Beating 2048 with AI using Heuristics and Random Roll-outs

A Project report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR'S OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

Vicky Kumar Gupta

MIS No: 112015161

Semester: IV

Computer Science and Engineering

Indian Institute of Information Technology Pune,

Near Bopdev Ghat, Kondhwa Annexe, Yewalewadi, Pune, Maharashtra

411048

APRIL 2022

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Beating 2048 with AI using Heuristic search and Random Roll-Outs"** submitted by **Vicky Kumar Gupta** bearing the **MIS No: 112015161**, in completion of his project work under the guidance of **Dr. Tanmoy Hazra** is accepted for the project report submission in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering , Indian Institute of Information Technology, Pune, during the academic year 2020-21.

Dr. Tanmoy Hazra
Project Guide
Assistant Professor
Department of CSE
IIIT Pune

Dr. Tanmoy Hazra
Head of the Department, CSE
Assistant Professor
Department of CSE
IIIT Pune

Project viva-voce held on 28/04/2022

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

This project would not have been possible without the help and cooperation of many. I would like to thank the people who helped me directly and indirectly in the completion of this project work.
First and foremost, I would like to express my gratitude to our beloved director, **Dr. Anupam Shukla**, for providing his kind support in various aspects.

I would like to express my gratitude to my project guides **Dr. Tanmoy Hazra**, Assistant Professor, Department of CSE, for providing excellent guidance, encouragement, inspiration, constant and timely support throughout this B.Tech project.
I would again like to express my gratitude to the head of department **Dr. Tonmoy Hazra**, Head of Department CSE, for providing his kind support in various aspects.
I would also like to thank all the faculty members in the Dept. of CSE and my classmates for their steadfast and strong support and engagement with this project.

ABSTRACT

2048 is a single-player sliding block puzzle game designed by Italian web developer Gabriele Cirulli. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048.

One of the possible ways to solve the game of 2048 is to exploit the MC algorithm. Its biggest advantage is that it is a general-purpose solver, which means that it can yield output without any game specific input. The idea of using a large number of random simulations of an experiment to gain insights into the experiment's end results. Random simulations of an experiment are frequently referred to as Monte Carlo simulations.

While MCTS algorithm proved to be very successful in solving the game of 2048, we don't have to stop here. MCTS can be implemented in other games and applications as well and I encourage myself to try different aspects of this algorithm and implement them to un making some impactful AI.

TABLE OF CONTENT

7. References

Chapter 1

Introduction

In this project, we implemented an agent which uses the Monte Carlo Tree Search algorithm to play the game 2048. Before hand, lets catch some details about 2048 game, and MCTS(Monte Carlo Tree Search)

1.a What is 2048 game?

2048 is a single-player game played on a 4x4 grid. In each turn, two events occur: 1) the user must decide how to slide the tiles, and 2) if the grid changes after the tiles slide, a random tile with the number 2 or 4 spawns in a free space on the grid. When the player presses an arrow button, the tiles slide as far as possible to the corresponding side of the board until it hits either the side of the board or another tile. If the tile hits a tile of the same value, they combine into one tile with double the original value. This combined tile cannot combine with other tiles in the same move. A merge score is kept track based on the value of a combined tile and will increase with every combined tile that is made throughout the duration of the game.

Overall, the objectives of the game are to maximize the tile with the largest number on it (or achieve 2048). The game ends when all free spaces are used up and no two tiles can be merged. In our project, our goal is to develop a bot which is able to play 2048 well through reinforcement learning.

Link for the reference: PLAY 2048 HERE!

Randomness of the game 2048:

The source of randomness in this game is the random placement of a new tile on a available space on the board after each turn and whether the tile value is a 2 or a 4. The bot would need to take into account this uncertainty and play with a strategy to maximize the largest tile achieved, the sum of the tiles on the final board, and the merge score.

1.b What is MCTS?

