# Artificial Intelligence - Game Solvers

Arnav Bhattacharya
*Trinity College Dublin*
Dublin, Ireland
*bhattaar@tcd.ie*
Student ID: 22307812

***Abstract*—This research paper aims to create Tic Tac Toe and Connect 4 games in Python and compare the performance of two reinforcement learning algorithms: Minimax with alpha-beta pruning and tabular Q-learning. A default opponent will also be created that performs better than a random opponent. The comparison will be based on various criteria such as games played against the default opponent, games played against each other, and overall performance.**

***Index Terms*—Tic Tac Toe, Connect 4, Minimax algorithm with Alpha-beta pruning, Tabular Q-learning, Reinforcement learning, Default opponent, Performance comparison,**

## I. Introduction

Reinforcement learning (RL) is a widely used machine learning technique that has been applied to various domains, including games. This research paper implements Tic Tac Toe and Connect 4 games in Python and compares the performance of two RL algorithms: Minimax with alpha-beta pruning and tabular Q-learning against a default opponent. Minimax is an algorithm for playing two-player zero-sum games that tries to minimize the maximum possible loss. Alpha-beta pruning is a variation of Minimax that reduces the number of game states that need to be evaluated. Tabular Q-learning is a model-free RL algorithm that learns a value function through trial-and-error interactions with the environment.

A default opponent is also implemented that performs better than a random opponent by selecting winning or blocking moves if they exist. The performance of the RL algorithms is compared by letting them play against the default opponent and each other.

## II. Explanation of the Aglorithms

### A. MiniMax with Alpha Beta Pruning

Alpha-beta pruning is a sophisticated device wielded within the Minimax algorithm in order to attenuate the magnitude of game states which necessitate evaluation. The rudimentary principle that constitutes the foundation of alpha-beta pruning stipulates that any game state lacking optimal quality, unequivocally inferior to those previously subjected to thorough evaluation procedures, must be completely and utterly disregarded in a catastrophic manner so as not to affect, even minutely, the definitive process of decision-making.

Perplexing as the task at hand may seem, we cannot fully fathom the depth of intricacy inherent in alpha-beta pruning techniques sans thorough contemplation. Therefore, let us cogitate on a rudimentary exemplar that bespeaks such complexities with utmost clarity: behold, a Minimax search tree expressly designed to attain optimal gameplay strategy in tic-tac-toe! This arboreal manifestation consists of three tiers, with the apex denoting the current game state and its ensuing branches embodying all viable directions that both adversaries can undertake.

In order to assess the worthiness of the root node, the Minimax algorithm undertakes a recursive procedure whereby it evaluates, inter alia, each child node that embodies a conceivable move by the present player. Subsequently, the algorithm elects to proceed with and execute that move which results in the highest value garnered by its duly evaluated child node. That being emphasized, there is a foreboding tendency towards the exponential growth of possible moves available upon progression through deeper instants within this tree-like construct; ultimately resulting in ontological ramifications considering how computationally expensive such a methodology can turn out to be.

Alpha-beta pruning is a highly effective heuristic search algorithm that intricately utilizes the two pivotal references, alpha and beta, to constantly refine its exploration of potential solutions within a given search tree. Alpha symbolically embodies the highest known value discovered thus far during traversal from any starting point of the tree up to the specific node in question. Conversely, beta represents an exclusive lower bound on the lowest recorded value on any route emanating from either axis stemming off from said node down through its siblings.

Whilst the Minimax algorithm determines its way through the intricate branches of a tree, alpha and beta values are constantly reassessed and amended upon reaching each novel node encountered en route. Whenever a node's value falls below or exceeds alpha and beta thresholds respectively, the algorithm can confidently prune away any subtrees underneath that node in order to conserve computational resources. This highly efficacious technique is perfectly sound because it rests on the fundamental principle that whenever the current player takes on an amplifying role in maximizing game scores, any nodes functionally reduced below their corresponding alpha thresholds will be of no consequence whatsoever to said player; ergo, they will simply chart a different course towards one with greater returns. Conversely, if such a threshold breach precipitates while playing from a minimizing standpoint whereby seeking to minimize potential losses is regarded as optimal play then inversely here as well: other players may choose alternate moves leading them towards rewards much lesser than previous valuations witnessed before crossing beyond their respective beta–yet still remain acceptable nonetheless considering these lower estimates continue indicating preferable positions than those which fired off trips earlier down alongside that glistening but distant path.

Through selective removal of subtrees, alpha-beta pruning confers a marked reduction in the number of game states necessitating evaluation, endowing the Minimax algorithm with heightened efficiency. It bears noting that alpha-beta pruning can remarkably diminish the number of nodes evaluated by upwards of twofold or more contingent upon the structure of the game tree.

