Analyzing Heuristic Functions for an Automated 2048 Agent

Jeff Miller*
mill9039@umn.edu

1 Abstract

The research conducted in this paper was done to find the best heuristic function option for an automated player of the game 2048. This was done by selecting a variety of heuristic functions that had been tried in other studies and having an agent play multiple games using the Minimax algorithm and each of these functions to determine which one it would play best with. The results were collected and analyzed, leading to the findings that the two best heuristic functions were based on weighted empty tiles and board smoothness, but also indicated that when these are combined the agent plays better and more consistently than when either of the heuristics are considered alone.

2 Introduction

The game 2048 gained a huge following when it was initially released. The game is simple enough for users to intuitively be able to pick up and play, challenging enough to make winning feel rewarding, and has a level of chance that makes every game feel different, while still allowing the player to feel in control of the game's outcome. Because of all this, the player base has remained very high and it has become one of the most recognizable mobile games on the market. Naturally, due to the popularity of the game, there have been numerous studies on the best way to win the game and which algorithms would work the best. Popular games have a tendency to motivate researchers to discover the optimal ways of play through machine learning and artificial intelligence. One of the most famous examples of this is the story of AlphaGo, which will not be discussed further, other than to say that this research led to a greater understanding of the field as a whole, plus it spreads awareness of the subject into more people's lives, thus advancing the research further still. For this reason, spending time studying the interactions of artificial intelligence on games such as 2048 can be more important and impactful than some may initially believe. To conduct this experiment there were four main phases: the background research phase, which gave ideas to solve the problem and provided a better overall understanding, the coding phase, which included writing the scripts for the game, agent, and a way to iterate over the different heuristics, the data analysis phase, which included collecting all the data and reviewing it to determine the results, and the writing phase, which included writing this paper to display all of the findings and results.

2.1 Problem

The problem that this paper is trying to solve is finding which method of heuristic evaluation is best when creating an agent to play the game 2048. A heuristic function is a function that takes in the current state of the game and returns how good the possible actions are in a quantifiable way. In the case of 2048, this means that at the beginning of every turn, the board must be evaluated by the agent in order to determine which of the possible actions is best for achieving the goal, which, in this case, is to win the game. This is not always a straightforward problem to solve, however. In the case of 2048, there is an innate randomness to the game due to a new tile being added to a random empty space with a value of either two or four. This means that the agent cannot perfectly predict what will happen in the turns following its action, adding a layer of complexity to the problem while still leaving the agent enough control over most of the board to make educated plays. By finding which method of evaluating is best, researchers can use these findings to build more efficient algorithms that utilize the heuristics to produce faster and better-performing agents to surpass what is currently considered to be best. There are many different ideas and strategies to consider when trying to decide which method to evaluate each move by, so a variety of differing heuristic methods must be tested. These different strategies were found during the research phase and were all considered to be successful to some degree by their initial researchers. By compiling these methods and running them against one another in a controlled setting with the same overarching algorithm as well as game rules, it was possible to directly compare them against each other to find which ones allowed the agent to perform best. It is also important to have an idea of a baseline to ensure there is a consistent method to compare each new heuristic function.

2.2 Hypothesis

The hypothesis going into this problem was based largely on what the common strategy is among people who play the game say is best. It was believed that the best heuristic function would be rewarding to an agent that makes a move to keep the highest value tiles in one corner and along the edges that meet to make that corner. When most people play the game, this is the dominant strategy that is seen to earn

*All authors contributed equally to this report.

higher scores and win the game, so the hypothesis was built on the assumption that this would hold when an artificially intelligent agent was deciding which action to take. Following the background research conducted for this problem, the hypothesis was slightly modified to take into account another idea that seems to come intuitively to humans, but not so to computers. This idea is that, ideally, each square should be adjacent to a square that has double its value, so that when it can be merged with itself, it can immediately merge with that adjacent tile and quickly chain into larger values. This trait will be defined as the smoothness of the board for the rest of the paper. Following this finding, the hypothesis was modified to predict that the ideal heuristic function would keep the highest value tiles in the corner and edges but also exhibit a high degree of board smoothness.

3 Background Research

This paper will analyze some of the recent efforts to earn a high score and review some of the knowledge gleaned from investigating previous research done on the topic. The main themes that are focused on are: algorithms used, heuristics used, and other methods for better solving. All of these considerations will be cross-examined and weighed based on what the authors found for each. This information helped inform decisions made during research and provides insights into possible expansions on the research conducted in this paper.

3.1 AI Algorithms

Researchers of this topic have attempted to achieve winning results with multiple artificial intelligence(AI) algorithms to varying degrees of success. Some of the most popular algorithms used are Minimax, Expectimax, and Monte-Carlo Tree Search(MCTS).