Monte Carlo method was coined by Stanislaw Ulam for the first time after applying statistical approach “The Monte Carlo method”. The concept is simple.
Using **randomness** to solve problems that might be deterministic in principle. For example, in mathematics, it is used for estimating the integral when we cannot directly calculate it. We use Monte Carlo method to **estimate the quality of states stochastically** based on simulations when we cannot process through all the states. Each simulation is a **self-play** that traverses the game tree from current state until a leaf state (end of game) is reached.

Monte Carlo Tree Search (MCTS), which combines Monte Carlo methods with tree search, is a method for finding **optimal decisions** in a given domain by **taking random samples in the decision space** and building a search tree according to the results.
It is a **probabilistic and heuristic driven search algorithm** that combines the classic tree search implementations alongside machine learning principles of reinforcement learning.

In tree search, there’s always the possibility that the current best action is actually not the **most optimal action**. In such cases, MCTS algorithm becomes useful as it continues to evaluate other alternatives periodically during the learning phase by executing them, instead of the current perceived optimal strategy. This is known as the ” exploration-exploitation trade-off “. It exploits the actions and strategies that is found to be the best till now but also must continue to explore the local space of alternative decisions and find out if they could replace the current best.
Exploration helps in exploring and discovering the unexplored parts of the tree, which could result in finding a more optimal path. In other words, we can say that exploration expands the tree’s breadth more than its depth.
Exploitation sticks to a single path that has the greatest estimated value. This is a greedy approach and this will extend the tree’s depth more than its breadth. In simple words.

Chapter 2

Objectives and Motivation

2.a Improved heuristics to play and learn by themselves.

Several computer players have been developed for 2048 over the past years. Early players employed a depth-limited tree search algorithms like minimax or expected max alongside an evaluation function. Later, more advanced approaches came around like that one employed by Szubert and Jaskowski who utilized N-tuple networks as the evaluation functions and apply a reinforcement learning method to adjust the weights of the N-tuple networks.

Our work for this paper involves implementing a Monte Carlo Tree Search. MCTS is an online, heuristic search algorithm that finds the optimal decision by running simulations on the given state and simply choosing the action that maximize the estimate of the action-value Q (s, a).

This method of reinforcement learning has achieved relative success in game AIs. Most notably, MCTS has been used in creating Google's AlphaGo and AlphaGo Zero. In a similar approach to AlphaGo Zero, where its neural networks learn by generating its own training data through simulations of playing Go, we will play simulations of 2048 to determine the most optimal direction move for our bot to play.

2.b Solving complex problems without extensive algorithm, just rolling-out random simulations.

Monte Carlo Tree Search is a heuristic algorithm. MCTS can operate effectively without any knowledge in the particular domain, apart from the rules and end conditions, and can can find its own moves and learn from them by playing random playouts.

The MCTS can be saved in any intermediate state and that state can be used in future use cases whenever required.

MCTS supports asymmetric expansion of the search tree based on the circumstances in which it is operating.

Chapter 3

Literature Review

1. MCTS combined with deep reinforcement learning, has become the backbone of **AlphaGO** developed by the **Google DeepMind** to play **Go** game at superhuman level, described in the Article from Silver et al. (2016).
   Link for the reference: https://www.deepmind.com/research/highlighted-research/alphago

2. MCTS is directly applicable to problems which can be modelled by a **Markov Decision Process (MDP)** (Lizotte and Laber, 2016), which is a type of discrete-time stochastic control process.

3. MCTS has been tried in the most of **combinatorial games** and even some real-time video games (Fa - rooq et al., 2016; Kim and Kim, 2017).

4. J. Yang, Y. Gao, S. He, X. Liu, Y. Fu, Y. Chen, and D. Ji, "**To Create Intelligent Adaptive Game Opponent by Using Monte-Carlo for Tree Search**," in Proc. 5th Int. Conf. Natural Comput., Tianjian, China, 2009.

