CMPE 260 - Reinforcement Learning

Connect Four

Team Members

Abhishek Bais

Haley Feng

Princy Joy

Shannon Phu

Table of Contents

Executive Summary

Connect Four is a popular two player game in which players select different colored circular pieces called coins and take turns to drop them in a 6x7 grid. The first player to place four of their pieces consecutively in a row, column, or a diagonal line wins the game.

Computer agents can learn to play the game using reinforcement learning techniques and compete against human opponents. In our project we aim to simulate a real-life gaming experience for a human player against trained computer agents. To do so, we create five different kinds of computer agents that can play the game. These agents are:

1. Random Move Agent
2. MiniMax Agent
3. Monte Carlo Agent
4. Q Learner Agent
5. Sarsa Learner Agent

We designate the Random Move Agent and the MiniMax Agent as baseline agents. While the former agent responds to a move by a human player with a random move the latter responds to a move by a human player with a non-reinforcement learning, backtracking guided move that gives the maximum immediate gain. The other three agents, namely the Monte Carlo Agent, Q Learner Agent and the Sarsa Learner Agent use reinforcement learning techniques to make moves.

In our project, we train the reinforcement learning guided computer agents by battling them against the Random Move Agent and the Minimax Agent over "N" games or iterations. We also battle the different reinforcement learning guided computer agents against each other to compare and contrast their performance.

To visualize the game in action, we implemented a graphic interface with a virtual Connect Four board and coins.

Background / Introduction

Connect Four is a popular two player game played in many countries around the world. In this game each player selects a different colored circular piece called coin. Players take alternate turns to drop their coins in a 6x7 grid called the Connect Four board. The first player to form a 4-in-a-row connection or place four of their pieces consecutively in a row, column, or diagonal line wins the game.

Board games have been a topic of interest in the machine learning community for quite some time now. Board games such as Chess, Go and Tic-tac-toe have gained particular attention. Computer agents have been developed to play them. These computer agents have used different algorithms such as Minimax, Monte Carlo, Q Learning, Sarsa Learning and Tabular Q Learning as building blocks.

For example, Deep Blue is a chess-playing supercomputer that uses custom VLSI chips to execute the alpha-beta search algorithm that seeks to decrease the number of nodes evaluated by the minimax algorithm in its search tree. Research in building it started with the ChipTest Project at Carnegie Mellon University in 1985. Released in 1996, it became the first supercomputer to win both a chess game and a chess match against a reigning world champion under regular time controls. The system derived its playing strength mainly from its computing power. It used a massively parallel RS/6000 SP Thin P2SC-based system with 30 nodes, each node containing a 120 MHz P2SC microprocessor enhanced with 480 special purpose VLSI chess chips. Its chess playing program was written in C and ran under the AIX operating system. It was capable of evaluating 200 million positions per second, twice as fast as the 1996 version [1].

AlphaGo is a computer program that plays the 19x19 grid board game Go and uses the Monte Carlo tree search algorithm to find its moves based on knowledge previously acquired via machine learning. First developed by DeepMind Technologies, a subsidiary of Google (now Alphabet Inc.) and released in 2016, it is a completely self-taught program that was initially trained to mimic human play by attempting to match the moves of expert players from recorded historical games, using a database of around 30 million moves [2].

Tic-tac-toe, popularly known as noughts and crosses, or Xs and Os is a paper-and-pencil/ board game for two players who take turns marking the spaces in a 3x3 grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins. Reinforcement learning techniques such as Q Learning, Sarsa Learning and Tabular Q Learning have been used to create computer programs to play the game [3].

Problem Statement

The goal of this project is to provide a real-life Connect Four gaming experience for a human player against a computer agent. To do so, the computer agent must learn to play from the player's prior moves using reinforcement learning techniques fairly well. It is desired that the gaming experience be fun and challenging. Therefore, it is important that the computer agent learns to:

1. Predict and block the user's next move

2. Make intelligent first and response moves
3. Make moves quickly to reduce wait times

It is also desired that reinforcement learning guided computer agents be implemented using multiple algorithms and their performance measured in terms of win-rate and efficiency defined as average time to play the game. It is also desired that their performance be compared and contrasted against a computer agent that makes random moves and against a computer agent that learns to make moves using a non-reinforcement learning backtracking technique - "MiniMax" which formed the backbone of the Deep Blue supercomputer, to pick the best computer agent to play the game.

