTS algorithm does have a Monte Carlo component: it uses sampling in order to inform its search. This concept of using of sampling methods to find approximate answers to otherwise intractable problems was explored in this course in the context of particle filters in Bayes Nets.

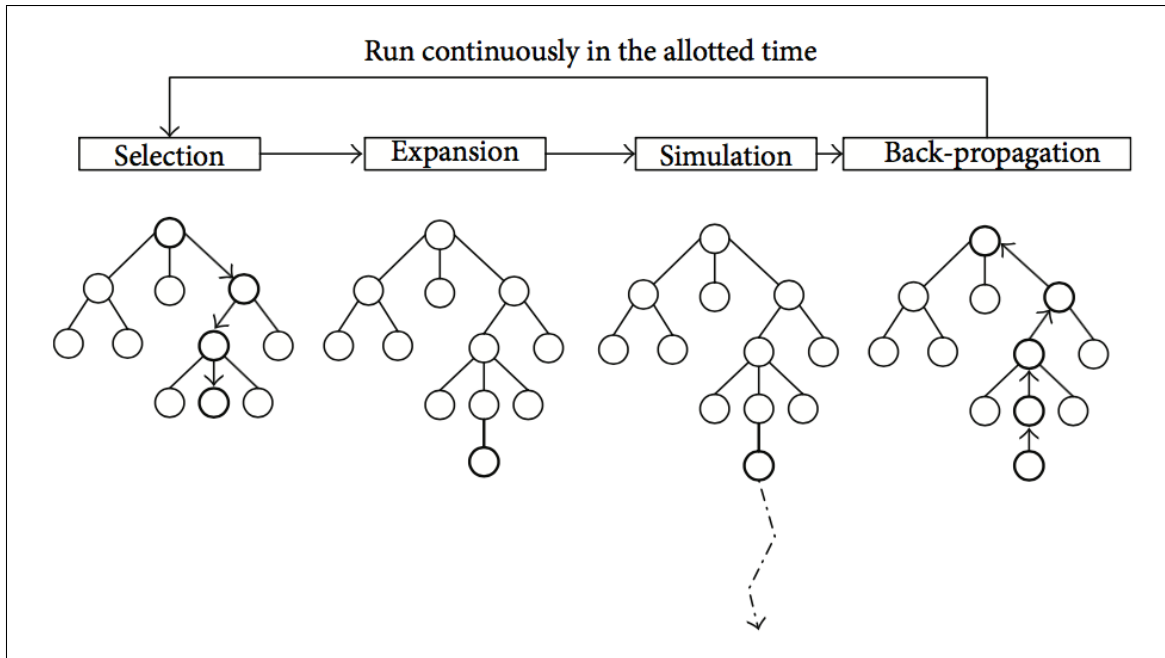
Monte Carlo Tree Search algorithms proceed in four main steps: (1) selection, (2) expansion, (3) simulation, and (4) back-propagation. As the algorithm searches the game tree from its current state, it proceeds with each of these four steps in each iteration until some computational limit is reached—typically a certain time limit, or number of iterations—at which point the best discovered move is chosen. A graphical representation of the algorithm can be seen in **Figure X**.

**Selection** In this first step, the algorithm—beginning at the root node—proceeds along the expanded tree to find a node to expand from the built game tree. This selection typically takes place with the use of some confidence algorithm. A common one—and the one used in our project—is the Upper Confidence Bounds applied for Trees (UCT) algorithm. Ultimately, this results in actions that have been more effective in the past being tested more frequently.

**Expansion** With the node selected, the algorithm proceeds to expand the game tree one step with the available moves.

**Simulation** From the new node(s) added, the algorithm proceeds to run a random simulation until a termination step is reached.

**Back-Propagation** The ultimate score achieved by the simulation is then back-propagated to the nodes visited in the Selection and Expansion steps for use in the decision-making steps of the next iteration of the algorithm.



**Figure X.** The four key steps of the Monte Carlo Tree Search algorithm are represented graphically for a given game tree.

## 2.3 Published solutions

As the game went viral in 2014, the AI game-playing community went to work at publishing several automated 2048 players. Most approaches that the authors are aware of have been minimax implementations, so most focused on developing useful heuristics and optimizing for faster run times in order to increase the depth of their minimax searches within reasonable time. Our implementation distinguishes itself from these solutions, in that it doesn't use minimax at all, instead using MCTS as our primary method.