The paper titled *An Investigation into 2048 AI Strategies*[5] considered using the Minimax and Expectimax algorithms, but they instead opted to do their research on the Monte-Carlo Tree Search and the Averaged Depth-Limited Search(ADLS) algorithms. They compared their findings with these algorithms to what others got with the Expectimax algorithm. They found that their ADLS performed better than their MCTS, so long as there was an evaluation function that had knowledge of the board. They also found that these evaluation functions did not help to improve the score of the MCTS, though the authors do admit that this is an area they would like to research further. The authors also note that in order to score better maximums, the algorithm may need to sacrifice some of its scores on average. In other words, a risk-seeking algorithm may be better in some ways but does not always lead to overall improvement.

Pedagogical Possibilities for the 2048 Puzzle Game[3] is an article that also aims to compare multiple of these algorithms; they chose to focus their efforts on MCTS and Expectimax, but they also referenced their results against a random player and a simple greedy player. They also ran their algorithms with multiple different depth limits, which affected the score of the agent greatly. They found that MCTS run at a depth of 3 and Expectimax at a depth of 2 perform at roughly equal levels, though once the depth of Expectimax is increased, the score dramatically increases as well. Both of these algorithms score much better than either the greedy agent or the random player, and the greedy agent does much better than the random player. They note that Expectimax is a good choice in part because of the randomness of the tile getting added at a random place and given that the value of the added tile can be either a 2 or a 4. Expectimax performs well with maximizing the minimum scores given some chance occurrence, so it does make sense to be one of the top contenders.

Another article, *USING ARTIFICIAL INTELLIGENCE TO CREATE A SOFTWARE AGENT FOR SOLVING THE 2048 GAME*[4], conducted their research using a Depth First Search method, followed by both Minimax and Expectimax. Their simple Depth First Search algorithm did much worse than the other two options, which were both able to win the game a majority of the time; however, the Minimax algorithm required a depth of 8 to win a majority while the Expectimax algorithm only needed a depth of 3, implying that it was able to run and win more efficiently, though they never made any comment on the speed of these algorithms. Still, this is yet another article suggesting that Expectimax is the best algorithm to use in this case, which is a theme that emerges across the literature and makes it seem as if this is the best option overall.

After analyzing these papers and comparing results, it seems as though the Expectimax algorithm is consistently among the best options to use to efficiently win the game, which makes sense given the randomness of the game that it can take into account and the lack of any direct adversarial player, which is more a strength of the Minimax algorithm.

3.2 Heuristic Functions

While the algorithm used to play the game is a very important part, it is perhaps equally important to have a good method to evaluate each move that the agent is taking into consideration. Different ways of considering these functions can drastically increase or decrease performance, no matter the algorithm. While there are multiple popular methods of evaluating heuristics, there certainly are some that emerge as better than others.

The paper *Minimax and Expectimax Algorithm to Solve 2048*[7] introduces multiple basic ways in which the algorithms can score and evaluate each move. One of the methods that the authors introduced was maximizing empty spaces on the board, another was "smoothness", which means that many values are neighboring a value that is a direct multiple of itself, and one that focuses on keeping the larger values along the border of the board. In their published tests, they attempted to implement all of these functions into one, which would give their algorithms a final score. They note that there is still a lot of room to tweak this heuristic function to improve results and note some things that others have done that could be used to better their results. In the article, the authors mention that the depth of the searches impacts the time to make decisions quite quickly. The jump from a depth of 4 search to a depth of 8 search runs increases the time to play a game by 5-10 times. Because of this, they tested on a depth of 4 search, despite the clearly worse performance. This is not mentioned by many other authors in this review, but many of them have opted to keep their depths of search in a similar range, likely for a similar reason.

From AlphaGo Zero to 2048[8] is a paper that introduces a heuristic function that rewards the agent for creating a board that they call "snake-shaped". Essentially, the agent will attempt to keep the highest score in one of the corners and have the values decrease along a row or column until the next corner, where it pivots and goes back along the row or column with the values still decreasing. This shape, when traced, is reminiscent of the classic game "Snake", hence the name. They did not use this heuristic as a comparison against other heuristic functions, though they did achieve fairly competitive results with it with the Minimax, Expectimax, and MCTS algorithms and it seems to take multiple other evaluation functions into account. The authors conclude that MCTS is the best algorithm to use and they trained their model in a similar vein to AlphaGo Zero. They were able to get their model of the automated player to achieve scores that were consistently better than those of human players, which is a feat that multiple other authors in this review struggled to accomplish.

Game Theory of 2048[2] is an article that dives deeper into the idea of making the board "snake-shaped", or "snaking" as this paper calls it. It describes that this idea came about from the basic idea of keeping higher values in the corners, but the problem that can lead to in the end game if the largest values have nothing to combine with. The author claims that "snaking" solves this problem by always ensuring the value, when combined, will be directly neighboring the value which it can combine with on the next turn. They then argue that this method is optimal because it is the way the board must be set up in order to achieve the highest possible score in the game when the board is filled.