### *B. Tabular Q Learning*

Tabular Q-learning, an emulation of Markov decision processes, is a machine learning algorithm that engenders the act of instructing a computer program to perform optimal actions through evaluating hypothetical scenarios relative to rewards and penalties. The program acquires knowledge on how best to execute possible actions in response to specific states, gaining either auspicious rewards or deleterious penalties for each venture taken. Gradually accumulating experience from its collective decisions which maximizes expected long-term reward, resulting in it curating a robust policy for action selection.

In Tabular Q-learning, the program maintains a table of the expected reward for each possible action in a given state. This table is called the Q-table. The process commences with the allocation of stochastic values to the Q-table, following which, the software initiates its interaction with the environment.

The program initiates an action within a designed and defined state, which subsequently invokes the environment to impart a consequential response in the form of either a gain or detriment termed as reward or penalty. In response to this deterministic reward-penalty mechanism, the Q-table undergoes updating based on both incremental rewards received and expected rewards associated with each subsequent state encountered. Continuation of this refined process is iterated until exhaustive exploration by the program uncovers all possible states available for consideration; consequently allowing the said program to have acquired optimal knowledge regarding the best courses of action for each individualized state domain under scrutiny.

One of the salient benefits attributable to Tabular Q-learning is its independence from requiring any prior knowledge regarding the environment at hand for its operation. Rather, it obtains a comprehensive understanding through exploring said surrounding under observation and maintaining regular updates on its Q-table according to personal experience. Nonetheless, this proceeds with a caveat that necessitates scrutiny, in that said program ought to scrutinize each and every plausible state-action pairing within discovery, so as to ascertain that the optimum policy has been acquired exquisitely. Withal, such a task tends toward being particularly computationally arduous regardless of whether or not one is dealing with an intricate world model.

Despite its inherent restrictions, the utilization of Tabular Q-learning possesses tremendous potential as a formidable mechanism for instructing artificial intelligence networks in negotiating intricate and multifaceted environments. This technique has been extensively employed across an extensive array of operations ranging from gaming simulations to mechanical automation.

## III. Explanation of the Code

### *A. Tic-Tac-Toe*

*1) MiniMax with Alpha Beta Pruning:* The max_alpha_beta function comprises the intellectual capacity of the computer player in its turn and endeavours to optimize the numerical evaluation of the ongoing game's status. It employs the Alpha-Beta Pruning algorithm to effectively curtail and abbreviate an exorbitant number of game states, thereby rendering them inconsequential for both operational efficiency and our practical purposes. The initial maximum value is set at a nadir point equaling -2; from this starting position, it examines each vacant cell on the board while computing what score will result if it were to make a move that occupies any given square. To further streamline the calculations required, min_alpha_beta is then utilized so as to cull what minimum value can be attained by an opposing player (X́´) for each respective state or configuration reached through successive iterations thus far encountered. If after executing these procedures, comparison results reveal that there exists within possible outcomes some viable chance still remaining such that hitherto unexplored options might prove advantageous towards augmenting score tallies significantly, we must quickly respond in kind with adaptability representative of clear-minded ingenuity aforethought. At times like these when information flows unrestrained as ripe harvest plucked from fields most fecund under nobler auspices whereupon all best-laid plans come ripening forthwith–these are moments when swift analysis necessitates coordinated execution par excellence themselves manifesting in manifold ways upon human consciousnesses wracked and rent wide apart into ever-proliferating horizons expanding beyond mere imagination before retreating once again unto shadows beckoning us back hither solitary contemplations hauntingly evocative yet infinitely elusive as moonbeams lost amid darkling skies bereft starlight dimmed by clouds scudding westwards indifferent their advance.

The min_alpha_beta function, denoting the turn of the human player, is geared at minimizing the score assigned to the current state of play in a fashion that mirrors that of max_alpha_beta. It aims to determine the minimum value this state could accrue by initially setting its base value at 2 and examining all available spots left unoccupied on the game board one after another. At each stage, it probes what would become of gameplay if an X́ẃere placed in said spot before calling upon max_alpha_beta which determines how far out an optimum move can be computed for 'O'. Subsequently, any output from max_alpha_beta which falls short of our contracted minimum rated score so far necessitates min_alpha_beta updating both such minimal indicator statistics and coefficients indicating where the next best effort should reside in order for further successful moves to be secured by human players. By contrast, however, should a path thus identified fail abjectly trailing behind previously set alpha values; loops will exit prematurely knowing that anything forthcoming will not bear fruit hereon based on original recommendation parameters made earlier as tests suggest. Should such subsequent outcomes fall below revised beta metrics somehow fares better than those initial suggestions generating corresponding feedback accordingly.

The play function is the main function that controls the game. It first draws the board and then checks if the game has ended (i.e., there is a winner or a tie). If the game has ended, it prints the result and initializes the game for the next round. If it is the human player's turn, it prompts the user to enter the coordinates of their move. If it is the computer player's turn, it calls min_alpha_beta to find the optimal move and sets the value of the cell to 'O'. Finally, it switches the turn to the human player.