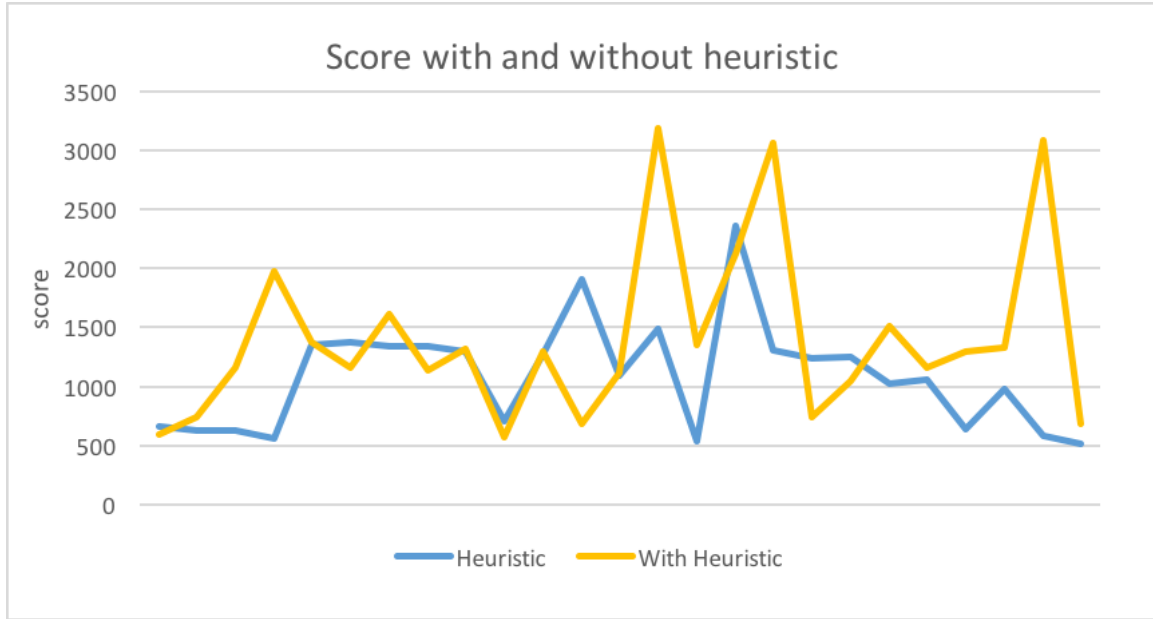
## 3 Problem Specification

## 4 Approach

## 5 Results

In order to confirm that our implementation was in fact following the algorithm as expected, we recorded the average score received by our AI at a number of different computational benchmarks in experiment, with the expectation that, if this AI indeed conducts Monte Carlo tree search, this would be a monotonic increase. This was indeed the case as

Figure 1: Comparing average scores of UCT with and without heuristics



you can see Figure 1, at a 1 iteration limit, the AI receives an average score of 987, just below the mean score of a random player, but as the number of iterations allowed increases, the AI steadily improves. In fact, in our tests, our AI was able to beat the random player consistently with as little as 4 iterations. This too, is consistent with UCT, as 4 iterations would allow the algorithm to sample all possible moves in order to inform its search.

Ultimately, with an 100 iteration limit - about a second per move on our machines - the AI earned an average score of 9015, significantly higher than the random strategy. In nearly all trials, the AI earned at least a 512 tile, in the majority of trails (71%) it acheived the at least the 1024 tile, and in a small fraction (9%) of trials, the AI earned a 2048 tile and hence beat the game.

Additionally, the AI was able to play reasonable well in various variations of generalized-2048. Though the mean score varied between the scoring systems, the AI's intelligent behavior is indicated by the result that the average number of moves in the standard, sum and logsum scoring standards are all greater than the 109.1 average moves of the random player (161,151.8 and 152.6, respectively).

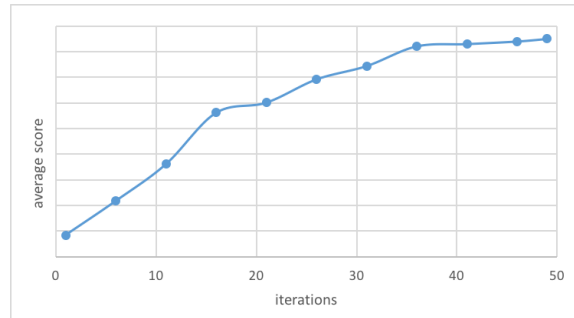
Even further, we tested the standard UCT player without any domain-specific knowledge against one which incorporated the monotonicity heuristic into its tree policy. The result shown in figure 2 is that, with time held constant, the heuristic AI performed significantly better on average, with a mean score of 1412.16 as opposed to 1085.29 without the heuristics. We also note in Table 1 that the variance of scores with the heuristic decreases

	Average Score	Variance
Random	1103	8.2 e4
UCT alone	8457	5.6 e5
UCT with heuristic	9015	2.0 e5

Table 1: UCT with heuristic performed consistently better than standard UCT

significantly, indicating that, in addition to increasing the average score, the incorporation of the heuristic also made the predictions of the AI more accurate, leading to more consistent final results.

Figure 2: Positive correlation between performance and number of iterations between



## 6 Discussion

Overall, the AI player significantly under-performed in comparison the several published minimax implementation. Several weaknesses in our implementation likely contribute to this. Most significantly, perhaps, the implementation is relatively slow, constraining the amount of iterations it can conduct for its planning in a reasonable amount of time for each move. As the trend in figure 1 indicates, an increase in the amount of iterations allowed would likely lead to an increase in performance. In future implementations we aim to reduce this weakness via optimizations such as parallelization of the rollout simulations and using a faster language, such as C++, with which we can utilize more efficient data structures.

Due to these shortcomings, our results do indicate that MCTS methods are not a viable method for 2048 AI's. One reason for this may be the relatively low branching factor of 4 in its game tree, especially when compared to a game such as Go, which has a branching factor of 250. Perhaps, for this reason, minimax is the superior algorithm in this case, since

a strict comparison of alternatives at each step is practically feasible and with the reasonable heuristics available, more effective. Our hope that depth of the game tree might tip the scale in the favor of MCTS methods turned out to be unfounded, perhaps due to the effectiveness of the heuristics. Another possibility is to incorporate learning in our MCTS search as did Rolet et al.[?], so that our AI can dynamically learn better heuristics into its tree search.

## A System Description

## B Group Makeup

Appendix 2 Ahmed Ahmed and Paul Lisker

## References

- [1] Gabrielle Cirulli. 2048.
- [2] Google. Alphago.
- [3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 2016.
- [4] Maciej Uwiechowski, HyunSoo Park, Jacek MaNdziuk, and Kyung-Joong Kim. Recent advances in general game playing. *The Scientific World Journal*, 2015(2015), 2012.