To accomplish these goals five different styles of computer agents should be implemented. These are:

1. Random Move Agent
2. MiniMax Agent
3. Monte Carlo Agent
4. Q Learner Agent
5. Sarsa Learner Agent

The Random Move Agent and the Minimax Agent should be designated as baseline agents, while the Monte Carlo Agent, Q Learner Agent and the Sarsa Learner Agent be designated as the reinforcement learning agents used for comparison against baseline agents.

In addition, to visualize the game in action, a graphic interface with a virtual Connect Four board and coins must also be implemented.

Purpose / Motivation

There are two primary drivers for this project. These are:

1. Simulate a real-life gaming experience for a human player against a computer agent that uses reinforcement learning to learn to predict and block the user's next move, make intelligent first and response moves, and move quickly in response to a human player.

2. To compare and contrast the performance of different reinforcement learning guided computer agents measured in terms of win-rate and efficiency i.e. average time to play the game. The performance should be computed via battles against a computer agent that makes random moves and via battles against a computer agent that learns to make moves using a non-reinforcement learning backtracking

technique - “MiniMax” which formed the backbone of the Deep Blue supercomputer.

Differentiator / Contribution

The project schema has two key differentiators. These are:

1. Unlike the Deep Blue supercomputer which built a computer agent to play chess using the Minimax algorithm, a non-reinforcement learning guided algorithm, this project will also build computer agents using reinforcement learning algorithms. This will help facilitate faster learning with fewer move evaluations per second and utilize fewer compute resources. Recall that the Deep Blue supercomputer used a massively parallel RS/6000 SP Thin P2SC-based system with 30 nodes, each node containing a 120 MHz P2SC microprocessor enhanced with 480 special purpose VLSI chess chips. Its chess playing program was written in C and ran under the AIX operating system. It was capable of evaluating 200 million positions per second.

   That a reinforcement learning guided computer agent requires fewer moves can be demonstrated using a Tic-tac-toe game example. Tic-tac-toe is a 3x3 grid game. Therefore, a non-reinforcement learning guided algorithm that wishes to learn all possible games to compete and win against a human player consistently would need to learn 9! Or 9*8*7*6*5*4*3*2*1 = 9! = 362880 games. Although this number is bound, these are still a lot of games to learn. A reinforcement learning guided agent could achieve the same degree of competitiveness in much fewer games. This is because the
    1. The number of games ending on the 5th move: 1440
    2. The number of games ending on the 6th move: 5328
    3. The number of games ending on the 7th move: 47952
    4. The number of games ending on the 8th move: 72576
    5. The number of games ending on the 9th move: 127872

   This gives a total of 255168 possible games, or 29.68% fewer games required by the reinforcement learning guided agent to learn to play the game effectively [4]

2. Unlike AlphaGo which builds a computer agent using the Monte Carlo Reinforcement learning algorithm, this project will evaluate other reinforcement learning algorithms like Q Learner and Sarsa Learner in addition to the Monte Carlo algorithm to identify the best performing agent in terms of win-rate and time to play the game. Recall that for Go, a 19x19 grid board game, AlphaGo at all times maintained a database of around 30 million moves. One reason for that is that the

Monte Carlo algorithm is a tree search algorithm and for best results requires extensive exploration.

Also, although computer agents have been developed using the Q Learner and Sarsa Learner algorithms for the Tic-tac-toe game, it has a significantly smaller 3x3 state space while Connect Four has a larger 6x7 state space.

Methodology

Game Setup

We set up the game to be played in three modes.

1. Single Player Mode
   In this mode, a human player can play the game against a computer agent. This agent can be any one of the five agents - Random Move, MiniMax, Monte Carlo, Q Learner or Sarsa Learner. The project design supports the option for a human player to select the computer agent it wishes to play the game against.

2. Two Player Mode
   In this mode, two human players can play against each other if so desired and should the game be hosted on a public website.

3. Training Mode
   In this mode, the set reinforcement learning guided computer agent battles against a Random Move Agent and against the Minimax Agent over “N” games to learn to play the game.

Agent Algorithms

We implemented five different computer agents to play the game. These agents were:

1. Random Move Agent
   This agent picks next moves randomly from available locations on the board.

2. MiniMax Agent
   This agent uses a non-reinforcement learning guided backtracking and recursion-based scheme commonly used in game theory to pick the next move to maximize the immediate gain.

3. Monte Carlo Agent
   This agent uses a reinforcement learning guided scheme to learn directly from game experiences without using any Markov Decision Process knowledge.

