Playing the SOS Game Using Feasible Greedy Strategy

Abas Setiawan*
Department of Informatics Engineering, Faculty of Computer Science, Universitas Dian Nuswantoro
Semarang 50131, Indonesia
Email: abas.setiawan@dsn.dinus.ac.id

***Abstract*—The research aims to make an intelligent agent that can compete against the human player. In this research, the feasible greedy strategy is proposed to make an intelligent agent by checking all possible solutions in the limited tree levels to find effective movement. Several matches are conducted to evaluate the performance of the feasible greedy agent. The board size for the evaluation consists of 3×3, 4×4, 5×5, 6×6, 7×7, and 8×8 squares. From the result, the feasible greedy agent never loses against the random agent and the pure greedy agent. In 3×3 squares match, the agent can compensate against the human player, so the game always ends with a draw. In 4×4, 5×5, 6×6, 7×7, and 8×8 squares matches, the feasible greedy agent slightly outplays the human player.**

***Index Terms*—SOS Game, Feasible Greedy Agent, Greedy Strategy, Game Tree**

I. Introduction

SOS game is a kind of paper-and-pencil game similar to tic-tac-toe with greater complexity. Two players play the SOS game where the set of possible positions are the same movement between the players. They can play the SOS game with turn-based playing. The objective of this game is to make the sequence of S-O-S pairs among the connected square as many as possible. The S-O-S pairs can be formed vertically, diagonally, or horizontally.

In the game theory, the SOS game is a combinatorial game involving two players. The combinatorial game means two players play the game with the set of possible positions (usually finite) and no chance moves or dice. It has comprehensive information for both players, and there is no distinction of movement between the players [1, 2]. The SOS game meets the combinatorial game condition and also a zero-sum game. Zero-sum means one player gains a score, and it equals the loss of another player (as payoffs) so that the total sum is zero [3]. The SOS game has greater complexity than tic-tac-toe. Moreover, the game has more than 3×3 squares of the board size.

In the usual game, the SOS game is played by two players. One player creates a game board by drawing a square grid with the size of at least 3×3 squares. Then, two players choose the turn. For each turn, the player draws S or O inside the empty square. The objective of this game is that each player makes the sequence of S-O-S among connected square as many as possible. The connection can be vertically, diagonally, or horizontally. When a player successfully creates one pair of S-O-S, a player can make one or more move again until there is no S-O-S pair created. Thus, the turn will end, and the next player moves to draws S or O in another empty square [1].

The game will end after there is no empty square inside of the game board. To track the S-O-S pairs that are successfully created by the player, he/she will draw the lines. The winner of this game is a player who has collected the highest S-O-S pairs. If two players have the same number of collected S-O-S pairs (including no S-O-S pair created), the game result is a draw [4]. When the SOS game is played in more than 7×7 squares, the players should move carefully. If a player makes a mistake, it can give the point to the other player.

The research aims to present SOS game as a digital game. To create an agent with the ability to play the SOS game, the researcher proposes the greedy strategy or greedy algorithm with some modification. Nowadays, the greedy strategy is still used in many studies like sensor placement [5], task scheduling [6], and detecting a mutually exclusive pattern in cancer mutation data [7]. The greedy strategy is also effective to be used in digital or board games, such as a card game [8], an educational game [9], and a puzzle game [10].

Furthermore, the greedy strategy is proposed because it matches the SOS game-play. Unfortunately, when the agent uses a pure greedy strategy, there are some problems. When the greedy agent does not

Received: Dec. 12, 2019; received in revised form: Mar. 18, 2020; accepted: Mar. 18, 2020; available online: Apr. 22, 2020.
*Corresponding Author