5. **Cameron Browne (IEEE)** received a Ph.D. in Computer Science from the Queensland University of Technology (QUT), Australia, in 2008. He is currently a Research Fellow at Imperial College, London, working on the EPSRC project UCT for Games and Beyond, **investigating MCTS methods for procedural content generation** in creative domains such as game design, linguistics, and generative art and music.

Chapter 4

Methodology

4.1 Working of MCTS

In MCTS, nodes are the building blocks of the search tree. These nodes are formed based on the outcome of a number of simulations. The process of Monte Carlo Tree Search can be broken down into four distinct steps, viz., selection, expansion, simulation and backpropagation. Each of these steps is explained in details below:

1. **Simulation**: In this process, the MCTS algorithm traverses the current tree from the root node using a specific strategy. The strategy uses an evaluation function to optimally select nodes with the highest estimated value. MCTS uses the Upper Confidence Bound (UCB) formula applied to trees as the strategy in the selection process to traverse the tree. It balances the exploration-exploitation trade-off. During tree traversal, a node is selected based on some parameters that return the maximum value. The parameters are characterized by the formula that is typically used for this purpose is given below.

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

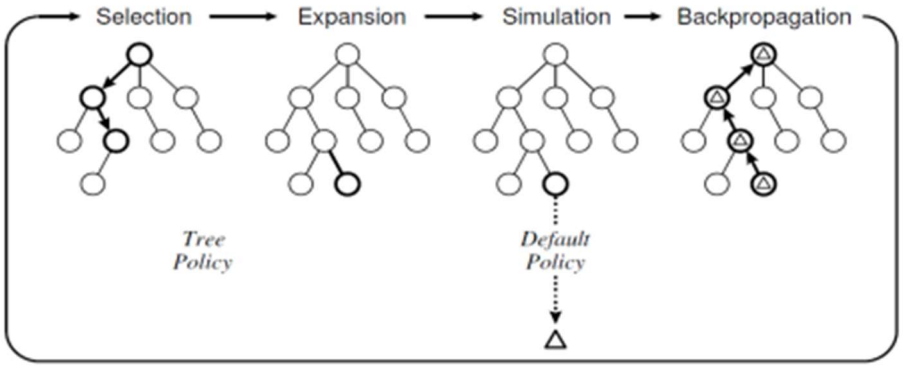
A child node j is selected to maximize: UCT,

where n is the number of times the current (parent) node has been visited, nj the number of times child j has been visited and Cp > 0 is a constant. If more than one child node has the same maximal value, the tie is usually broken randomly.

Trading off between Exploration and Exploitation:

The values of Xi,t and thus of Xj are understood to be within [0, 1] (this holds true for both the UCB1 and the UCT proofs). It is generally understood that nj = 0 yields a UCT value of infinity, so that previously unvisited children are assigned the largest possible value, to ensure that all children of a node are considered at least once before any child is expanded further. This results in a powerful form of iterated local search. There is an essential balance between the first (exploitation) and second (exploration) terms of the UCB equation.

2. **Expansion**: if the selected leaf node is expandable (i.e. it does not represent a terminal state) then one (or more) node is added to expand the tree.
3. **Simulation**: the nodes selection is completed following a default policy from newly added node to a terminal node, obtaining thus an outcome.
4. **Backpropagation**: the outcome is backpropagated through the tree, i.e., in each node traversed the average outcome of the simulations is computed.



4.2 Pseudo Code of MCTS:

```
# main function for the Monte Carlo Tree Search
def monte_carlo_tree_search(root):

    while resources_left(time, computational power):
        leaf = traverse(root)
        simulation_result = rollout(leaf)
        backpropagate(leaf, simulation_result)

    return best_child(root)

# function for node traversal
def traverse(node):
    while fully_expanded(node):
        node = best_uct(node)

    # in case no children are present / node is terminal
    return pick_unvisited(node.children) or node

# function for the result of the simulation
def rollout(node):
    while non_terminal(node):
        node = rollout_policy(node)
    return result(node)

# function for randomly selecting a child node
def rollout_policy(node):
    return pick_random(node.children)

# function for backpropagation
def backpropagate(node, result):
    if is_root(node) return
    node.stats = update_stats(node, result)
    backpropagate(node.parent)

# function for selecting the best child
# node with highest number of visits
def best_child(node):
    pick child with highest number of visits
```

