Comparison of Different AI Search Algorithms on Sokoban, Tic Tac Toe and Snake Game (revised December 2021)

Saumya Patel, Utkarsh Sharma, and Prakhar Saxena, *Member, JK Lakshmipat University*

Abstract— The gaming business has grown at a breakneck pace, with a diverse range of companies, values, and scope. Each character's every nuance, as well as their surroundings, must be coded. Artificial intelligence has been utilized in software games to emulate human players, allowing game developers to design personalized experiences and outcomes for each player. Laptop chess players are a good example, with modern chess algorithms being able to beat the best human players. Although there are AI algorithms that can be employed in games, the need for best solution is increasing. To determine a strategy in games, search algorithms are used. The algorithms comb through the possibilities and select the best solution. Speed, precision, and intricacy are just a few of the key things to consider. We require a comparative analysis of several algorithms are being studied in order to determine which is the most efficient and perfect. In this paper, the performance of a set of state-space search methods in solving Sokoban and Snake Game puzzles is compared. We look at how well Breadth-First, Depth-First, and Uniform Cost Search, as well as in-formed search algorithms like A*, perform. The Min-max method is used to replicate tic-tac-toe. In this paper, a simulation approach is described for predicting whether a game would win, or draw based on a player's first move. We provide five different algorithms or approaches for a computer to automatically play Snake and Sokoban, including artificial intelligence-related searching algorithms, Best First Search, and A Search. These strategies may be the foundational technique, yet their results are vastly different. We ran tests to compare and explain their disparities in performance. Furthermore, we show how various strategies might become caught in a dead end and forecast further advances as future work.

Index Terms—A* searching, Depth-first Searching, Breadth-first searching, Best-first searching, Min-Max Algorithm, Uniform Cost Search

1 INTRODUCTION

Artificial Intelligence is turning into more and more vital in a ramification of fields. Games offer a good ideal platform for testing and exploring with these algorithms in a sandbox with far less constraints than in real life. Artificial Intelligence (AI) is described as a form which can learn, reason, and resolve problems in the same way that a human can. We opted to put multiple AI algorithms to the trial by playing Snake, Sokoban, and Tic Tac Toe with them. This involves creating a game as well as five different AI searching algorithms.

The purpose of this research is to create a artificial intelligence agent to solve the classic Japanese board game, Sokoban with the use of a range of algorithms and heuristics, and then assess the effectiveness using evaluation metrics. The purpose of the game is for a single agent to push boxes to a series of specified storage sites while existing on a grid of tiles. The world is always fully visible to the player, and each puzzle has only a limited number of alternative states (hence the states are discrete). At any time during the game, the player has the option of moving up, down, left, or right to a neighboring tile. The player can always move into the adjacent tile if it is unoccupied or a storage spot. If the neighboring tile is a box and the tile behind it is empty or a storage place, the player can push the box into the tile behind it. The player may not move into an adjacent tile if it is a wall or a box that cannot be pushed. As a result, the preceding criteria show that these acts are deterministic. A solution to the problem consists of a series of moves that result in all the boxes being placed in storage places after the last move is completed. The Sokoban game is a difficult computational challenge. It was shown to be NP-hard at first, and then PSPACE-complete later. While the rules are straight- forward, even minor levels can necessitate a significant amount of work.

Tic-tac-toe is a one-of-a-kind game. Two vertical & two horizontal lines are utilized to construct a 3-by-3 matrix before the game starts. Tic-tac-toe, a 2 people, game in which players play alternatively counting the empty places in a three-by-three grid. Either row must contain three X or O markers in order to win. In the game of tic-tac-toe, a player seeks to avoid two scenarios: Decrease your chances of winning. Reduce the chances of a block failing. To lower the chances of defeating the opponent, increase profitability. The 9 spaces can be filled with either ('X') and ('O') being most popular game. (AI) heuristic technique, the alpha-beta pruning algorithm, and the min-max algorithm in this study.