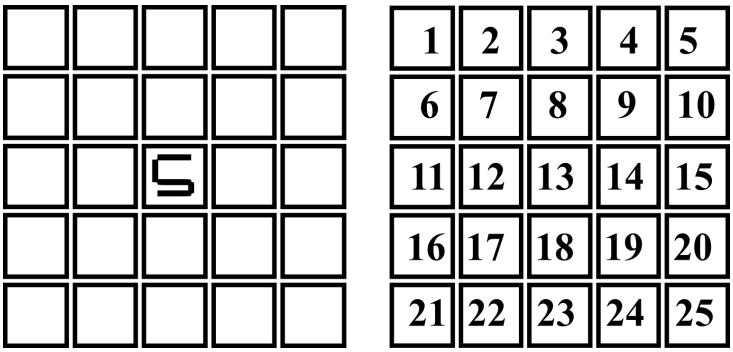

Fig. 1. The SOS board (left) and the SOS board with its position (right).

notice a chance to make S-O-S pair, the greedy agent will place the "S" or "O" randomly on the board. In consequence, the pure greedy agent cannot guarantee that the move is correct or wrong. Thus, the researcher proposes the improvement of the greedy strategy by checking all possible solutions in the limited tree levels.

There is no related work with the SOS game research. Most of them are related to the tic-tac-toe game. The tree-based search methods, such as the Minimax [11] or Alpha-Beta Pruning algorithm, work well in tic-tac-toe [12]. Reference [13] made a robot to solve the tic-tac-toe game with Minimax as the main algorithm. Similarly, Ref. [14] used one of the concepts of automata theory, which is the Multi-tape Turing Machine algorithm, to solve the tic-tac-toe game with optimal results.

Moreover, extensive research uses theoretical computer science. It is proven that this method will always give a draw if the enemy is playing optimally [15]. The solution to playing tic-tac-toe is not only solved by the tree-based search method, but also with the method of machine learning or reinforcement learning [16].

The tic-tac-toe game and the SOS game are different. However, the SOS game size is the same as the tic-tac-toe which is 3×3 squares. The best result against an enemy with the optimal moves is a draw. The reason that Minimax or Alpha-Beta is not applied because the players can get another turn (combo move) after successfully making the S-O-S pair. Hence, the tree levels are inconsistent for each player's turn.

In addition, the machine learning method is not applied in this research because the size of the board to be evaluated is not only 3×3 squares, but it also can be up to 8×8 squares. If the algorithm is based on a comprehensive tree search or training algorithm applied to large board size, it may take a long time to build the overall tree structure or training process. The main contribution of this research is to use a greedy strategy in a limited tree search to create an intelligent agent that can play effectively and optimally in the board size of 3×3, 4×4, 5×5, 6×6, 7×7, and 8×8 squares.

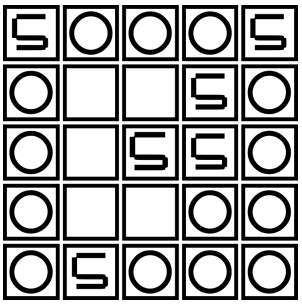

Fig. 2. The SOS board after several moves.

II. Research Method

The research applies a structured method in several phases. The first phase is analyzing the SOS gameplay. The second phase is to design the feasible greedy agent to play the SOS game. Then, the third phase is implementation. Last, the fourth phase conducts the playtesting evaluation.

A. The Analysis of the SOS Gameplay

Before the game start, every player must know the game rules and the information about the preceding events. The analysis of SOS gameplay starts with the advantage of choosing the player's turn. Suppose that there are two players: player 1 (P1) and player 2 (P2), and the game board has 5×5 squares. In the case of 5×5 squares, the possible way to fill all 25 squares is 3^{25}. There are three states for each square in 25 squares. The three states are empty, marked by "S", and marked by "O". Figure 1 depicts the sample of the SOS board and its position.

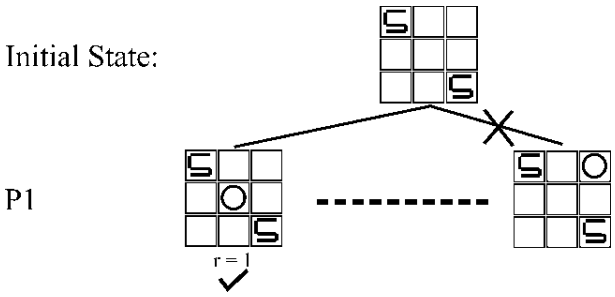

Fig. 3. The SOS game tree with the correct place in level two.