While there are differing strategies that exist to optimize an automated player for the game, the one that emerges following a review of the literature is the so-called "snaking" technique. This technique derives ideas from both the "smoothness" concept as well as keeping the largest value in the corners. Keeping the high value in a corner is a common strategy used by people who play the game in order to achieve higher scores, so it makes sense that a winning heuristic evaluation would consider that somehow.

3.3 Other Findings

Optimization and Improvement for the Game 2048 Based on the MCTS Algorithm[6] is a paper done by researchers who extensively looked into how the parameters of their algorithms changed their success in winning the game. They utilized the MCTS algorithm and did rigorous testing to see which parameters performed the best for their study. They do not reveal this in the paper, but they note that they achieved what they consider to be advanced levels of play by human standards, that being that their agent got to 4096 more than 60% of the time. One interesting factor that they incorporated was the concept of an urgency variable. Based on how many tiles were empty, the algorithm would play more or less aggressively. One more thing that the authors noted was that they utilized C++'s ability to have parallel computing, which allowed them to quickly process the branches that were being expanded on in their search. By extension, this allowed them to also search deeper than many of the other papers in this review.

Composition of Basic Heuristics for the Game 2048[1] is another article that focuses mainly on finding the best heuristic function to win the game. They introduce concepts that many of the other papers also tested, such as a greedy solver, the number of empty tiles, and monotonicity, which is a similar concept to the "snaking" method mentioned earlier. Their findings were most in favor of emphasizing empty tiles along with having a high degree of monotonicity, which is similar to what some of the other papers found as well. One more noteworthy thing about this study is that the authors did some tests with a degree of randomness for the agent's choice. If the heuristic evaluation produced a tie, then they would choose a move randomly. This was a feature not described in other papers and, while it can be hard to directly compare its impact since they all use slightly different algorithms to make these choices, including the random element did seem to assist in raising the score. This only held when the randomness was used as a tiebreaker, not as a first choice in evaluating heuristics. By introducing this randomness in the heuristics, the game may be able to reach higher peaks of success, similar to hill-climbing algorithms. It is an interesting addition to the considerations that an agent makes when choosing an action and may be worth further investigation.

3.4 Comparisons

These articles help provide a clearer picture of what research in this area is like. Many of these articles draw similar conclusions about how best to develop the algorithm and evaluation functions that will make for the best-automated player. In the articles that included research using Expectimax, that algorithm consistently was considered to be the best, followed by MCST. It also seems well-documented that the strategy of "snaking" is considered to be one of the best, if not the best, forms of heuristic evaluation. The only article that mentioned forms of evaluation that strayed from this trend was: *Composition of Basic Heuristics for the Game 2048*. This article had "snaking" listed as the second option behind prioritizing empty tiles. There were some unique features, as noted above, such as randomness, urgency, and parallel computing that helped some researchers get a better performance from their algorithms, though it is difficult to say exactly how much of a benefit each of these provided. Difficulty in one-to-one comparisons is a theme across all of the papers in this review, they all used slightly different algorithms, heuristic functions, search depths, and game iterations. This means it is generally hard to come to one conclusion of what methods work best, but it does give the advantage of having many different solutions attempted that can be sifted through to get an idea of what works well.

4 Approach

To study the problem, a comparison was made between multiple heuristic functions in order to compare their performance when playing the game. These heuristic functions were chosen based on the previous research conducted above and each of them was made to return a score on each possible move for each turn. It was important to write all of the code for this research from scratch to make sure it functioned in a specific and unbiased way across all heuristics, which may not have been the case when using other repositories that may have been built with one method in mind. This also allowed for better control of the exact parameters that were needed to conduct the research.

4.1 Algorithm

To ensure that they were being compared fairly and consistently, they were functions that would be called and scored from a Minimax algorithm. The Minimax algorithm was chosen for its relatively simple yet effective design that allows the agent to compare all of the heuristic functions in a very consistent way. The classic algorithm was only slightly modified to allow for the modular design of the experiment that required the option to have one or more of multiple modes be considered by the agent for each action.

The idea of the Minimax algorithm in this case is to take in the current state of the board and have the agent find the maximum value of minimum scores that can be achieved through recursively solving for the maximum and minimum actions. This assumes the board is the opposing player in the scenario, so it would be trying to keep the agent's score as low as possible. The algorithm used did not take into account the random tile that gets added after each turn, meaning that it would solve each turn until the board was either fully simplified or the depth limit of five recursive calls was reached. The depth limit of five was settled on as it provided the best scores within a reasonable amount of time, as going any deeper meant that the program would spend a significantly longer time deciding each move. This implementation also allowed the agent to check the scores from multiple modes at the same time and take these into account when taking an action. This was done by iterating through a list of "modes" that all returned scores, which would be summed together to make the decision. By allowing multiple modes to be considered at the same time, it was possible to test whether there was any benefit to combining