- *Saumya Patel is with the Department of Engineering and Technology, JK Lakshmipat University, Jaipur, E-mail: saumyapatel@jklu.edu.in*
- *Prakhar Saxena is with the Department of Engineering and Technology, JK Lakshmipat University, Jaipur, E-mail: prakharsaxena@jklu.edu.in*
- *Utkarsh Sharma is with the Department of Engineering and Technology, JK Lakshmipat University, Jaipur, E-mail: utkarshsharma19@jklu.edu.in*

A human player is commonly used to play Snake. The human player controls the snake by using only the left and right arrow keys to turn it in the direction it is travelling. The snake goes forward automatically, lengthening and speeding up every time it hits an objective, which in this report is referred to as apples. This makes achieving a high score harder. The game's goal is to get the snake to devour as many apples as possible without causing it to run out of the board or bite itself. When this occurs, the game is over, and the final score is re-calculated. Even if conventional Snake is played by humans, it might be a useful place to test AI algorithms, particularly searching algorithms, on a computer.

Artificial intelligence is crucial in the game-playing area. Games don't necessitate a lot of knowledge; all we need to know are the rules, allowed moves, and the criteria for winning or losing. To find out a strategy in games, search algorithms are used. The algorithms comb through the possibilities and select the best solution. Speed, precision, and intricacy are just a few of the key things to consider. These algorithms evaluate all available options at the time and forecast future moves based on them.

These algorithms' goal is to figure out the best combination of moves to get them to the intended result. Every game has its own set of rules that must be followed to win. Based on those conditions, these algorithms look for a sequence of moves. Things get a little trickier when it comes to multi-player games. Let's look at a game with two players. For every move made by the player, the opposing player will attempt to prevent the player from achieving the goal. In this work, the implementation of some search algorithms is done to determine their performance in a game, and the performance of these algorithms is compared be- tween games. Some of the chosen searching algorithms include breadth-first search, depth-first search, A* search, and best-first search. These algorithms were chosen because they are the most regularly used searching algorithms and are path-finding algorithms.

2 LITERATURE REVIEW

According to [1,] numerous heuristics are employed to solve the game, such PI-corral pruning (which minimises the number of locations growing), hashing (which prevents looking at the same function multiple times), deadlock tables (which estimates the positions of the deadlocks in the sport), and so on. As noted in [3,] several authors attempted to translate this challenge into a device mastery problem, in which the device is utilised to develop new ranges mostly depending on the complexity requested by the user. The BFS-A* and DFS-A* solvers are used in paper [2] and are evaluated against elementary and unsophisticated approaches. We may deduce from this literature review that the set of rules for fixing Sokoban has been extensively explored, and a variety of implementations have been developed for quickly resolving this issue with greater performance.

Because tic-tac-toe is a little 3x3 game, the constructed nation area tree may be little as well. We discovered that using the min-max algorithm only estimates one step ahead of time and is a perfect situation for a participant to evaluate all the alternatives to ensure success. We wanted to see how many no-loss responses there could be. Some reseachers [5] employed Genetic Algorithms to keep and unravel what makes this kind of procedure not crash, as well as to design new methods to analyze an answer utilising MATLAB matrix evaluation. Carl, a researcher [6] uses AB pruning for solve his Tic-Tac-Toe conundrum. The goal of the study was to lessen the amount of sport tree that had been generated. Another pair of researchers[7] examine the performance of simultaneous minimax algorithms for exploration in a reconstruction tree in their research Tic-Tac-Toe is a recreation they utilise in their studies to gauge performance. The proposed parallel computational version takes advantage of tree partitioning at width for every stage of the game tree and is completely primarily based on a aggregate of parallel algorithmic paradigms. Some reseachers [8] used the sport precept, minmax, and ideal method of their version. All viable triumphing techniques are examined to make certain achievement no longer handiest via obstructing the alternative participant's fulfilment however additionally through guaranteeing that obstructing the opponent's fulfilment does now not reveal additional weaknesses. They drew on Al-Khateeb's debate, which cautioned that artificial neural networks must now not be employed as feature evaluators for the introduction of tic-tac-toe endeavor play procedures [9].