4. Q Learner Agent
   This agent uses a reinforcement learning guided off-policy value-based scheme based on the Bellman's equation to learn the value of the optimal policy regardless of action

5. Sarsa Learner Agent
   This agent uses a reinforcement learning guided on-policy value-based scheme to learn the value of the optimal policy based on the action derived from the current policy.

Hyperparameter Tuning Setup

To get best results, we performed hyper parameter tuning on the different reinforcement learning guided computer agents.

The Q Learner and the Sarsa Learner agents were first tuned for alpha/ learning rate. Then further tuned for gamma/ discount factor. The Table 1.0 captures the values of alpha and gamma for which the tuning experiments were performed.

The Monte Carlo agent was tuned for exploration coefficient. Monte Carlo is tree search algorithm, where the exploration coefficient controls the amount of search to perform.

1. Smaller values of the exploration coefficient leads to greater exploitation i.e. the visited nodes of the tree are more likely to be visited again.
2. Larger values of the exploration coefficient leads to greater exploration i.e. the previously unvisited nodes of the tree are more likely to be visited.

Algorithm	**Hyper Parameters Tuned**
Q Learner	alpha/ learning rate, tuning values [0.05, 0.25, 0.3, 0.75, 0.9] gamma/ discount factor, tuning values [0.25, 0.50, 0.75, 0.9, 0.98]
Sarsa Learner	alpha/ learning rate, tuning values [0.05, 0.25, 0.3, 0.75, 0.9] gamma/ discount factor, tuning values [0.25, 0.50, 0.75, 0.9, 0.98]
Monte Carlo	exploration coefficient, tuning values [0.8, 1, 1.4, 1.6]

Table 1.0

Performance Evaluation Criteria

To evaluate the performance of the different reinforcement learning guided computer agents we came up with two performance metrics. These were:

1. Win-Rate
   The Win-Rate is defined as the percentage of times a set reinforcement learning guided computer agent wins in N battles against its opponent.

2. Average Play Time
   The Average Play Time is defined as the total time taken for the reinforcement learning guided computer agent to win/ lose/ tie the game against its opponent.

Implementation & Results

Game Implementation

An object-oriented design scheme was employed to ensure the software is easy to maintain, objects are standalone entities, easy to re-use and extend in the future. For example should more reinforcement learning guided computer agents be required, it is easy to achieve this by extending existing classes and without changing existing code. This was achieved using the following scheme:

Global Variables

The Connect4_Globals.py file was created to define all the global variables used by the different game classes. Some examples global variables include:

1. Board size
2. Color of coins

Game Utilities

The Connect4_Utilities.py file was created to define the utility classes used by the game. These include:

Class	**Responsibility**
Slot	Implement a position on the connect four board
Board	Implement the 6x7 connect four grid
Coin	Implement a piece to play the game

SlotTrackerNode	Implement an internal node in the graph representation of the board
ColumnFullException	Implement an exception handler should coins drop in filled positions on the board

Table 2.0

Game Players

The Connect4_Players.py file was created to define the base players who can play the game. These include:

1. Human Player
2. Random Move Agent

The Connect4_RLPlayers.py file was created to define the computer agents who can play the game. These include:

1. MiniMax Agent
2. Monte Carlo Agent
3. Q Learner Agent
4. Sarsa Learner Agent

Game Logic

The Connect4_GameLogic.py file was created to define the game logic. These includes:

Game Condition	Details
Game winning condition	A 4-in-a-row connection (horizontal, vertical, or diagonal)
Game state	Captures the state of the game, checks whether the game is over?
Game outcome	Determines which player won the game

Game graphical Interface

The Connect4_GameView.py file was created to configure the game. It includes:

Game Unit	Description
Game menu	Displays the graphical interface
Game mode	Manages the game play mode (Single Player, Two Player or Training)
Game setup and play	Sets up the game between two agents and initiates the game

Table 3.0

Figure 1.0 shows the Game Interface and the game in action in different modes.