Algorithm 1 Minimax Algorithm

```
1:  function MINIMAX(board, modes)
2:      $bestMove$ := MaxValue($board$, 0, $modes$, 0, $None$)
    return bestMove
3:  end function
4:
5:  function MAXVALUE(board, prevScore, modes, depth, prevAction)
6:      if depth = 5 then return prevAction, prevScore
7:      end if
8:      $maxMove$ := 0
9:      for each action in actions do
10:         $newBoard$ := MakeMove($board$, $action$)
11:         $score$ := 0
12:         for each mode in modes do
13:             $score$+ = CheckScore($mode$)
14:         end for
15:         $minMove$ := MinValue($newBoard$, $score$, $modes$, $depth$+1, $action$)
16:         if minMove > maxMove then
17:             $maxMove = minMove$
18:         end if
19:     end forreturn maxMove
20: end function
21:
22: function MINVALUE(board, prevScore, modes, depth, prevAction)
23:     if depth = 5 then return prevAction, prevScore
24:     end if
25:     $minMove$ := 0
26:     for each action in actions do
27:         $newBoard$ := MakeMove($board$, $action$)
28:         $score$ := 0
29:         for each mode in modes do
30:             $score$+ = CheckScore($mode$)
31:         end for
32:         $maxMove$ := MaxValue($newBoard$, $score$, $modes$, $depth$+1, $action$)
33:         if maxMove < minMove then
34:             $minMove = maxMove$
35:         end if
36:     end forreturn minMove
37: end function
```

heuristic functions and which combinations may be the best. The "CheckScore" function in the pseudocode is a simplified version where the score for the move is incremented based on the current mode selected for the agent. Throughout these recursive calls, the agent will keep incrementing the score until the end of the recursion comes along, then it will return the move that gives it the best score.

4.2 Heuristics

The main purpose of this research was to determine which heuristics, or combinations of heuristics, could be used to create an agent that would best play the game 2048. After researching multiple possible evaluation functions and methods, the ones that were chosen to study, along with how they will be referred to in the following sections, include the following:

- Highest individual tile (highest score), this was a simple greedy heuristic that wanted to achieve the highest single tile each turn. The score returned was the value of the highest tile.
- The maximum number of empty tiles (maximize empty tiles), which iterated over the board and returned the number of empty tiles as the score.
- Weighing empty tiles based on board positioning in favor of a corner (weighted empty tiles). This was made to favor empty tiles being in the bottom right of the board by adding the index of rows and columns to the score when there was an empty tile.
- Smoothness as described in the background research section (smoothness), which would increase the score when a tile was half the value of the tile next to it in any direction. This would mean that it could easily combine with an equal tile to produce a value that could combine into the next one, creating a chaining effect. The score returned was a summation of the tile values that were next to a tile that was double their score.
- Weighing tiles scores by position in favor of higher scores in a corner and along the edges (edge weight). This was done with a simple weight matrix that multiplied the score of the tile by the weight at each position to reach the score. This was made to favor higher scores being in the top left.

These methods encompassed multiple ideas across previous research in order to find the best way to evaluate which move to make. For any given move the functions can take in a board and calculate a score relevant to their individual goals. In addition to the ones just listed, a random move selector was also implemented as a baseline heuristic to help compare the others.

4.3 Other Components

Some other components were considered in the design of the approach to this problem. One of these considerations is that in the case of ties between scores, the algorithm will randomly choose between the moves that are tying. This was done for two main reasons, one being that it keeps the agent from getting stuck in a loop, especially early in a game. The second reason is that this random effect was done with the intention of giving the agent some randomness to counteract the randomness that the board has when adding new tiles. The hope is that by giving the agent some element of randomness as well, it will be more adaptable and may reach higher scores as a result.

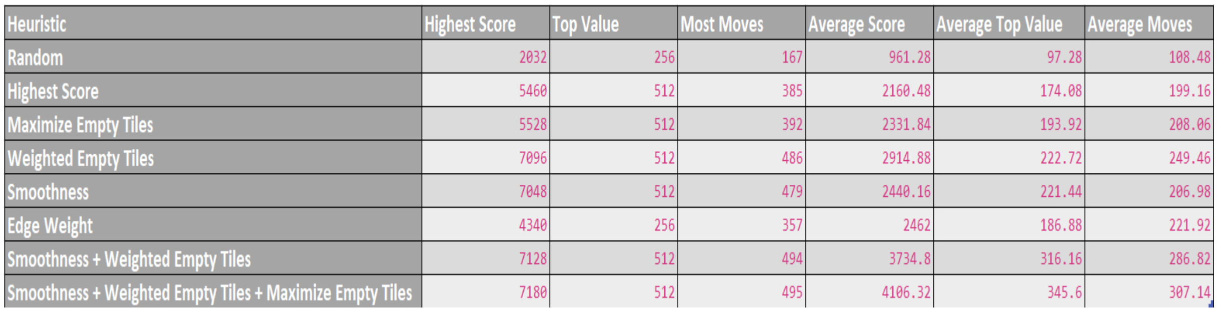
Another thing that was considered on the topic of randomness is that as the agent is recursively working through which move to take, it does not consider that with each move it thinks ahead there will also be a new tile getting added. By leaving this element out of consideration, there are likely cases where it has a "plan" that according to the Minimax algorithm would be the best option, but that option gets taken away thanks to the addition of a new tile at the end of the turn. This is a problem that will be addressed in the Future Work section of this paper.