*2) Tabular Q Learning:* The Agent class is quite the overlord when it comes to managing the learning process. It is the one in charge of initializing the Q class, which represents the Q-values utilized in the Q-learning algorithm. When the Agent class gets created, it sets the value of eps to 1.0, which ultimately decides the probability of choosing a random action during training.

The get_action method of the Agent class selects an action based on the current game state and the valid actions that can be taken. In the case that the probability of taking a random action is greater than a randomly generated value, it chooses an action at random. If not, it chooses the action with the highest Q-value by calling upon the get_best_action method of the Q class. If there are no Q-values for the current game state, it defaults to a random action.

The learn_one_game method of the Agent class plays one game of Tic Tac Toe and updates the Q-values based on the results. It starts by initializing a new game and repeatedly selects actions using the get_action method. After each action, it updates the Q-values based on the reward received. If the game ends in a win or draw, it breaks out of the loop.

The learning method of the Agent class is responsible for training the agent. It plays a specified number of games and gradually reduces the value of eps after each game to reduce the probability of taking random actions as the agent learns.

The Q class defines the Q-values used in the Q-learning algorithm. Its update method updates the Q-value for a given state-action pair based on the observed reward and the maximum Q-value for the next state. Its get_best_action method returns the action with the highest Q-value for a given state.

The redefine function is used to convert the state of the game, which is represented as a JSON string, into a list of lists that represents the game board. The play function tests the agent's performance against a default player. It initializes a new game and repeatedly selects actions using the agent's Q-values and the default player's strategy. At the end of the game, it prints the result of the game.

### *B. Connect Four*

*1) MiniMax with Alpha Beta Pruning:* The minimax function takes in the current board state, search depth, alpha and beta values for alpha-beta pruning, and a boolean indicating if it's the AI's or player's turn. It checks if the game is over or if the depth is zero and returns the score. Otherwise, it calls itself recursively with the boolean flipped. If it's the AI's turn, it searches for the best move by iterating through valid locations and calling minimax on the resulting board state. It returns the column with the highest score. Alpha-beta pruning speeds up the search. If it's the player's turn, it searches for the move that minimizes the AI's chances of winning. The play_game function creates a new board and alternates between the player and AI. For the player's turn, it checks for a winning move, then a blocking move then chooses a random valid move. For the AI's turn, it calls minimax to determine the best move and drops a piece in that column. It checks for a winning move after each turn and prints the board. When a winning move is found, it prints the winner and ends the game.

*2) Tabular Q Learning:* The class ComputerPlayer is a subclass of the Player class and has several properties such as piece, num_episodes, max_steps_per_episode, learning_rate, discount_rate, exploration_rate, rewards_all_episodes, and q_table.

The q_table is a dictionary that stores the expected reward values for each state-action pair. The chooseAction() method chooses an action based on the epsilon-greedy strategy, which involves exploring with probability exploration_rate and exploiting with probability (1-exploration_rate). The disableaction() method is used to disable an invalid action for a certain state and prompt the selection of another action.

The fetchQ() method finds a value in the q_table or initializes a new state to 0.0 if not already in the dictionary. The updateQ() method updates the q_table with the reward obtained from the current action, and the maximum expected reward for the next state.

Finally, the actual_start_game() method prints the current board state and allows the computer player to make a move based on its learned policy. If the selected move is valid, it is dropped onto the board, and the updateQ() method is called to update the q_table. If the move is not valid, the disableaction() method is called to disable the invalid action and prompt the selection of another action.

## IV. Performance Analysis

### *A. **Tic-Tac-Toe***

***1) Playing Against Default Opponent:*** In the following figures, the **Player is signified as the AI Bot following the Minimax algorithm with alpha-beta pruning** for the tic-tac-toe game. Figure 1 visualizes the outcomes of the 1000 games played between the default opponent and the Minimax with

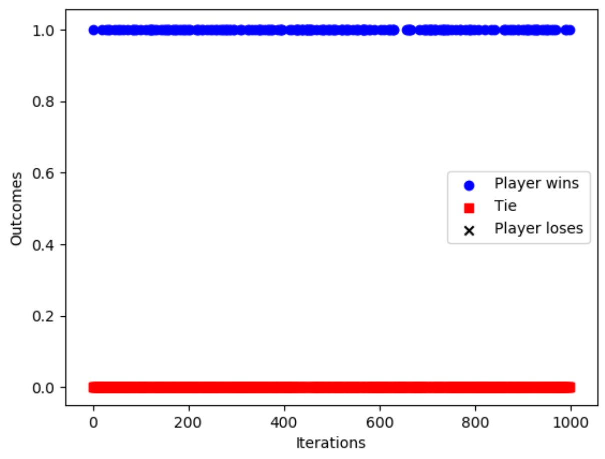

Fig. 1. Scatter plot of the outcomes of 1000 games of Minimax Algorithm against Default Opponent for Tic Tac Toe Game