For snake game, we researched about similar games and the algorithms implemented in them. One successful implementation of AI was found in Ms. Pacman. Ms. Pacman is a sport that has three or more ghosts, the ones are at the closing role of the door whilst the game begins the Ms. Pacman desires to acquire the factors as the time goes on, the ghosts choose one of a kind paths to kill Ms. Pacman. Gallager and Ryan [10] developed a rule-based controller for the Ms. Pacman sport. The modern-day state of Ms. Pacman is decided by the controller based on the gap between the ghosts. This related paintings facilitates in the development of the snake recreation by applying some of the guidelines which makes the algorithms extra ordinary, as an example, if the snakes head collides with the box then the AI should prevent by way of using the set of the guidelines, the snake seems to be implemented in this type of manner that it moves away if any danger takes place within the games. So, this work facilitates to apply some of the restrictions and policies that must follow to the snake in the sport.

3 METHODOLOGY

3.1 Search Algorithms

A dataset's searching methods are used to locate or search for one or more factors. These techniques are used to search for additives within a data structure. It is possible to search in a sequential or non-sequential manner. Sequential searching is required if the facts in the dataset are random. In all other cases, a diffusion of strategies can be used to reduce complexity. In AI, searching is the go-to method for resolving problems. Single-player video games such as tile games, Sudoku, crossword puzzles, and other single-player video games are available. Search algorithms aid you in discovering a specific spot in such games.

3.1.1 Uninformed Searching Techniques

These are the techniques which blindly performs the search operation to reach the goal node.

3.1.1.1 Depth-First Search

The Depth-First Search investigates a vertex to its deepest level, then backtracks until it reaches unexplored vertices nodes. It's also referred to as an edge-based technique. The stack data structure, which uses a last-in-first-out method, is used for Depth-First Search. Depth-First Search iterates until it finds the desired node. Along the path, the Depth-First Search traverses the edges twice and the vertex just once. It has a linear space requirement, which means it cannot grow exponentially like Breadth-First Search when searching for a node in depth.

3.1.1.2 Breadth-First Search

The method involves travelling or examining the deepest node before moving on to the next node. That is, it explores or traverses the search tree by first expanding all nodes in the first level, then expanding in the second level, and eventually reaching the target. As a result, it's known as the level-by-level traversal method. All solutions for each node are obtained using this method. This ensures that the best option is found. The values are stored using a queue data structure in the Breadth-First Search. Even though the graph contains cycles, the visited vertices in the search tree are stored in an array. It operates on the FIFO principle (first in first out).

3.1.1.3 Uniformed Cost Search

Uniform Cost Search is used to find the cheapest way to traverse a graph tree to find a path. One of the state space search algorithms is this one. It finds the lowest cost of adjacent nodes using a priority queue. It goes backwards and comes up with a fresh solution for every conceivable route to the destination. Then it selects the cheapest route from root to destination. When the path that is explored fails to produce satisfactory results, it always analyses the paths minimum costs for a graph/tree and chooses the ideal cost. Dijkstra's single-source shortest path algorithm is also known as the Uniform Cost Search algorithm. When the minimal cost nodes are located, the successors with greater costs in the queue are deleted. When the minimum cost nodes are identified, it stops looking for a path and returns to the previous unexplored path if it cannot achieve the optimal state.

3.1.2 Informed Search Techniques

These are the techniques which uses the Heuristic function for the search operation.

3.1.3 Heuristic Search

When traditional approaches fail, a Heuristic Search is a methodology that assesses the available information each time while exploring to additional nodes to find an ideal answer in a precise amount of time. Heuristic Search algorithms outperform uninformed search algorithms in terms of speed. The Heuristic Evaluating Function is the cost be- tween the nodes in the states space search tree that is used to evaluate the problem attractiveness. The Heuristic Evaluating Function calculates the best cost between two nodes in a state space tree. It estimates optimal distances for each node iteratively until the goal state is reached. The issue domain, cost metrics , and heuristic information in the problem are all important factors in the heuristic assessment function. Admissibility is defined as the ability of heuristic functions to choose lower boundaries between two pairs of nodes than the real cost.