4.3 Evaluating Methods to Implement MCTS to solve 2048.

When evaluating an agent, we run 100 trials (games). We consider three different factors to mimic what people may want to achieve when playing 2048. Our first factor is the percentage of games which achieved a maximum tile number that is equal to or more than 2048: given that the name of the game 2048 is 2048, many people consider achieving a 2048 tile as winning the game. Our second factor is the sum of the squares on the final board, as players may evaluate their final board as a whole and this incorporates all of the information in the final state of the board. Our third factor is the average merge score for the 100 games, where the merge score is the sum of the values for all squares that were merged together throughout the game (we use a discount factor of 1). An agent which achieves a higher 2048 square percentage, a higher average sum of final tiles, and a higher average merge score is considered better performing.

MCTS Value Functions:

When playing the game, we considered three different types of value functions used to determine which action to take at each step in the game.

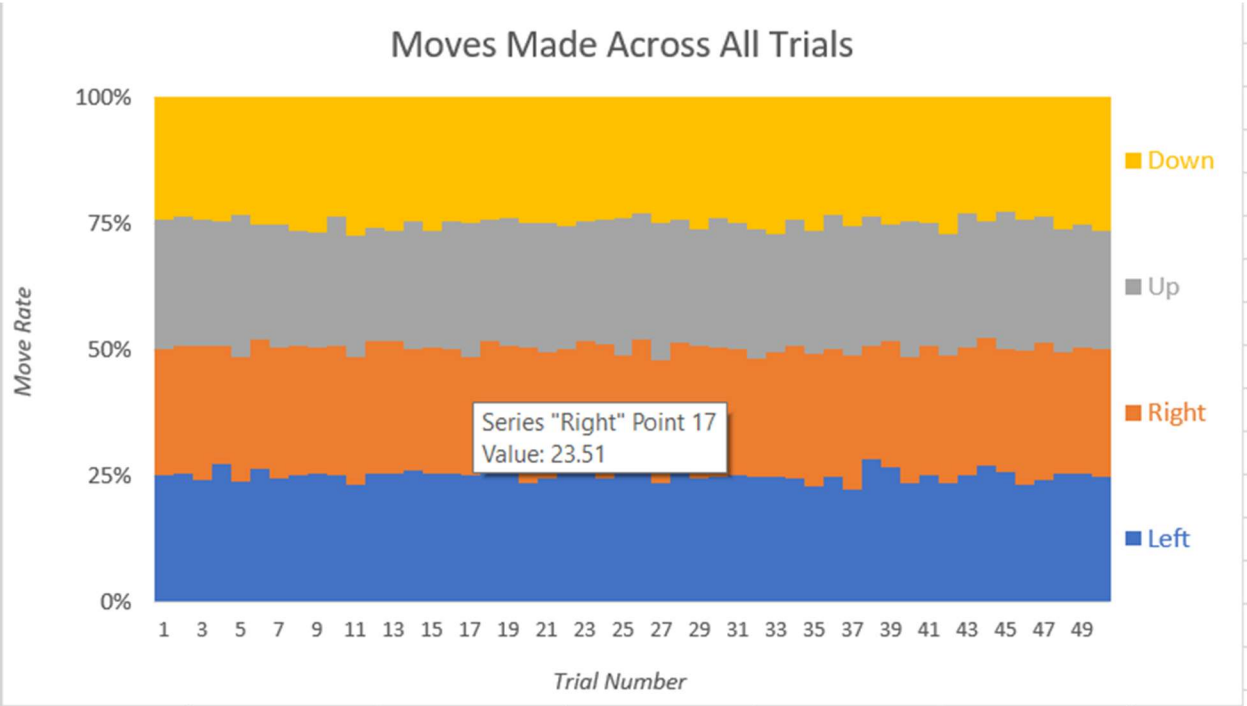
1. The first value function we used was the merge score at the end of the game, which we also discuss when evaluating the performance of an agent.
2. The second value function we used was the largest board sum at the end of the game, as this metric captures information about every number on the board at the end of the game. As many 2048 players are not aware of the merge score the 2048 provides, we felt that this more closely represents the judgement of a human 2048 player at the end of the game.
3. The third value function we used was the largest tile at the end of the game, with ties being broken with the sum of the tiles on the board at the end of the game. This uses the idea that one metric some 2048 players use is the highest number (aligning with one of our evaluation metrics, the 2048 square percentage), and gives a way to break ties