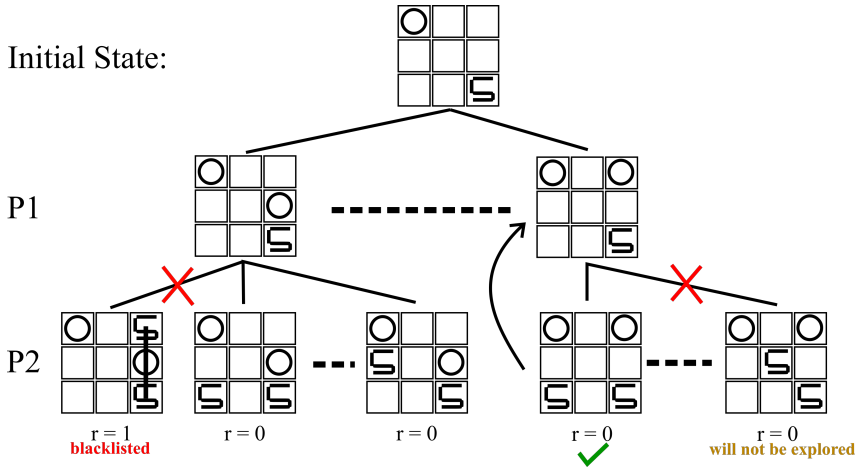

Fig. 4. The SOS game tree with a depth of two.

At the beginning of the game, if the first turn is for P1, P1 can place the "S" or "O" in one of all empty squares randomly, and then, P2 takes a turn. Usually, P2 makes a decision after exploring the board. The board, for instance, matches the condition of Fig. 1 (left). There is the "S" mark placed in the board position of 13. P2 will avoid making "O" near the "S" or making "S" around of that square. P2 can block the other player's movement by placing "O" around the squares. Perhaps, P2 chooses "O" in the board position of 1, which is a safe place. After several moves conducted by both players, the game board is depicted in Fig. 2. In Fig. 2, the game board shows that a player can make the consecutive movement to win the game.

In this case, if the SOS game is played digitally, there is a chance for a human player to play against the computer's agent. For the weak computer agent, all of their movements will choose randomly. It can give a chance to the human player to make an S-O-S pair as soon as possible.

Therefore, a human player can beat that agent easily and not challenging. Then, the greedy strategy is applied in the agent. The greedy agent is very effective when the board situation is like in Fig. 2. Nevertheless, when the greedy agent meets the board like in Fig. 1 (left), it still possible for the agent to place "S" or "O" in the restricted area. It is because of the randomness of the greedy strategy in making any decision. So, the researcher will make a greedy agent more considerate of that situation.

B. The Agent Design

Designing the agent for making decisions in a particular state is represented as a game tree. The agent uses a pure greedy strategy immediately after another player makes a mistake. A pure greedy strategy is illustrated in Fig. 3 when the agent meets the correct place in level 2 of the game tree. The board size formed is in 3×3 squares to simplify the situation.

Figure 4 represents the game tree states with a depth of 2. The situation in Fig. 4 shows that the agent cannot make any S-O-S pairs. To avoid moving in a restricted area, the agent will take a look at the next tree level. Moreover, at level 3, it shows all possible states for another player's movement while another player's state successfully creates S-O-S pair. Then, the former state will not have explored. The agent chooses only a state with no possible S-O-S pair for another player.

For example, the initial state is s=["O","","","","","","S"]. Then, the agent will find the correct position by checking one-by-one in level 2 of a tree. In this case, the agent is P1. The agent chooses the value of "O" in the square position of 6 so that the P1 state is s_{P1}=["O","","","","O","","", "S"]. The agent's move is bad because another player can easily pick the square position of 3 and the value of "S", so the state is s_{P2}=["O","","S","","", "","","O","","","S"]. Then, the S-O-S pair is formed by the combination of the square position of 3, 6, and 9. After seeing that possibility, the agent will not choose the position of 6, and the prior state will not be explored. When the agent successfully makes a good enough movement, there are two possibilities, a safe position or blocking the enemy. The safe position is the position which not related to the last player's position. Blocking the enemy is that the agent chooses the value of "O" in near another player's position. The blocking area can be diagonally, vertically, or horizontally.