3.1.3.1 Best First Search

One of the most basic Heuristic Search algorithms is the Best First Search method. The Best First Search algorithm operates on the premise of following a set of rules when investigating the goal node. The heuristic function is used by the Best First Search method to estimate the specific rule in the issue area. The heuristic function is used to determine the distance between nodes in the state-space search tree. The open and closed list are the two lists used in the algorithm. The open list keeps track of the nodes in the state space tree that need to be investigated, while the closed list keeps track of the nodes that have been visited in the state-space search tree. When an open node finds the shortest path, it saves it and discards the longer one. The closed node is transferred to an open node associated with it once it has determined the minimal path. The best node in all the unvisited nodes in the state-space search tree is used in the Best First Search. It's a hybrid of the Depth-First Search and Breadth-First Search algorithms that uses a heuristic function to estimate node costs. The priority queue data structure is used by the Best First Search method to keep the distance between nodes in ascending order constant.

3.1.3.2 A* Search

The A* search algorithm combines the best qualities of Uniform Cost Search and pure Heuristic Search to seek an optimum solution, completeness of the path, and optimal efficiency in the state space search tree. The A* Search algorithm's goal is to identify a path from the starting node to the goal node by maintaining and extending the node tree until it reaches the goal state. The A* Search algorithm widens the path by applying cost estimation each time it iterates, facilitating in reaching the objective node. The A* Search algorithm uses the formulas $f(n) = g(n) + h(n)$ to find the shortest path through node n in the state space search tree, where g(n) is the cost of the path from the source node to n, and h(n) is the projected heuristic cost from the target node to n, and f(n) is the total optimal cost of a path through node n. If the heuristic function in the A* Search algorithm is admissible, the algorithm always chooses the least cost path from the start to the goal node. A* Search makes use of the priority queue data structure. At each phase, the lower f values are removed, and the f(n) and g(n) values' neighboring nodes are updated accordingly. When the least f(n) value is greater than any node in the queue, the loop terminates.

3.2 Minimax Search Algorithm

The Minimax algorithm is backtracking set of rules utilized in selection and game theory. It makes use of game theory, choice principle, records, and philosophy to discern out what a player ought to do if the opponent follows match. It's commonly used in two-player turn-based games like Tic-Tac-Toe, Chess, and others. A player must meet two requirements to win. First and foremost, a player must enhance his or her chances of winning the game. Second, the player must decrease the likelihood of the opponent winning. The objective is to find the optimum path to minimise the greatest overall loss. There are 2 conceivable outcomes: a positive result for system winning and a negative result for system losing.

3.3 Alpha-Beta Pruning

The Alpha-Beta Pruning technique optimizes the minimax algorithm. It significantly decreases computing time. It enables faster searching as well as searches deeper into the game tree. And chop the branches of trees which don't need to be explored for improved mobility. The purpose of alpha-beta pruning is to reduce the number of nodes in the search tree that the minimax algorithm must evaluate.

3.4 State Diagram in Sokoban

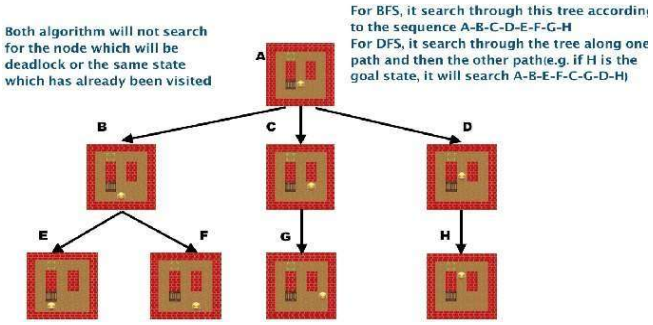

Figure 1 State diagram for Sokoban, depicting the sokoban next moves according to the BFS and DFS search algorihtm.

3.5 State Diagram in Tic Tac Toe

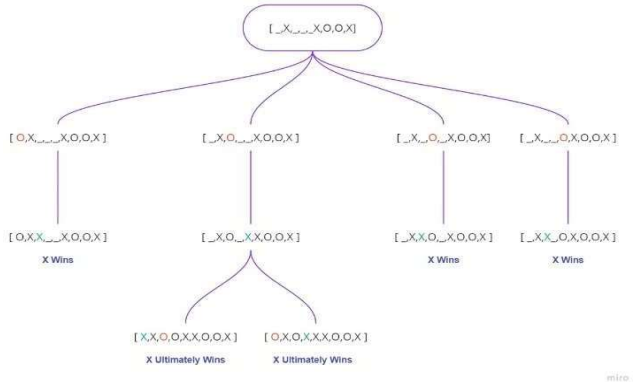

Figure 2 State Diagram of Tic Tac toe using Min Max Algorithm showing all possible moves

At each turn, the algorithm analyzes all potential consequences for each move and picks the one that ensures a win or a tie. The alpha beta search algorithm ensures that an AI player never loses. The value of each node, as well as the ideal game play, are represented in the figure below.