4.4 Data Collection

After the game code, agent algorithm, and heuristic functions had been written and tested, it was time to gather data to find results and a conclusion. For this portion of the research, an iterator script was made that would create an agent with a specific evaluation mode given, have the agent play through a game, record the game statistic, and then repeat. For each way of evaluation, fifty iterations were completed to ensure the data was representative of how the function performs on average. After the initial rounds of iterations were completed, the statistics were analyzed and two more rounds of iteration were run; one of these rounds included the top two performers combined in their evaluation, and the other included the top three performers combined. This was done to see if combinations of evaluation functions would perform better than their single goal-oriented counterparts. After running all of these games along with the random agent for the same number of iterations, all the data was compiled and the results were compared.

5 Results

The data collected from each agent included the total game score, the highest tile value, and the number of moves made. The total game score is the total value from the summation that occurs when two tiles combine, meaning it will increase with every combination. The highest tile value is the maximum value left on the board when the game is lost and the number of moves made is simply incremented by one every time the agent does an action that changes the board. Results



Heuristic	Highest Score	Top Value	Most Moves	Average Score	Average Top Value	Average Moves
Random	2032	256	167	961.28	97.28	108.48
Highest Score	5460	512	385	2160.48	174.08	199.16
Maximize Empty Tiles	5528	512	392	2331.84	193.92	208.06
Weighted Empty Tiles	7096	512	486	2914.88	222.72	249.46
Smoothness	7048	512	479	2440.16	221.44	206.98
Edge Weight	4340	256	357	2462	186.88	221.92
Smoothness + Weighted Empty Tiles	7128	512	494	3734.8	316.16	286.82
Smoothness + Weighted Empty Tiles + Maximize Empty Tiles	7180	512	495	4106.32	345.6	307.14