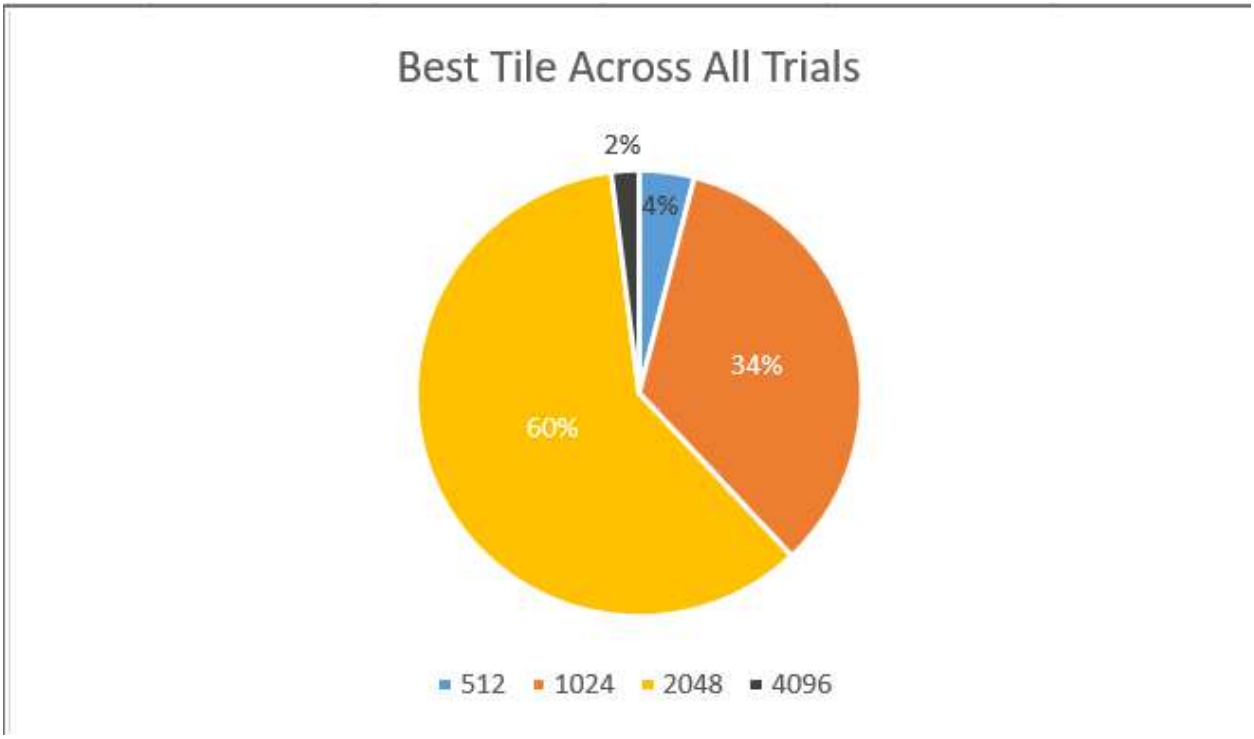
Random Policy:

When evaluating our models, we first ran experiments on the random policy. We simulated 100 full games to find the average largest tile, sum of the final board, and merge score. The random policy never reached the 2048 tile, had a really low average sum for the final board of 262.8, and