3.6 State Diagram in Snake Game

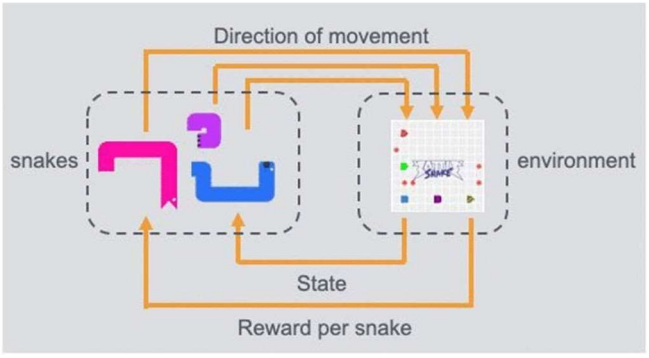

Figure 3 Environment and states in Snake Game

4 RESULTS

4.1 Sokoban

All the algorithms' results are based on common criteria such as time spent, and total number of states investigated. It was also discovered that the states explored by all the algorithms were nearly identical, differing primarily in terms of duration and response stages. There are two cost functions considered for UCS.

TABLE 1

Algorithm	Cost	Search Time (in sec)	Nodes expanded	States Generated	State cycle check pruned	State Cost Bound Pruned
A star Search	23	0.0799	2344	5790	3446	0
Depth First Search	23	0.079	2957	7407	4450	0
Breadth First Search	23	0.069	2613	6517	3904	0
Best First Search	23	0.030	1111	2595	1484	0
Uniform Cost Search	23	0.079	2555	6382	3827	0

TABLE 2

Algorithm	Cost	Search Time (in sec)	Nodes expanded	States Generated	State cycle check pruned	State Cost Bound Pruned
A star Search	35	0.349	9510	22322	12812	0
Depth First Search	55	0.099	3895	9267	5372	0
Breadth First Search	35	0.299	10624	24948	14324	0
Best First Search	53	0.209	5211	12300	7089	0
Uniform Cost Search	35	0.350	10980	25891	14911	0

TABLE 3

Algorithm	Cost	Search Time (in sec)	Nodes expanded	States Generated	State cycle check pruned	State Cost Bound Pruned
A star Search	27	0.429	11134	27805	16671	0
Depth First Search	47	0.430	14480	36955	22475	0
Breadth First Search	27	0.340	12667	31182	18515	0
Best First Search	27	0.010	650	1354	704	0
Uniform Cost Search	27	0.450	12620	31102	18482	0

TABLE 4

Algorithm	Cost	Search Time (in sec)	Nodes expanded	States Generated	State cycle check pruned	State Cost Bound Pruned
A star Search	20	0.740	16990	34778	17788	0
Depth First Search	42	0.049	1379	3216	1837	0
Breadth First Search	20	1.090	35614	81246	45632	0
Best First Search	24	0.010	165	285	120	0
Uniform Cost Search	20	1.200	33932	77601	43669	0

TABLE 5

Algorithm	Cost	Search Time (in sec)	Nodes expanded	States Generated	State cycle check pruned	State Cost Bound Pruned
A star Search	41	0.560	15705	37176	21471	0
Depth First Search	59	0.170	5330	12689	7359	0
Breadth First Search	41	0.519	16556	39864	23308	0
Best First Search	41	0.109	3260	7644	4384	0
Uniform Cost Search	41	0.629	16607	40003	23396	0

TABLE 6

Algorithm	Cost	Search Time (in sec)	Nodes expanded	States Generated	State cycle check pruned	State Cost Bound Pruned
A star Search	41	0.649	15705	37176	21471	0
Depth First Search	59	0.140	5330	12689	7359	0
Breadth First Search	41	0.5	16556	39864	23308	0
Best First Search	41	0.140	3260	7644	4384	0
Uniform Cost Search	41	0.579	16607	40003	23396	0

Best First search performed best overall in all the problems among all other searching algorithms. It is most cost efficient and took least time among other search algorithms. Uniform Cost Search performs worst among all as it took the most searching time among all other, but depth first search turns out to be the costliest algorithm among all.