Figure 1. Compiled Data For All Heuristics

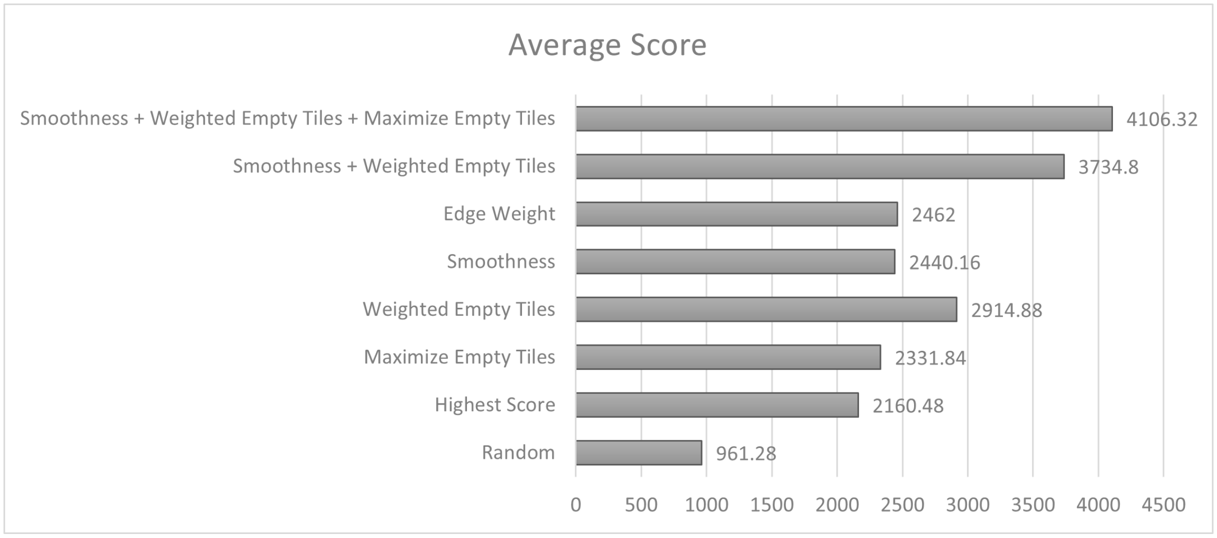


Figure 2. Average Total Game Score

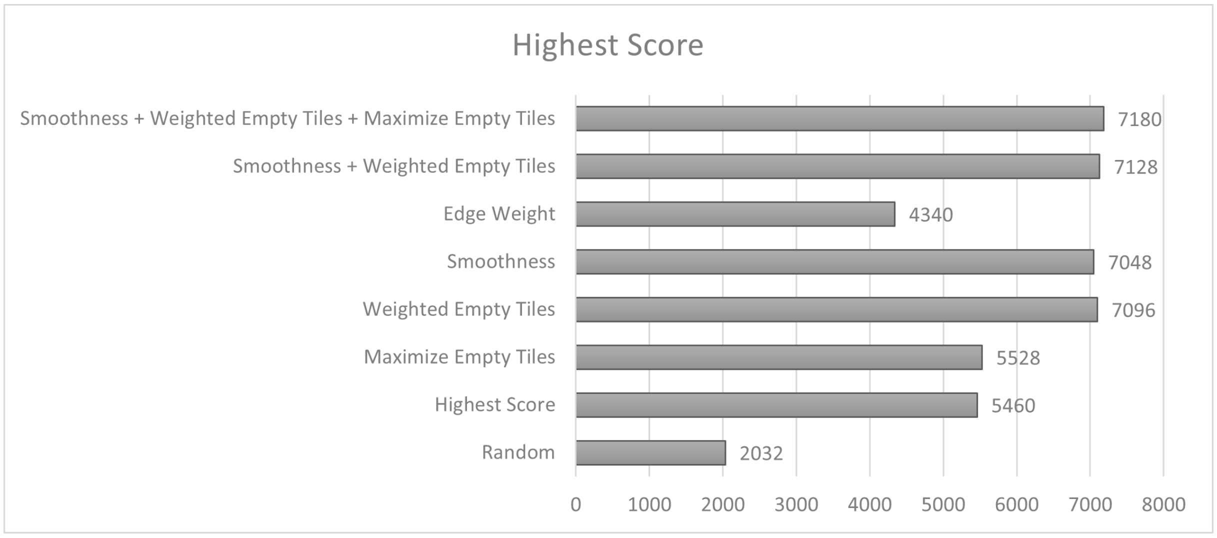


Figure 3. Highest Total Game Score

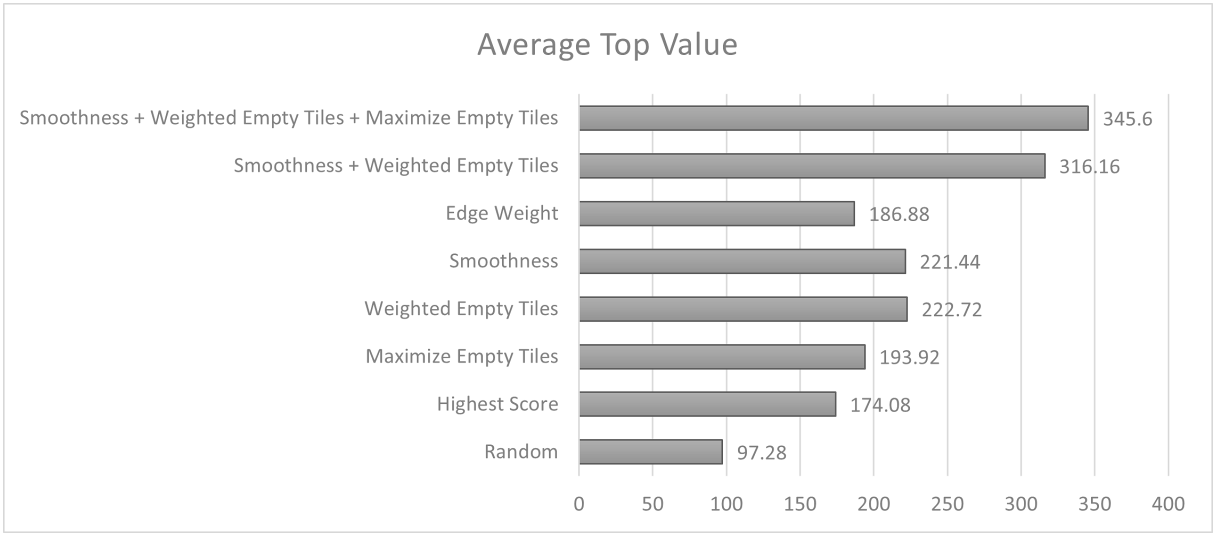


Figure 4. Highest Total Game Score

were then able to be compiled so the average of each of these and the highest of each of these could be recorded for each heuristic function. This allowed the data to give a good idea of both the average performance and the maximum performance of each evaluation method. Figure 1 shows all of the data compiled together.

5.1 Total Game Score

Perhaps the most important indicator of performance was the total game score, as it gives an idea of how many tiles were combined and how far into the game the agent could progress. Figures 2 and 3 compare the average and the maximum of this value across all methods, respectively. A general idea of a baseline total score can be observed by the random agent in Figure 2. It can be seen that it only averages a total of around 961. After noticing this and comparing it to the heuristic function options, it becomes immediately obvious that it performs much worse, with the second worst of all options averaging around 2160, over double the random agent. A similar conclusion can be made by observing the highest total score achieved in Figure 3, with the random agent only getting 2032, whereas the next lowest was the edge weight heuristic at 4340. The best-performing heuristic in these categories was not as clear cut as the worst, with weighted empty tiles being the best for both average and highest total score, but not by as wide of a margin. In the comparison for average total scores, it is pretty close for all of the single-mode heuristic evaluations. There is, however, a large jump again in performance when multiple of these methods get combined. Interestingly, the top scores can be seen to be around 7000 for both the combined and single-mode evaluations.