From the previous explanation, the position and the value must be chosen effectively. Then, the position and value are included in the utility property. There is one additional property called as the result of the S-O-S pair. The three property is the position (p), the value (v), and the result (r). The p is a square position and correlated with the board size. The value of $v \in \{'S', 'O', ''\}$ the value of "S" or "O" or empty filled inside a square board. Meanwhile, the r is the heuristic score. When the S-O-S pair presents in the game tree, the r has a value of 1. Otherwise, it is 0. The complete algorithm to collect the utility property is presented in Algorithm 1.

There is a function called *CheckSOSPair* to calculate the r by comparing the last move by another player with the subsequent solution or winning combination set. The winning combination set contains all possible positions that can form S-O-S pairs. It is generated in the early game.

The search space of Algorithm 1 is not linear. When the S-O-S pair does not present, the p and v will be chosen randomly in all possible empty squares (s_e). It is intended that another player does not easily read the agent's movements. After defining the algorithm to get the utility property, the optimal move solution is found based on Algorithm 2. Algorithm 2 needs the initial state of s.

Algorithm 2 returns two outputs: the correct position and value. The temporary $p*$, $v*$, and $r*$ are the utility property after applying the position and value for a certain node of level 2. The $r*$ has the purpose of evaluating the next move. When the next move (taken by another player) makes S-O-S pair, the last agent's state is blacklisted. Then, the agent will search for another possible state in level 2.

Algorithm 1: Get Utility

Input: state s
Output: result r**, position** p**, and value** v
function GetUtility (s)
  s_e ← **All empty squares from** s
  p ← **0**
  v ← **"S"**
  found ← **false**
  for each i in s_e do
    v ← **{"S", "O"}**
    $s[i]$ ← v
    r^* ← ***CheckSOSPair***($s[i]$)
    $s[i]$ ← **""**
    if $r^* = 1$ then
      r ← r^*
      p ← $s[i]$
      found ← **true**
      break
    endif
  endfor
  if not **found** then
    p ← **random choice of** s_e
    if **random** $\in [0,1]$ **> 0.5** then
      v ← **"O"**
    endif
  endif
  return $[p,v,r]$

Algorithm 2: Find Solution

Input: **state** s
Output: **position** p **and value** v
function **FindSolution**(s)
  $[p,v,r]$ ← ***GetUtility*** (s)
  if $r = 1$ then
    return $[p,v]$
  else
    $s[p]$ ← v
    $[p^*,v^*,r^*]$ ← ***GetUtility*** (s)
    c ← **0**
    while $r^* = 1$ do
      $s[p]$ ← **""**
      $[p,v,r]$ ← ***GetUtility*** (s)
      $s[p]$ ← v
      $[p^*,v^*,r^*]$ ← ***GetUtility*** (s)
      c ← $c + 1$
      if $c == (board\ size \times 2)$ then
        break
      endif
    endwhile
    return $[p,v]$
  endif

Sometimes, there is a situation when all possible states in level 3 tree have a utility result of 1. It

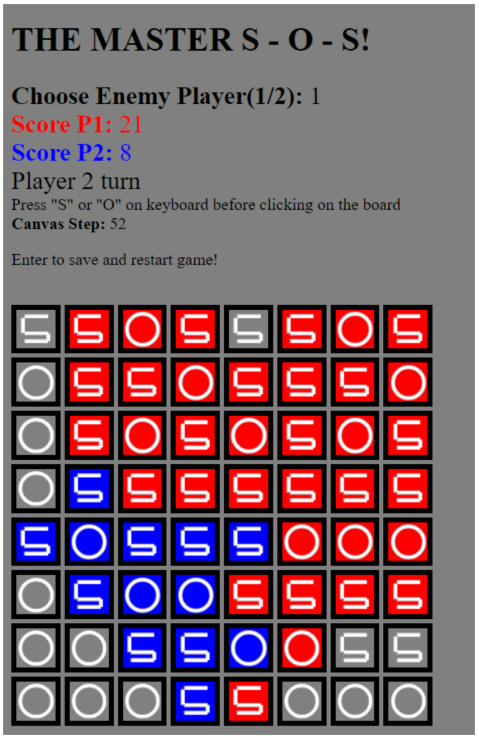

Fig. 5. The interface of the game environment.