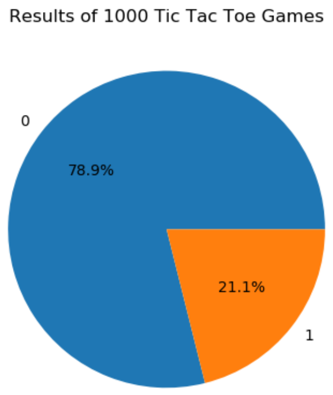

Fig. 3. Pie plot of the outcomes of 1000 games of Minimax Algorithm against Default Opponent for Tic Tac Toe Game

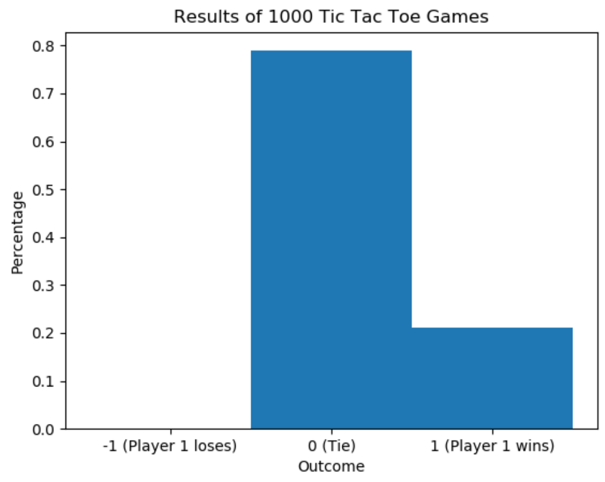

Fig. 2. Histogram plot of the outcomes of 1000 games of Minimax Algorithm against Default Opponent for Tic Tac Toe Game

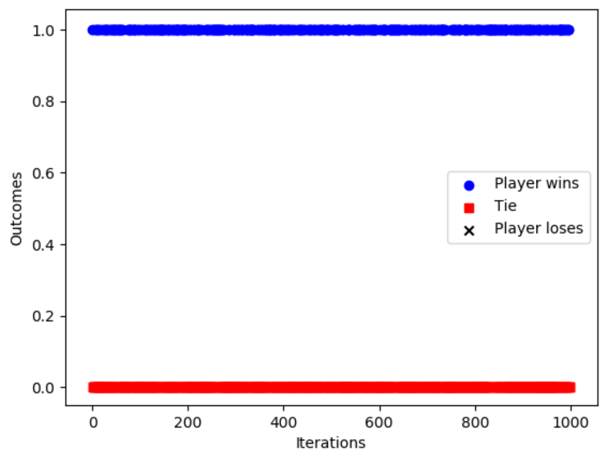

Fig. 4. Scatter plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Default Opponent for Tic Tac Toe Game

the alpha-beta pruning AI bot. X-axis signifies the iteration number. Figure 2 visualizes the frequency of the outcomes of the 1000 games between the AI bot and the Default Opponent as a histogram plot. Figure 3 visualizes the percentage values of the outcomes of the 1000 Tic Tac Toe games as a pie chart. In all of the above plots, '1' signifies the AI Bot's favourable outcome(win).

I tried to increase the number of iterations and even then, the AI bot never lost a game.

In the following figures, the **Player is signified as the AI Bot following the Tabular Q Learning algorithm** for the tic-tac-toe game. Figure 1 visualizes the outcomes of the 1000 games played between the default opponent and the Tabular Q Learning AI bot. The iteration number is on the x-axis. Figure 2 visualizes the frequency of the outcomes of the 1000 games between the AI bot and the Default Opponent as a histogram plot. Figure 3 visualizes the percentage values of the outcomes of the 1000 Tic Tac Toe games as a pie chart. In all of the above plots, '1' signifies the AI Bot's favourable outcome(win).

I tried to increase the number of iterations and even then, the AI bot never lost a game.