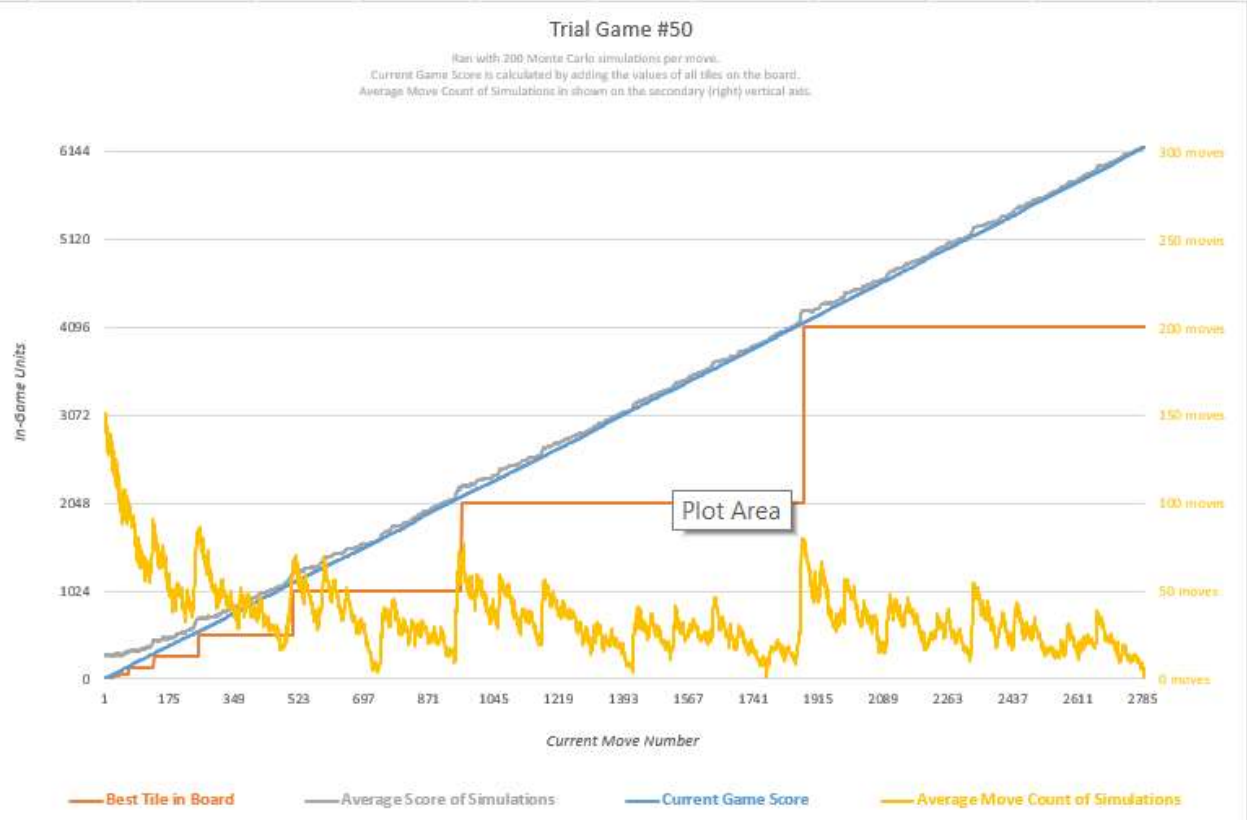
had a really low average merge sum of only 942.72. Table 2 shows our results for the random policy.