means all the states in level 3 contains an S-O-S pair. Algorithm 2 will undergo continuous repetition. Therefore, it needs to break that repetition and selects one of the states randomly. The c variable is presented to counter the repetition. The c variable will increment until the number of board size is multiplied by two.

C. Implementation of the Agent

The SOS game is running on the web platform. The agent is implemented by using the javascript programming language. If a human player wants to play this SOS game, the person must comply with the following game procedures:

1) The human player selects the enemy (agent) first (whether the agent will be the first or second player).
2) If the first player is an agent, the agent will immediately play and proceed with a changing turn. If the agent is the second player, the human player will play first.
3) When the human player's turn arrives, she/he must press the "S" or "O" button on the keyboard and click the mouse on the game board.
4) The human player or agent can make consecutive moves when it finds an SOS pair on the game board.

TABLE I
THE PERFORMANCE OF THE FEASIBLE GREEDY AGENT AGAINST THE RANDOM AGENT AND THE PURE GREEDY AGENT.

Board size	Random			Pure greedy		
	Win	Lose	Draw	Win	Lose	Draw
3×3	123	0	77	101	0	99
4×4	195	0	5	196	0	4
5×5	200	0	0	200	0	0
6×6	200	0	0	200	0	0
7×7	200	0	0	200	0	0
8×8	200	0	0	200	0	0

Figure 5 is the game environment. The red color represents the S-O-S pair for P1 and the blue color for P2. Game statistics are also available on the top page of the game environment.

III. Results and Discussion

The evaluation of the feasible greedy agent is conducted with 200 matches against the random agent, the pure greedy agent, and the human player. Both players alternately become the first player and second player. The tree of states formed by the agent is the search space for the agent, and it is not structured as a tree data. At the beginning of the game, the agent who acts as the first player is only evaluated up to level 2. For agents to go to level 3, there must be at least the possibility of a state that can form S-O-S pairs. That S-O-S pair is a point for the second player. So, if the beginning of the game is an agent, the agent will immediately get a random position in the tree level of 2.

When the agent plays as the second player, the first player will choose the position first (usually random). After that, the agent will create a search space in the tree based on the initial state performed by the first player. Thus, there will be S-O-S pairs that may form in the level 3. When the agent encounters this situation while searching the optimal position, the agent will repeat the search starting from other states in level 2 by randomly selecting one of them.

Unlike the case in which the agent successfully meets a position that can form a pair of SOS at level 2, the agent will immediately take that position as an optimal step (greedy). The results of 200 matches against the random agent and pure greedy agent are presented in Table I. The researcher decides to use the six types of board size. Those are 3×3, 4×4, 5×5, 6×6, 7×7, and 8×8 squares.

In Table I, the performance of the feasible greedy agent is significant and never loses against the random agent and pure greedy agent. It is because the random agent or pure greedy agent sometimes makes a mistake by placing "S" or "O" randomly when the agent cannot

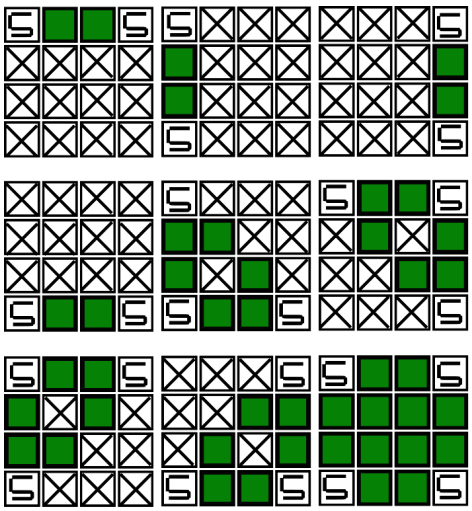

Fig. 6. No option condition.

make the S-O-S pair. In the board size of 3×3 squares, there is no chance for the random agent or the pure greedy agent to win the game. In larger board sizes, random agent and greedy agent make more mistakes.