*2) **Playing Against Each Other:*** In the following figures, the **Player is signified as the AI Bot following the Minimax algorithm with alpha-beta pruning** for the tic-tac-toe game. These games are played between the Minimax algorithm with alpha-beta pruning and Tabular Q Learning AI bots. Figure 7 visualizes the outcomes of the 1000 games played between the Minimax algorithm with alpha-beta pruning and Tabular Q Learning AI bots. X-axis signifies the number of

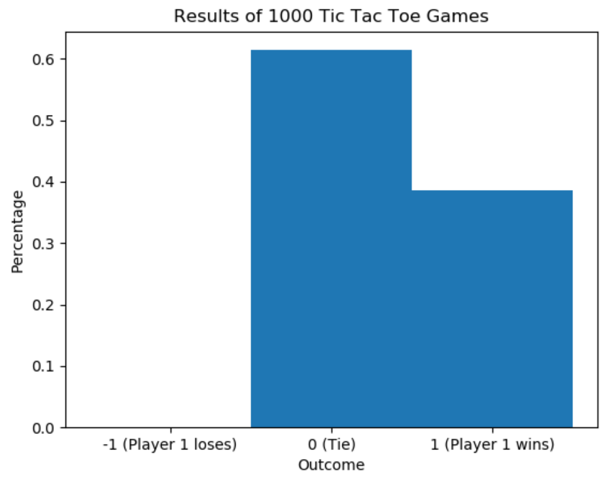

Fig. 5. Histogram plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Default Opponent for Tic Tac Toe Game

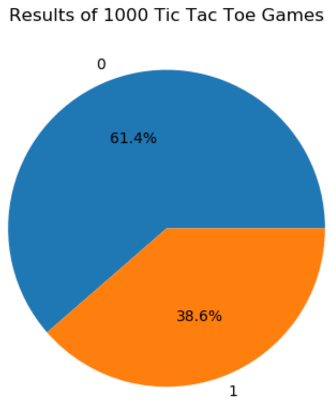

Fig. 6. Pie plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Default Opponent for Tic Tac Toe Game

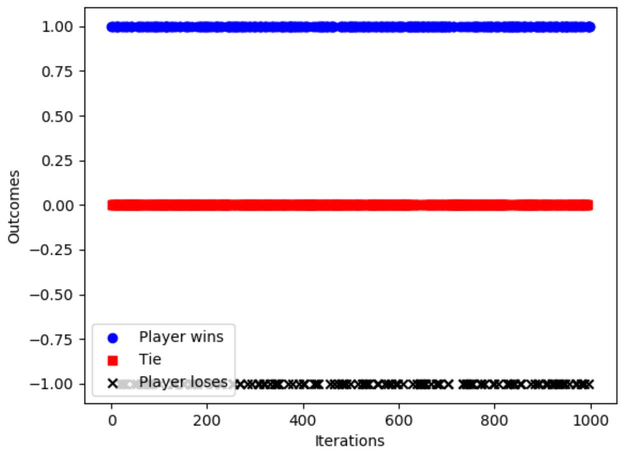

Fig. 7. Scatter plot of the outcomes of 1000 games of Minimax Algorithm against Tabular Q Learning Algorithm for Tic Tac Toe Game

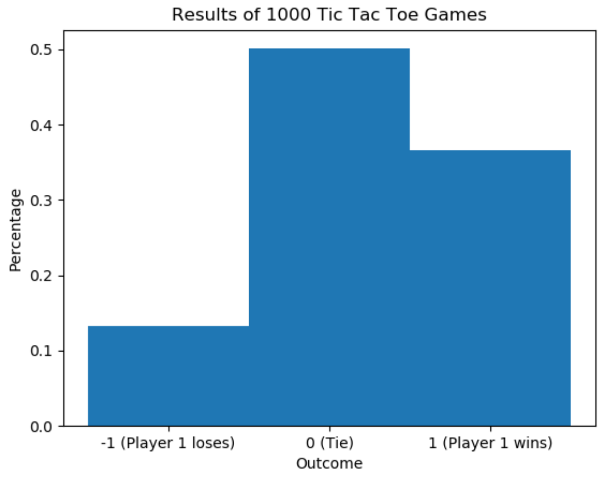

Fig. 8. Histogram plot of the outcomes of 1000 games of Minimax Algorithm against Tabular Q Learning Algorithm for Tic Tac Toe Game

iterations. Figure 8 visualizes the frequency of the outcomes of the 1000 games as a histogram plot between the AI bots. Figure 9 visualizes the percentage values of the outcomes for 1000 games between the same as a pie chart. In all of the above plots(Fig7-9), '1' signifies the AI Bot's(Minimax algorithm with alpha-beta pruning) favourable outcome(win). After numerous attempts(of 1000 games played together), I was able to find a plot where Tabular Q Learning won a match. Usually, the Minimax algorithm either ties or wins.

### *B. Connect Four*

*1) **Playing Against Default Opponent:*** In the following figures, the **Player is signified as the AI Bot following the Minimax algorithm with alpha-beta pruning** for the connect four game. Figure 10 visualizes the outcomes of the 1000 games played between the default opponent and the Minimax with the alpha-beta pruning bot. X-axis signifies the number of Figure 11 visualizes the frequency of the outcomes of the 1000 games between the AI bot and the Default Opponent. Figure 12 visualizes the percentage values of the outcomes. In all of the above plots, '1' signifies the AI Bot's favourable outcome(win).