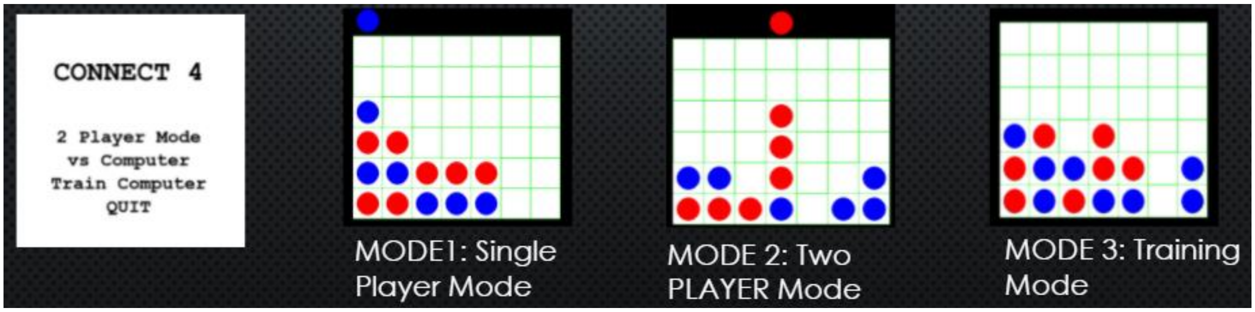


Figure 1.0

Game Invocation

The Connect4_Play.ipynb file was created as the single entry point in the system. It is used to set up the game with different players and initiate battles. The codes referenced to build the game system are listed in [5][6].

Sensitivity Analysis and Results

The sensitivity analysis was performed to understand the relationship between the hyper parameters tuned for the different reinforcement learning guided computer agents and the game win-rate.

Q Learner Sensitivity Analysis results

The Q Learner was first tuned for alpha/ learning rate. Then further tuned for gamma/ discount factor. The Table 1.0 captures the values of alpha and gamma for which the tuning experiments were performed.

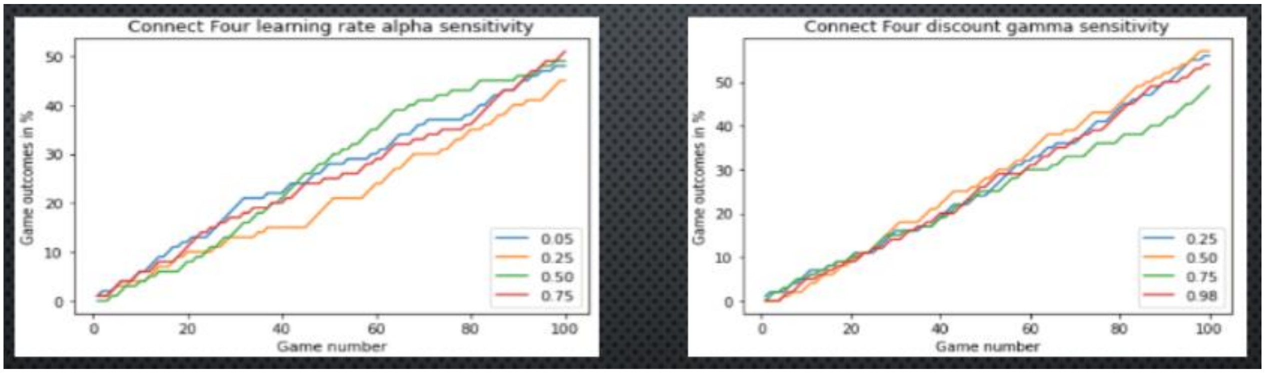
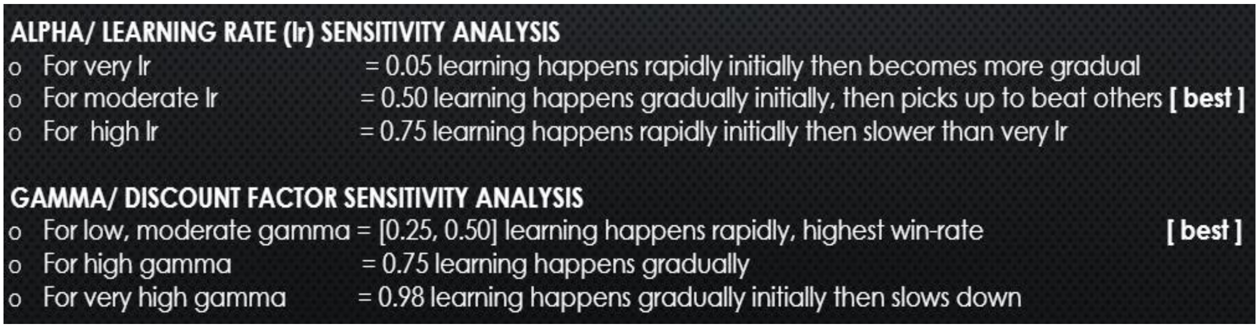


Figure 2.0

Sensitivity Analysis was performed for both the hyper parameters. Figure 2.0 captures the results of this analysis while Figure 3.0 captures the inferences drawn.