The next evaluation is the match between the human player against the feasible greedy agent. It is by considering that the human player is a normal player or not a master in playing the SOS game. In this case, there are no performance measurements for an SOS player leveling evaluation as a chess master or go master. Then, the normal player will take the random movement when she/he faces the no option condition. The no option condition depicted in Fig. 6. The no option condition appears when the minimum board size is 4×4 squares. There are nine conditions. Each position can be found separately or jointly.

The green color in Fig. 6 is the no option area. Anyplace in that area will cause an S-O-S pair for another player. No option condition will appear after all players play the game until almost the end, in which there are some movements that all players conduct. The "X" mark in Fig. 6 is the area that has been filled with "S" or "O". A normal player perhaps never think too long to solve the no option condition. However, a master player may think carefully and move optimally to solve that condition.

The number of matches between the feasible greedy agent and the human player are 200. The performance of the feasible greedy agent against the human player is depicted in Table II. In the board size of 3×3 squares, the human player plays optimally because she/he never meets no option condition. The human player can easily identify the game board because the number of squares is still minimum. The human player and agent

TABLE II
THE PERFORMANCE OF THE FEASIBLE GREEDY AGENT AGAINST THE HUMAN PLAYER.

Board size	Win	Lose	Draw
3×3	0	0	200
4×4	88	84	28
5×5	103	94	3
6×6	118	82	0
7×7	135	65	0
8×8	147	53	0

are both playing optimally so that no one wins the game in the board size of 3×3 squares. An agent has the possibility of defeating the human player with a minimum board size of 4×4 squares. With a few mistakes made by the human player, the agent will immediately take the opportunity.

To illustrate an agent defeating the human player, the researcher considers the board size of 5×5 squares. The player places the "S" mark in the 13^{th} square index (see Fig. 1 to find out the square index), even though there is an "S" mark before in the 1^{st} square index. Then, an agent will immediately place "O" in the 7^{th} square index with the solution, as shown in Fig. 3. If the agent does not see the S-O-S pairs, that can be made again, it will place the "S" or "O" mark in a safe position with the solution, as shown in Fig. 4, as long as it does not meet the no option condition. If the number of S-O-S pairs obtained by an agent is more than a human player, the agent wins the game.

In Table II, the major reason why the feasible greedy agent and human player loss is meeting one or several no option conditions. Besides, the greater board size makes the performance of a human player decrease. The feasible greedy agent will more easily defeat the human player who is in a hurry or carelessness. However, if the human player has a lot of considerations in playing, the game time will take a long time. In the overall match, the feasible greedy agent makes every movement in less than 100 milliseconds.

IV. CONCLUSION

The feasible greedy strategy is successfully implemented in the agent to play the SOS game. The feasible greedy strategy considers the next move in the deeper level of the tree so that the agent does not carelessly make a move at the beginning of the decision. The feasible greedy agent can effectively play better than the random agent and the pure greedy agent.

When the feasible greedy agent plays against the human player, the agent can compete well. In the board size of 3×3 squares, it shows that the agent makes optimal movements that can compensate for the human player. So, the game will end in a draw. From the 200

matches conducted for the board size of 4×4, 5×5, 6×6, 7×7, and 8×8 squares, it reveals that the agent slightly outplays the human being. Although the result is not significant, it can be said that the ability of a feasible greedy agent is at the human-level.

The reason why this agent cannot exceed human ability is because of the possibility of no option conditions in which the agent has no other choice to place "S" or "O" in any board position. A human who is experts in playing SOS can estimate the optimal thinking to determine the movement when meeting the no option condition. Perhaps, two possibilities can handle the no option condition. It is calculating the optimal movement when meeting these conditions or avoiding these conditions by making optimal movement in the early game. In future research, it will be more challenging if the agent can handle the no option condition.