In the following figures, the **Player is signified as the AI Bot following the Tabular Q Learning algorithm** for the connect four game. Figure 13 visualizes the outcomes of the 1000 games played between the default opponent and the AI bot. X-axis signifies the number of Figure 14 visualizes the frequency of the outcomes of the 1000 games between the AI bot and the Default Opponent. Figure 15 visualizes the percentage values of the outcomes. In all of the above plots, '1' signifies the AI Bot's favourable outcome(win).

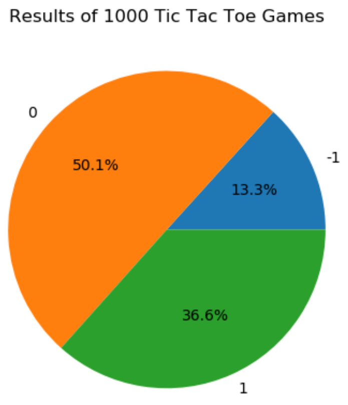

Fig. 9. Pie plot of the outcomes of 1000 games of Minimax Algorithm against Tabular Q Learning Algorithm for Tic Tac Toe Game

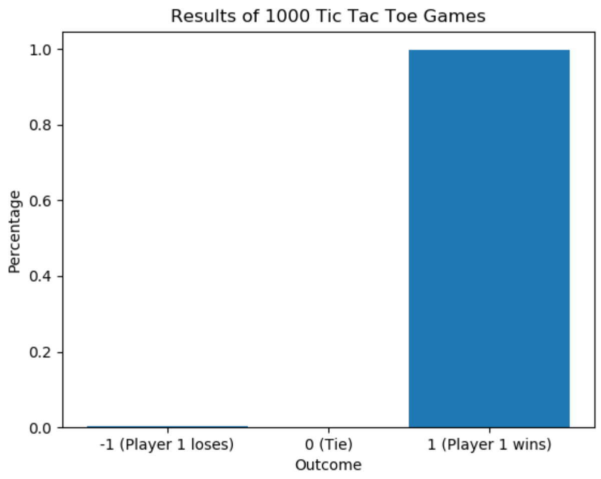

Fig. 11. Histogram plot of the outcomes of 1000 games of Minimax Algorithm against Default Opponent for Connect Four Game

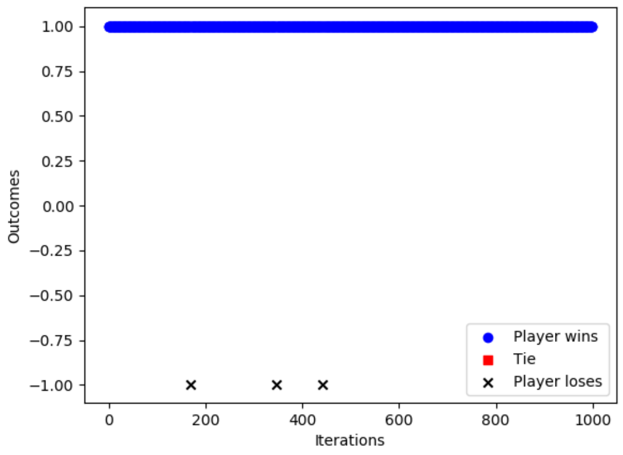

Fig. 10. Scatter plot of the outcomes of 1000 games of Minimax Algorithm against Default Opponent for Connect Four Game

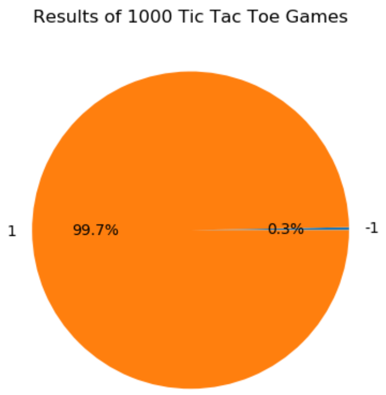

Fig. 12. Pie plot of the outcomes of 1000 games of Minimax Algorithm against Default Opponent for Connect Four Game

*2) **Playing Against Each Other:*** In the following figures, the **Player is signified as the AI Bot following the Tabular Q Learning algorithm** for the connect four game. These games are played between the Tabular Q Learning algorithm and the Minimax algorithm with alpha-beta pruning AI bots.

Figure 16 visualizes the outcomes of the 1000 games played between the AI bots. X-axis signifies the number of iterations. Figure 17 visualizes the frequency of the outcomes of the 1000 games as a histogram plot between the AI bots. Figure 18 visualizes the percentage values of the outcomes for 1000 games between the same as a pie chart. In all of the above plots(Fig7-9), '1' signifies the AI Bot's(Tabular Q Learning) favourable outcome(win).