4.2 Tic Tac Toe

Minimax makes a lot of moves immediately and can be difficult to debug at times. With a few simple changes to the board geometry and how we loop through the actions, the minimax algorithm can be implemented in any 2-player board sport. Furthermore, for complicated video games like Chess, it's not always feasible for minimax to compute each plausible game kingdom. As a result, we solely compute up to a certain intensity and use the assessment feature to calculate the board's worth.
AI won 7 out of 10 times we played the game, and match was drawn only 2 times.

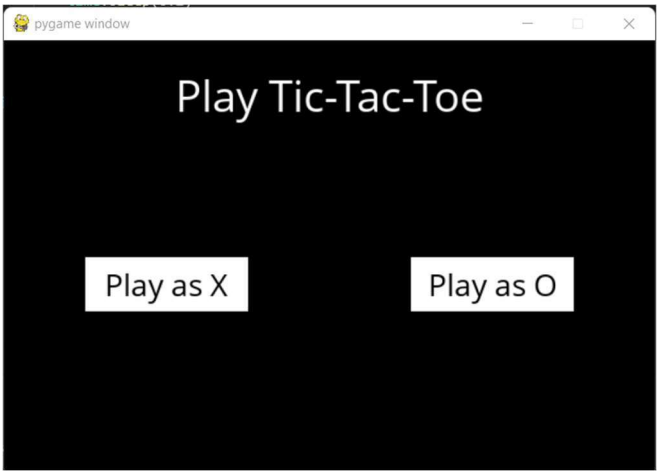

Figure 5 PyGame window showing game in progress

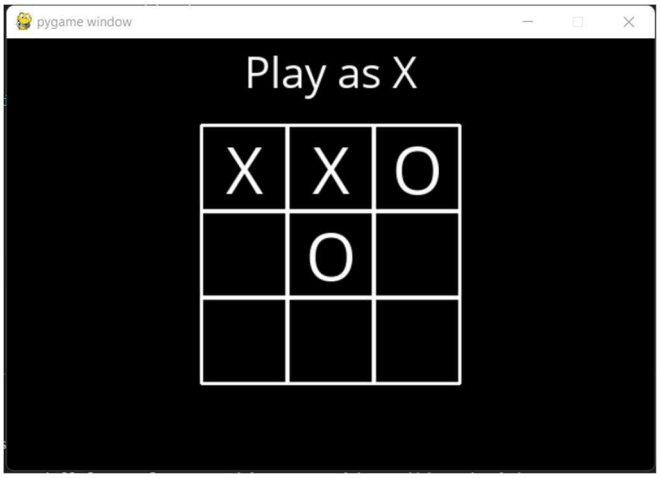

Figure 4 Starting window of Tic Tac Toe on a PyGame window

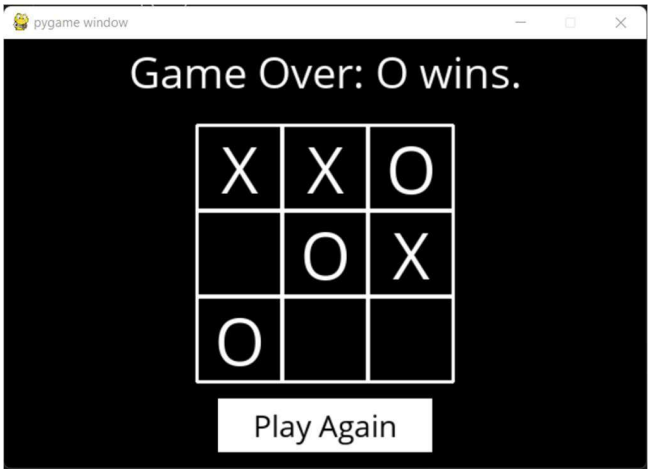

Figure 6 AI won the game against human agent

4.3 Snake Game

In this section, we see the experimental results of using different search algorithms for playing Snake game:

TABLE 7

Algorithm	First Trial	Second Trial
BFS Actions	959	841
DFS Actions	4174	3744
A-star Actions	389	709
UCS Actions	447	691

TABLE 8

Algorithm	First Trial	Second Trial
Raw BFS Score	40	40
Raw DFS Score	76	71
Raw A-star Score	37	58
Raw UCS Score	42	58

TABLE 9

Algorithm	First Trial	Second Trial
Calc BFS Score	0.9583133684714902	1.0683760683760684
Calc DFS Score	7.924921793534932	8.442330558858501
Calc A-star Score	9.511568123393316	8.180535966149506
Calc UCS Score	9.395973154362416	8.393632416787264

After comparing Search algorithms, we can conclude that BFS performs the best overall. DFS runs slowly wit lots of actions, always lowest food score. BFS runs quickly and linearly, good food score, low actions. A star runs quickly and diagonally and good food score with more actions and UCS runs quickly and diagonally with good food score and more actions.'

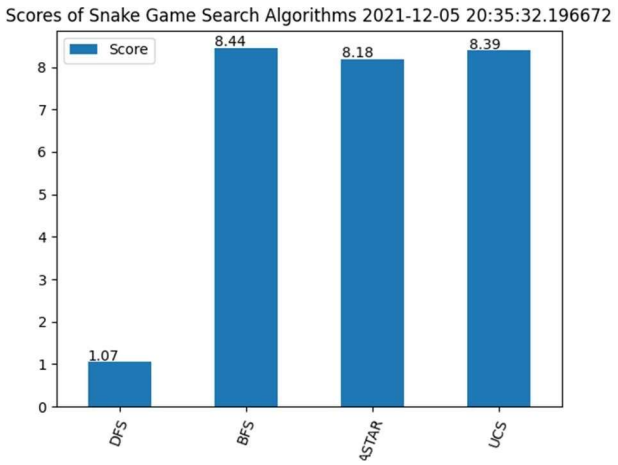

Figure 7 Comparison of calculated scores of Snake Game Search Algorithms

5 Discussion

In this study, we presented five ideas and techniques for creating an automated Snake game and Sokoban solver. We looked at the overall performance of a number of different algorithms by analyzing them and conducting experiments. While informed search algorithms may appear to have a respectable level of reliability and efficiency at the start of the game, these characteristics vanish towards the finish. In conclusion, there is still a lot of work to be done to improve our solver. More advanced kinds of deadlock detection (such as detecting corral deadlocks), macro moves for pushing blocks to goals and travelling through tunnels, and strategies that divide levels into discrete sub- problems are among the techniques we discovered during our research. Nonetheless, the research reported in this study is still relevant to the topic of solving smaller Sokoban puzzles.

We have proven that when memory is not a constraint, simpler heuristics and algorithms perform better. In general, distance-based heuristics tend to be computationally expensive and do not successfully prune the search tree. Domain-specific heuristics like deadlock detection, on the other hand, are quite effective. This implies that algorithms that work well on Sokoban will most likely not work well on other problems. Overall, this challenge is both fascinating and difficult from an AI standpoint, since it demonstrates how tough it is to make state-space search algorithms effective when applied to situations with enormous search areas.

The tic tac toe game was discussed and built utilizing an artificial intelligence heuristic technique in this research. Min-max and alpha-beta pruning algorithms have exceptional ability to produce excellent results in tic-tac-toe.

In some games, AI performs better than human agent for certain algorithms but lack in other, similarly, some algorithms performs only in certain type of environments and games.

Acknowledgment

We'd like to express our gratitude to Dr. Alok Kumar, our project instructor/manager and faculty supervisor, for patiently teaching us the fundamentals of the duties and guiding us through the research. We thank the assistance of these individuals with a great sense of pleasure and gratitude.

We'd like to express our gratitude to Dr. Sanjay Goel, our department's head, for his constructive feedback throughout this study. We owe a debt of gratitude to our department, coworkers, and friends for assisting us in completing our paper.