ALPHA/ LEARNING RATE (lr) SENSITIVITY ANALYSIS

- For very lr = 0.05 learning happens rapidly initially then becomes more gradual
- For moderate lr = 0.50 learning happens gradually initially, then picks up to beat others **[best]**
- For high lr = 0.75 learning happens rapidly initially then slower than very lr

GAMMA/ DISCOUNT FACTOR SENSITIVITY ANALYSIS

- For low, moderate gamma = [0.25, 0.50] learning happens rapidly, highest win-rate **[best]**
- For high gamma = 0.75 learning happens gradually
- For very high gamma = 0.98 learning happens gradually initially then slows down

Figure 3.0

Sarsa Learner Sensitivity Analysis results

The Sarsa Learner was first tuned for alpha/ learning rate. Then further tuned for gamma/ discount factor. The Table 1.0 captures the values of alpha and gamma for which the tuning experiments were performed.

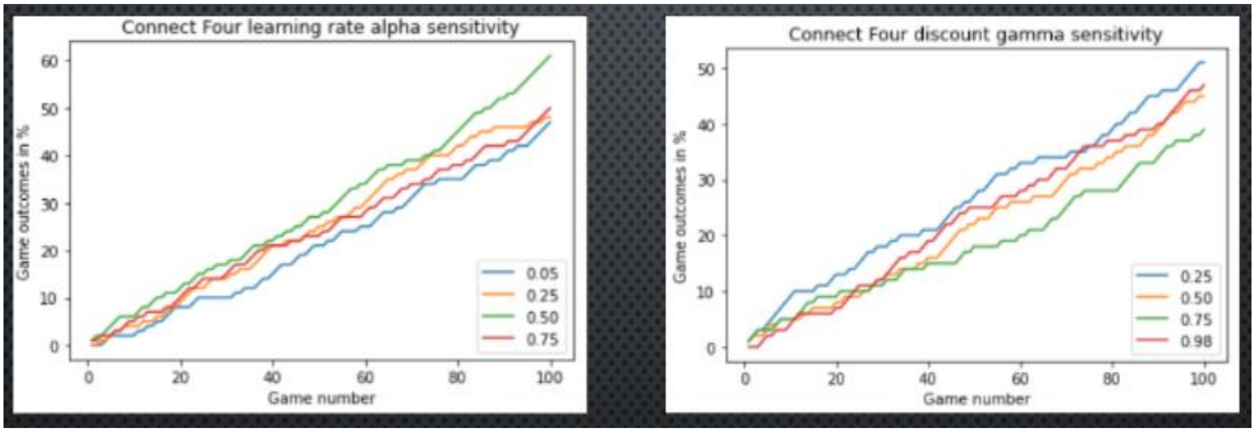
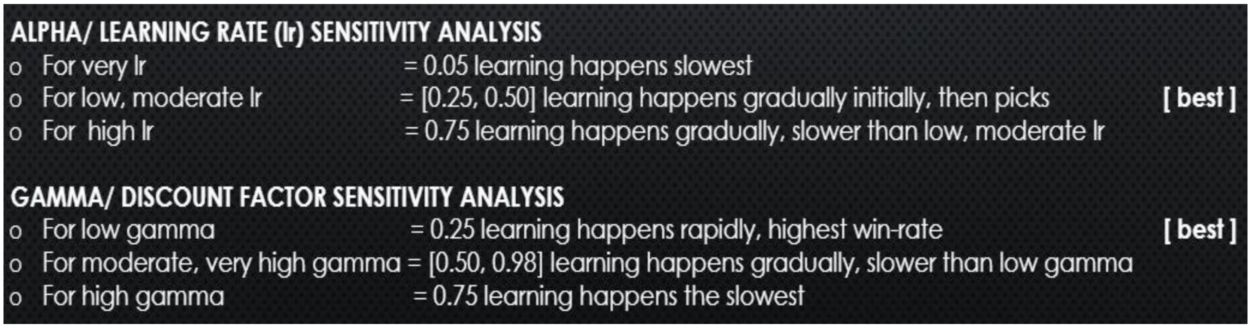


Figure 4.0

Sensitivity Analysis was performed for both the hyper parameters. Figure 4.0 captures the results of this analysis while Figure 5.0 captures the inferences drawn.