ACKNOWLEDGEMENT

The researcher is indebted to the Faculty of Computer Science at Universitas Dian Nuswantoro, which provides financial support for this research.

REFERENCES

[1] T. S. Ferguson, *Game theory*, 2nd ed. Los Angeles: Mathematics Department, UCLA, 2014.

[2] R. J. Gould, *Mathematics in games, sports, and gambling: The games people play*. Boca Raton: CRC Press, 2015.

[3] E. K. Donkoh, R. Davis, E. D. Owusu-Ansah, E. A. Antwi, and M. Mensah, "Application of combinatorial techniques to the Ghanaian board game Zaminamina draft," *European Journal of Pure and Applied Mathematics*, vol. 12, no. 1, pp. 159–175, 2019.

[4] D. Keiser and B. McGee, *Patterns - Literature, arts, and science*. Prufrock Press Inc., 2008.

[5] B. E. Aydin, H. Hagedooren, M. M. Rutten, J. Delsman, G. H. Oude Essink, N. van de Giesen, and E. Abraham, "A greedy algorithm for optimal sensor placement to estimate salinity in polder networks," *Water*, vol. 11, no. 5, pp. 1–17, 2019.

[6] Z. Dong, N. Liu, and R. Rojas-Cessa, "Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers," *Journal of Cloud Computing*, vol. 4, no. 1, pp. 1–14, 2015.

[7] C. Yang, T. Zheng, Z. Zhao, X. He, X. Zhang, X. Xiao, and J. Wang, "A greedy algorithm for detecting mutually exclusive patterns in cancer mutation data," in *International Work-Conference on Bioinformatics and Biomedical Engineering*. Granada, Spain: Springer, May 8–10, 2019, pp. 154–165.

[8] K. Lichy, M. Mazur, J. Stolarek, and P. Lipiński, "The use of heuristic algorithms: A case study of a card game," *Journal of Applied Computer Science*, vol. 26, no. 2, pp. 107–116, 2018.

[9] I. Toma, C. E. Alexandru, M. Dascalu, P. Dessus, and S. Trausan Matu, "Semantic Boggle: A game for vocabulary acquisition," in *European Conference on Technology Enhanced Learning*. Tallinn, Estonia: Springer, Sept. 12–15, 2017, pp. 606–609.

[10] E. Butler, E. Torlak, and Z. Popović, "Synthesizing interpretable strategies for solving puzzle games," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, Hyannis, MA, USA, Aug. 14–17, 2017, pp. 1–10.

[11] R. Garg and D. P. Nayak, "Game of tic-tac-toe: Simulation using Min-Max algorithm," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 7, pp. 1074–1077, 2017.

[12] P. Borovska and M. Lazarova, "Efficiency of parallel Minimax algorithm for game tree search," in *CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies*, University of Rousse, Bulgaria, June 14–15, 2007, pp. 1–6.

[13] S. Jain and N. Khera, "An intelligent method forsolving tic-tac-toe problem," in *International Conference on Computing, Communication & Automation*. Noida, India: IEEE, May 15–16, 2015, pp. 181–184.

[14] S. Garg and D. Songara, "The winner decision model of tic tac toe game by using multi-tape turing machine," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Jaipur, India: IEEE, Sept. 21–24, 2016, pp. 573–579.

[15] S. Garg, D. Songara, and S. Maheshwari, "The winning strategy of tic tac toe game model by using theoretical computer science," in *2017 International Conference on Computer, Communications and Electronics (Comptelix)*. Jaipur, India: IEEE, July 1–2, 2017, pp. 89–95.

[16] D. Draskovic, M. Brzakovic, and B. Nikolic, "A comparison of machine leaming methods using a two player board game," in *IEEE EUROCON 2019-18th International Conference on Smart Technologies*. Novi Sad, Serbia: IEEE, July 1–4, 2019, pp. 1–5.