References

[1] Klavík, Pavel. n.d. "Sokoban Solver-a Short Documentation." Accessed December 5, 2021. http://pavel.klavik.cz/projekty/solver/solver.pdf.

[2] Li, Zheng, et al. "Object-oriented Sokoban solver: A serious game project for OOAD and AI education." Frontiers in Education Conference (FIE), 2014 IEEE. IEEE, 2014.

[3] Taylor, Joshua, and Ian Parberry. "Procedural generation of Sokoban levels." Proceedings of the International North American Conference on Intelligent Games and Simulation. 2011.

[4] Siby Samuel, Kajal Mahawar and Isac João França, Improved technique in Tic-Tac-Toe game to minimize the condition of draw using min-max over optimal strategy.International Journal of Scientific& Engineering Research Volume 10, Issue 3, (2019) 663 ISSN 2229-5518

[5] Using Tic-Tac-Toe for Learning Data Mining Classifications and Evaluations by Chen-Huei Chou http://www.ijiet.org/papers/314-k010.pdf

[6] Anurag Bhatt, Pratul Varshney and Kalyanmoy Deb .Evolution of No-loss Strategies for the Game of Tic-Tac-Toe.Department of Mechanical Engineering, Indian Institute of Technology Kanpur. https://www.iitk.ac.in/kangal/papers/k2007002.pdf

[7] The Analysis of Alpha Beta Pruning and MTD(f) Algorithm to Determine the Best Algorithm to be Implemented at Connect Four Prototype by Lukas Tommy, Mardi Hardjianto and Nazori Agani; DOI : https://doi:10.1088/1757-899X/190/1/012044

[8] https://intellipaat.com/community/26314/minmax-trees-when-min-can-win-in-two-steps

[9] https://www.codeproject.com/Articles/7344/A-brute-force-search-algo

[10] Rina Dechter and Itay Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. Artificial Intelligence, 68(2):211– 241, 1994.

[11] Zain, A. M., C. W. Chai, C. C. Goh, B. J. Lim, C. J. Low, and S. J. Tan. "Development of Tic-Tac-Toe Game Using Heuristic Search." In *IOP Conference Series: Materials Science and Engineer- ing*, vol. 864, no. 1, p. 012090. IOP Publishing, 2020.

[12] Sharma, Shubham, Saurabh Mishra, Nachiket Deodhar, Akshay Katageri, and Parth Sagar. "Solving The Classic Snake Game Using AI." In *2019 IEEE Pune Section International Conference (Pu-*

neCon), pp. 1-4. IEEE, 2019.

[13] Garg, Roopali, and Deva Prasad Nayak. "GAME OF TIC-TAC-TOE: SIMULATION USING MIN-MAX ALGORITHM." *Interna- tional Journal of Advanced Research in Computer Science* 8, no. 7 (2017).

[14] Froleyks, Nils Christian, and Tomas Balyo. "Using an algorithm portfolio to solve sokoban." In *Tenth Annual Symposium on Com- binatorial Search*. 2017.

[15] Venkatesan, Anand, Atishay Jain, and Rakesh Grewal. "AI in Game Playing: Sokoban Solver." *arXiv preprint arXiv:1807.00049* (2018).

[16] Hemanth, D. J. "Analysis of Minimax Algorithm Using Tic-Tac-Toe." (2020).

[17] Appaji, Naga Sai Dattu. "Comparison of Searching Algorithms in AI Against Human Agent in the Snake Game." (2020).

ANNEXURE

GitHub link: https://github.com/saumyapatel17/AI-Games-using-Search-Algorithms

Saumya Patel BTech Computer Science and Engg (2019-23); AI Intern – Anemoi Technologies Pvt. Ltd. (May-August 2021); Data Science, Artificial Intelligence and Machine Learning

Utkarsh Sharma BTech Computer Science and Engg (2019-23); Web Development Intern – Ownrox Technologies Pvt. Ltd. (May-August 2021); Web development, Artificial Intelligence and Machine Learning

Prakhar Saxena BTech Computer Science and Engg (2019-23); Digital Marketing Intern – MCME Techserve Pvt. Ltd.;Web Development