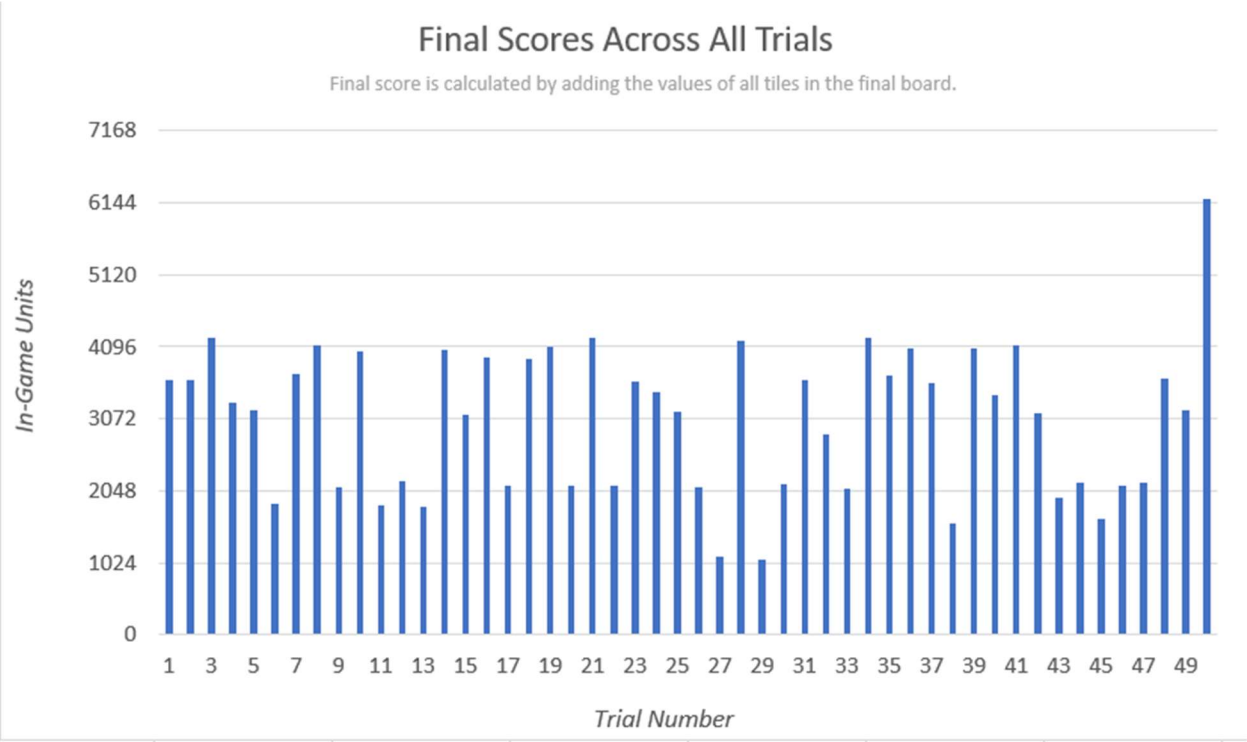
Result of the Simulations:





Best Move	MS To Calcu	Best Tile in	Average Sco	Average Mo	Current Game Score
up	47	2	274.64	145.76	6
down	36	2	267.48	140.86	8
up	50	4	274	144.9	10
right	48	4	290.92	151.34	12
left	62	8	278.52	143.82	14
left	54	8	274.16	140.52	16
down	55	8	283.92	143.78	18
up	45	8	275.44	138.52	20
right	57	8	261.48	128.98	22
up	54	8	275.36	137.44	24
left	53	16	281.36	139.38	26
up	48	16	260.12	126.84	28
down	45	16	285.08	139.4	30
right	25	16	284.2	137.54	32
up	48	16	283.44	135.02	34
down	48	16	270.92	128.02	36
right	45	16	271.4	127.82	38
right	47	16	292.2	139.52	40
up	50	16	288.04	136.12	42
left	66	16	292.68	134.72	44
left	72	16	280.16	129.64	46
right	44	16	283.84	130.22	48
right	38	16	266.2	118.68	50
right	46	16	288.28	130.18	52
down	42	16	269	118.46	54
down	43	16	277.68	121.68	56
up	54	16	268.84	115.98	58
left	49	32	278.36	121.18	60
down	46	32	285.84	123.28	62
up	34	32	273.32	114.96	64
up	46	32	269.04	110.92	66
right	51	32	293.12	124.94	68
down	44	32	291.04	122.14	70