## V. Observation

### *A. **How do your algorithms compare to each other when playing against default opponent overall***

*1) Tic-Tac-Toe:* Against the default opponent, for the tic-tac-toe game, the Tabular Q Learning algorithm fares much better than the Minimax algorithm with alpha-beta pruning as the percentage of wins for the Tabular Q Learning algorithm is much higher than the Minimax with alpha-beta pruning algorithm. Also, the time taken for the Tabular Q Learning algorithm including the training time was much lesser when compared to the Minimax algorithm with alpha-beta pruning.

*2) Connect Four:* Against the default opponent, for the connect four game, the Minimax algorithm with alpha-beta pruning fares much better as the win percentage of the Minimax algorithm is significantly higher when compared with the

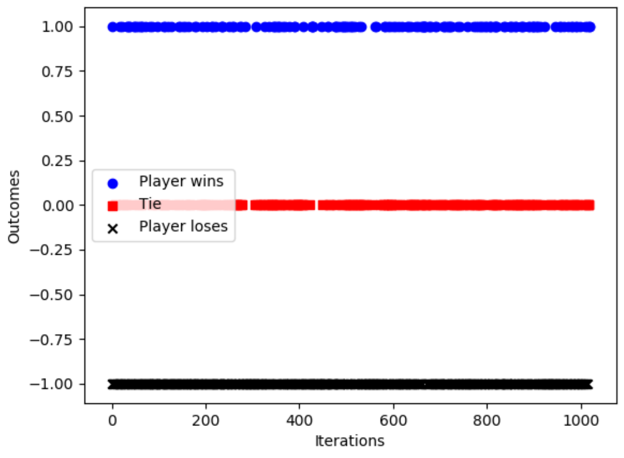

Fig. 13. Scatter plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Default Opponent for Connect Four Game

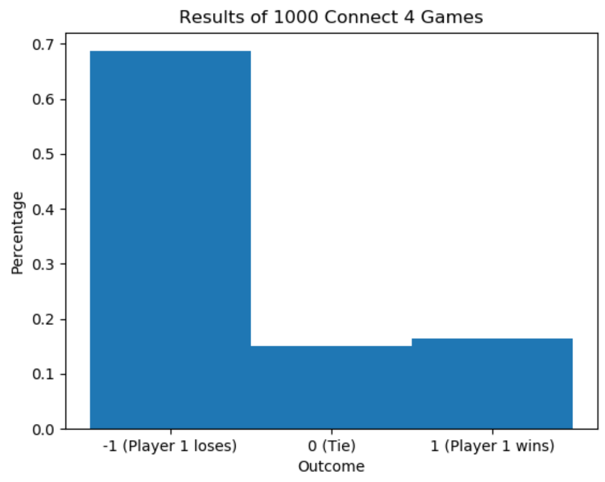

Fig. 14. Histogram plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Default Opponent for Connect Four Game

Tabular Q Learning algorithm. The Minimax algorithm almost always wins the game against the Default Opponent.

### *B. How do your algorithms compare to each other when playing against each other overall*

*1) Tic Tac Toe:* Overall, the Minimax Algorithm with alpha-beta pruning is a better choice as the win percentage is significantly more when compared to the Tabular Q Learning algorithm. Even though against the default opponent for the Tic Tac Toe game, Tabular Q Learning has a higher win rate as the number of possible states is lesser when compared to Connect Four, but when pitted against each other, the Minimax algorithm with alpha-beta pruning is significantly superior. There were numerous 1000 sets of games where the Tabular Q Learning algorithm did not even win a single game whereas the Minimax algorithm won quite a lot. It is observed that Tabular Q Learning seldom wins. This is logical as even though it did train for 100000 games, all the possible states were not covered in those games. For a Tic Tac Toe game, to learn all the possible states, it should play at least 177147 games. Due to system restrictions, it was not possible. But, it is believed that in doing so, it is possible that Tabular Q Learning would fare better. But still, the Minimax algorithm would be far better than Tabular Q Learning in any case. The time consumed is significantly less for the Minimax algorithm with alpha-beta pruning when compared to Tabular Q Learning. Thus, Minimax is a better choice for Tic Tac Toe Game.