5.2 Top Value

The last statistic gathered that reveals interesting trends is the average highest value tile at the end of the game. This is shown in Figure 4. Yet again the lowest by a substantial amount is the random agent. The highest among the single-mode heuristics is very close between the weighted empty tiles and smoothness, with weighted empty tiles being slightly better. Again the multi-mode evaluations outperform all the others by quite a large margin in this area.

5.3 Other Statistics

Some other interesting trends can be observed from the gathered statistics in addition to the figures. One of these is that, across all heuristics and their varying success in different categories the highest value tile that any of them ever achieved was 512, which is not even close to winning the game. This is something to note and is likely more due to algorithm choice or implementation than any heuristic performance. This will be discussed more in the Future Work section. Another interesting note is that there is a very strong correlation between moves and total game score. This makes sense as the score is likely to increase with most moves. The correlation did not seem to be as strong between top value and moves, however, though this may be because the highest value can be achieved from a fairly large window of moves before that highest score combines with itself and doubles its value, whereas the total score is a more precise measurement. Because of the correlation between moves and score, moves were not considered as strongly in analyzing results, as it was largely redundant and considered to be less important when judging how well a game went.

6 Analysis

After gathering and organizing all of the statistics above, observations and analysis revealed some interesting findings from the data.

6.1 Baseline

While not all heuristics performed well, they clearly showed that having any strategy is better than having no strategy. This is obvious when one looks at Figures 2, 3, and 4 to see that the random agent is the worst across the board and by a large margin. This suggests that there is some degree of strategy required to play the game well and proves that this problem is worth studying and looking deeper into.

6.2 Consistency

One thing that stands out when analyzing the data is that some functions seem to produce much more consistent results than others do. This is noticeable when the bars in Figures 2 and 3 do not line up, indicating that the average total score and the highest total score that each heuristic reaches are not always the same. Smoothness is a good example of this observation, as it has one of the highest maximum scores, but is more average when comparing mean scores. Meanwhile, both of the multi-mode heuristics had significantly higher averages than the rest, but their maximums were about the same as some of the single-mode ones. This indicates that the smoothness heuristic may be less consistent but has the potential to score really well. On the other side, the multi-mode heuristics seem to score well much more consistently, but their maximum scores are not much higher than those of others.

6.3 Single-Mode Evaluation

Out of the single-mode heuristic functions, the weighted empty tiles seemed to consistently rank among the best, indicating that out of the methods chosen to evaluate the actions of an agent, this may be the best option. This was a somewhat surprising finding following the background research that indicated that the smoothness evaluator would likely be the best. However, while smoothness was not the best, it was always among the best and had the potential to score very well. Because of this, smoothness and weighted empty tiles were considered to be the two best single-mode heuristics. Finding the third best was less clear, as edge weight and maximize empty tiles were fairly similar in their performance, but maximize empty tiles were considered to be better due to it slightly edging out edge weight in Figure 4. The highest score heuristic was the worst overall, which is not surprising as this was a purely greedy approach with very little strategy when compared to the others, with the exception being the random agent.

6.4 Multi-Mode Evaluation

After determining which ones of the single-mode evaluations were the best, agents were made to play with multiple of them. The hopes were that the agent would combine the strengths of multiple functions and play better as a result. As shown in Figures 2, 3, and 4, they very clearly performed better in nearly every regard. This was promising and proved the concept that the game is complex enough to require multiple considerations to win. It is worth noting that the agent with the best three heuristics outperformed the agent with just the top two, however, the speed at which the agents were able to play slowed drastically with the addition of each new heuristic that it would consider. While the combination of heuristics still was not enough to get the agents to win the game, it made them play more consistently and got their average total scores to be much higher than others, which indicates that they were closer to merging higher value tiles into potentially winning range. The consistency of these agents is worth noting because if their performance overall were to be improved, they likely would be able to reliably score much higher than the best of human players. While right now their scores leave much to be desired, they are on track to play very well at a stable level when the agent considers multiple or many of the heuristics proposed in this paper.

7 Future Work

Multiple aspects of this study could be expanded on in further projects or papers. These are things that could have been implemented given more time, though due to constraints they were not able to be investigated further.

7.1 Expectimax

As noted in the Approach method of this paper, the Minimax algorithm was used as the standard algorithm on which the heuristic functions would be tested. This algorithm was used in multiple other studies as described in the Background section, however, it may have been beneficial to cross-validate findings with another algorithm. The Expectimax algorithm is similar to the Minimax one, but it is better at taking chance into perspective and the board would not have been treated as an antagonist. There is a chance that Expectimax would outperform Minimax in this type of problem, but that has not been verified in this study.