You can find the source code on my Github Account: [AI_LEARNS_2048]

Chapter 4

Conclusions

MCTS has become the pre-eminent approach for many challenging games, and its application to a broader range of domains has also been demonstrated. In this paper we present by far the most comprehensive survey of MCTS methods to date, describing the basics of the algorithm, major variations and enhancements, and a representative set of problems to which it has been applied. We identify promising avenues for future research and cite almost 250 articles, the majority published within the last five years, at a rate of almost one paper per week. Over the next five to ten years, MCTS is likely to become more widely used for all kinds of challenging AI problems. We expect it to be extensively hybridised with other search and optimisation algorithms and become a tool of choice for many researchers. In addition to providing more robust and scalable algorithms, this will provide further insights into the nature of search and optimisation in difficult domains, and into how intelligent behaviour can arise from simple statistical processes.

MCTS shows great promise in non-game applications, in areas such as procedural content generation (indeed a recent issue of this journal was devoted to this topic) as well as planning, scheduling, optimisation and a range of other decision domains. For example, the introduction of an adversarial opponent provides an ability to work with "worst-case" scenarios, which may open up a new range of problems which can be solved using MCTS in safety critical and security applications, and in applications where simulation rather than optimisation is the most effective decision support tool.

Chapter 5

Future Work

Combining the precision of tree search with the generality of random sampling in MCTS has provided stronger decision-making in a wide range of games. However, there are clear challenges for domains where the branching factor and depth of the graph to be searched makes naive application of MCTS, or indeed any other search algorithm, infeasible. This is particularly the case for video game and real-time control applications, where a systematic way to incorporate knowledge is required in order to restrict the subtree to be searched. Another issue arises when simulations are very CPUintensive and MCTS must learn from relatively few samples. Work on Bridge and Scrabble shows the potential of very shallow searches in this case, but it remains an open question as to whether MCTS is the best way to direct simulations when relatively few can be carried out. Although basic implementations of MCTS provide effective play for some domains, results can be weak if the basic algorithm is not enhanced. This survey presents the wide range of enhancements considered in the short time to date. There is currently no better way than a manual, empirical study of the effect of enhancements to obtain acceptable performance in a particular domain. A primary weakness of MCTS, shared by most search heuristics, is that the dynamics of search are not yet fully understood, and the impact of decisions concerning parameter settings and enhancements to basic algorithms are hard to predict. Work to date shows promise, with basic MCTS algorithms proving tractable to “in the limit” analysis. The simplicity of the approach, and effectiveness of the tools of probability theory in analysis of MCTS, show promise that in the future we might have a better theoretical understanding of the performance of MCTS, given a realistic number of iterations. A problem for any fast-growing research community is the need to unify definitions, methods and terminology across a wide research field. We hope that this paper may go some way towards such unification.

References:

1. Leila Amgoud and Mathieu Serrurier. An argumentation framework for concept learning. In 11th Intl. Workshop on Non-Monotonic Reasoning (NMR06), 2006.

2. Peter Auer, Nicol`o Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. Machine Learning, 47(2–3):235–256, 2002.

3. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and et al. Mastering the game of go without human knowledge. Nature, 550(7676):354–359, 2017.

4. M. Szubert and W. Jaskowski. Temporal difference learning ´ of n-tuple networks for the game 2048. pages 1–8, 2014.

5. https://web.stanford.edu/class/aa228/reports/2020/final41.pdf

6. https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/#:~:text=The%20process%20of%20Monte%20Carlo,%2C%20expansion%2C%20simulation%20and%20backpropagation.

7. **Masoud Masoumi Moghadam,**
https://towardsdatascience.com/monte-carlo-tree-search-implementing-reinforcement-learning-in-real-time-game-player-25b6f6ac3b43

8. http://www.incompleteideas.net/609%20dropbox/other%20readings%20and%20resources/MCTS-survey.pdf