*2) Connect Four:* Overall, the Minimax Algorithm with alpha-beta pruning performs way better than the Tabular Q Learning algorithm for connect four game. It is understandable as for the Q Learning algorithm to fare better, it needs to have

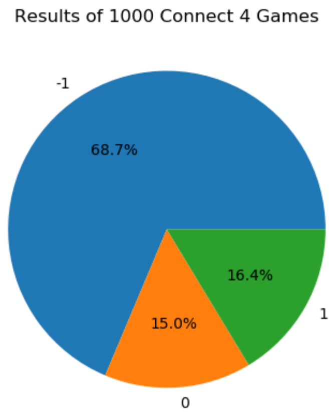

Fig. 15. Pie plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Default Opponent for Connect Four Game

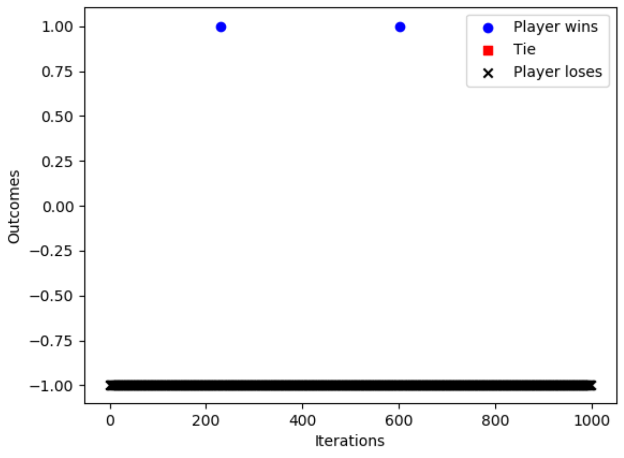

Fig. 16. Scatter plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Minimax Algorithm with alpha-beta pruning for Connect Four Game

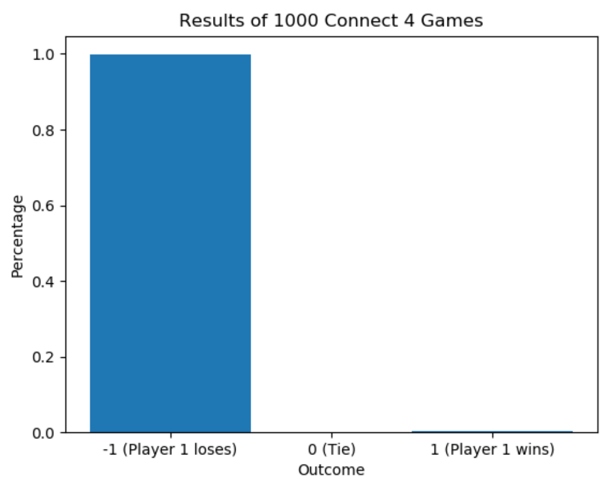

Fig. 17. Histogram plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Minimax Algorithm with alpha-beta pruning for Connect Four Game

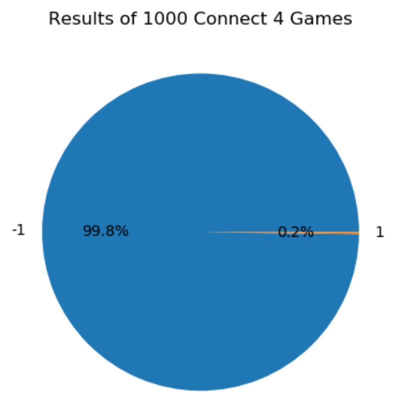

Fig. 18. Pie plot of the outcomes of 1000 games of Tabular Q Learning Algorithm against Minimax Algorithm with alpha-beta pruning for Connect Four Game

all the possible states of the game in its q table which is around 84 billion. I trained the model for only a meagre 100000 games due to system restrictions. Also, the time consumed for the Q Learning Algorithm to train is directly proportional to the number of games to be played. Hence, the time complexity of the Q Learning Algorithm is significantly more. Which makes it a bad choice. With the current number of training games played, the Tabular Q Learning algorithm rarely won any games against the Minimax algorithm with alpha-beta pruning.

## VI. Analysis

From the above plots, it is evident that the Tabular Q Learning algorithm performs decently for a game which has a relatively less number of game states. When the number of states in a game increases, the number of games required for the algorithm to learn all the states also increases sharply and this increases the time complexity of the algorithm.

The Minimax algorithm with alpha-beta pruning is a better choice when compared to the Tabular Q Learning algorithm as it has a far better win percentage and definitely takes lesser time to execute.

Overall, the Tabular Q performs well for the Tic Tac Toe game against the Default Opponent as the number of possible game states is less. But, for Connect Four game the number of possible game states and a number of tunable parameters is significantly more, hence it's hard to find a suitable set of hyper-parameters along with the limitations of the system hardware. Minimax with alpha-beta pruning, performs greatly for both of the games - against each other as well as against the default opponent.

## References

[1] https://medium.com/analytics-vidhya/i-programmed-a-tic-tac-toe-game-with-minimax-algorithm-but-how-does-it-work-8c689842c9ee
[2] https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f
[3] https://medium.com/@carsten.friedrich/part-3-tabular-q-learning-a-tic-tac-toe-player-that-gets-better-and-better-fa4da4b0892a
[4] http://romain.raveaux.free.fr/document/ReinforcementLearningby QLearningConnectFourGame.html
[5] https://medium.com/@carsten.friedrich/part-2-the-min-max-algorithm-ae1489509660