7.2 Probabilities

One potentially major reason why the agent was never able to win a game is that it was not able to take the chance of adding a new tile to a space into account when it was considering multiple moves into the future. This ties into the Expectimax idea, but is a slightly separate problem where the agent would have needed a way to take these chances into account to fully consider what all the possible outcomes could be. Doing this, however, would be very computationally expensive and would slow down the speed of the program immensely as the agent would need to check many more possibilities.

7.3 Heuristic Optimization

The heuristics used in this study were well-researched and planned with the game in mind, however, there is always the opportunity to make slight adjustments that could impact the speed and performance of the agent, possibly to the point where better results are achieved. This is something that would simply take more time and testing and should be something to consider in future projects. Another factor to consider in this aspect is that the scores of the heuristic functions were not normalized to one another. This did not seem to play a major role in their success but may be worth looking into when continuing this research in the future.

7.4 More Combinations

The fact that the combination of three heuristic functions performed better than two heuristic functions which performed better than any of the single-mode heuristics indicates a strong correlation that it may be worth considering many factors in each decision. This was not done in this research due to how computationally expensive it becomes to run more than three functions in a single move for the agent. Each heuristic function must iterate over the whole board to get its score and this is done recursively over all actions for each recursive step, so the number of times the board must be iterated over grows rapidly with each additional heuristic function to consider. This could be combated by manually coding the combinations of heuristic functions together, which would cut back on the need to iterate over the board so many times. This however was not the purpose of this research, which wanted to keep the flexibility of being able to assign multiple modes at any time to find the best preliminary heuristic functions to be built off of in the future. Given more time and a wider scope, this would likely significantly improve the success of the agent.

8 Conclusion

The goal of this study was to find which heuristic evaluation method would be most effective for an agent to consider when playing the game 2048 and attempting to achieve the highest score. After setting up a testing environment that included coding the game, the Minimax algorithm, all of the heuristic functions, and a method to iterate through many games, this study found that the single most effective method to evaluate the heuristics is a function based on weighted empty tiles, which creates an incentive for the agent to keep the tiles pushed towards a corner, thus keeping the empty tiles in the opposite corner free. While this was the single most effective method, it was also concluded that combining the efforts of multiple heuristics is beneficial to the agent and can greatly increase performance. The highest-performing method overall was a combination of the smoothness, weighted empty tile, and maximize empty tile heuristic functions from this study. It outperformed every other evaluation function in nearly every metric, though it also greatly increased the computational cost of finding the optimal move, and hence slowed down the program. Also, despite outperforming other metrics and doing so consistently, it still was not able to win a single game over fifty iterations, which indicates that there are still improvements to be made. These findings are similar to what was expected by the hypothesis but in a somewhat surprising way. The hypothesis was in support of the top two heuristic functions being edge weight and smoothness rather than weighted empty tiles and smoothness. The edge weight rewarded high-value tiles being placed in a corner of the board, whereas weighted empty tiles rewarded empty spaces in a corner. The end goal of these two is fairly similar, but the fact that weighted empty tiles outperformed edge weight by a fair margin suggests that it may be more important to keep more empty spaces than to focus on keeping high values in the corner. This does make sense too, because, as the game progresses, there will always be higher values being stored, so keeping the board more empty and keeping that emptiness focused in one corner will naturally force larger values to the opposite corner.

References

[1] Iris Kohler, Theresa Migler, and Foaad Khosmood. 2019. Composition of Basic Heuristics for the Game 2048 *(FDG '19)*. Association for Computing Machinery, New York, NY, USA, Article 52, 5 pages. https://doi.org/10.1145/3337722.3341838

[2] Kevin Lu. 2014. Game Theory of 2048. (2014).

[3] Todd W Neller. 2015. Pedagogical possibilities for the 2048 puzzle game. *Journal of Computing Sciences in Colleges* 30, 3 (2015).

[4] Mikloš Pot, Milovan Milivojević, and Srđan Obradović. [n. d.]. USING ARTIFICIAL INTELLIGENCE TO CREATE A SOFTWARE AGENT FOR SOLVING THE 2048 GAME. ([n. d.]).

[5] Philip Rodgers and John Levine. 2014. An investigation into 2048 AI strategies. In *2014 IEEE Conference on Computational Intelligence and Games*. 1–2. https://doi.org/10.1109/CIG.2014.6932920

[6] Jun Tao, Gui Wu, Zhentong Yi, and Peng Zeng. 2020. Optimization and Improvement for the Game 2048 Based on the MCTS Algorithm. In *2020 Chinese Control And Decision Conference (CCDC)*. 235–239. https://doi.org/10.1109/CCDC49329.2020.9164764

[7] Ahmad Zaky. 2022. Minimax and Expectimax Algorithm to Solve 2048. (May 2022). https://doi.org/10.31219/osf.io/xfdsr

[8] Yulin Zhou. 2019. From AlphaGo Zero to 2048. https://api.semanticscholar.org/CorpusID:198918500