ALPHA/ LEARNING RATE (lr) SENSITIVITY ANALYSIS

- For very lr = 0.05 learning happens slowest
- For low, moderate lr = [0.25, 0.50] learning happens gradually initially, then picks **[best]**
- For high lr = 0.75 learning happens gradually, slower than low, moderate lr

GAMMA/ DISCOUNT FACTOR SENSITIVITY ANALYSIS

- For low gamma = 0.25 learning happens rapidly, highest win-rate **[best]**
- For moderate, very high gamma = [0.50, 0.98] learning happens gradually, slower than low gamma
- For high gamma = 0.75 learning happens the slowest

Figure 5.0

Monte Carlo Sensitivity Analysis results

The Monte Carlo agent was tuned for exploration coefficient. Sensitivity Analysis was performed for different values of the hyper parameter. Figure 6.0 captures the results of this analysis while Figure 7.0 captures the inferences drawn.

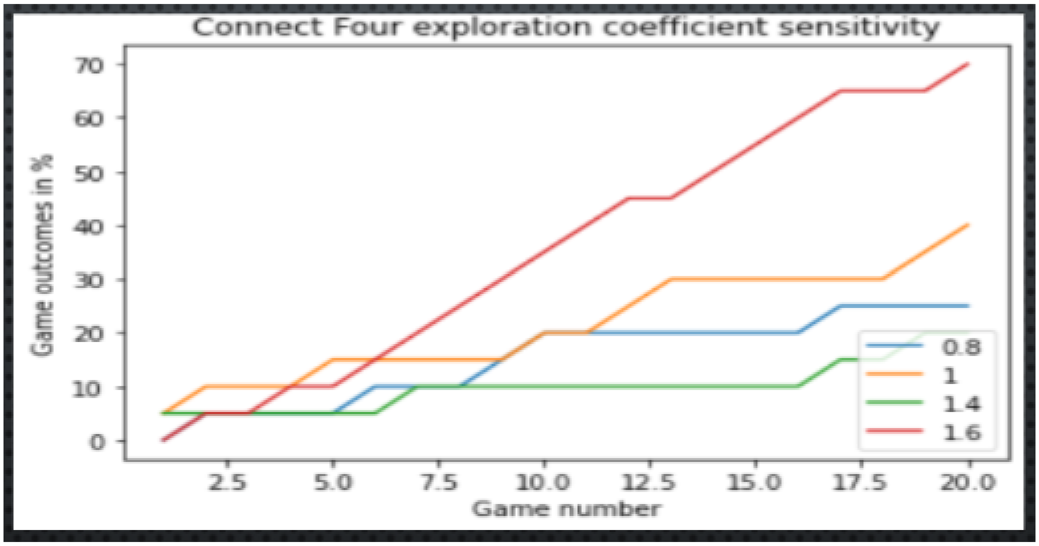
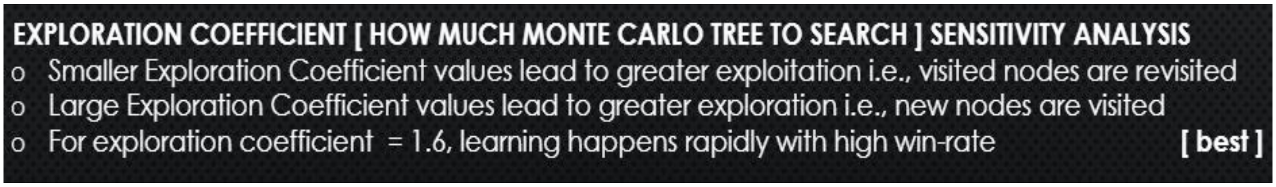


Figure 6.0



EXPLORATION COEFFICIENT [HOW MUCH MONTE CARLO TREE TO SEARCH] SENSITIVITY ANALYSIS

- Smaller Exploration Coefficient values lead to greater exploitation i.e., visited nodes are revisited
- Large Exploration Coefficient values lead to greater exploration i.e., new nodes are visited
- For exploration coefficient = 1.6, learning happens rapidly with high win-rate **[best]**

Figure 7.0

Battles between Agents and Results

The different computer agents were trained in battles against the Random Move computer agent. The results of those battles were captured for win-rate and efficiency i.e. average time to play the game. Figure 8.0 captures the results for the battles for the Q Learner and the Sarsa Learner with a Random Move Agent.

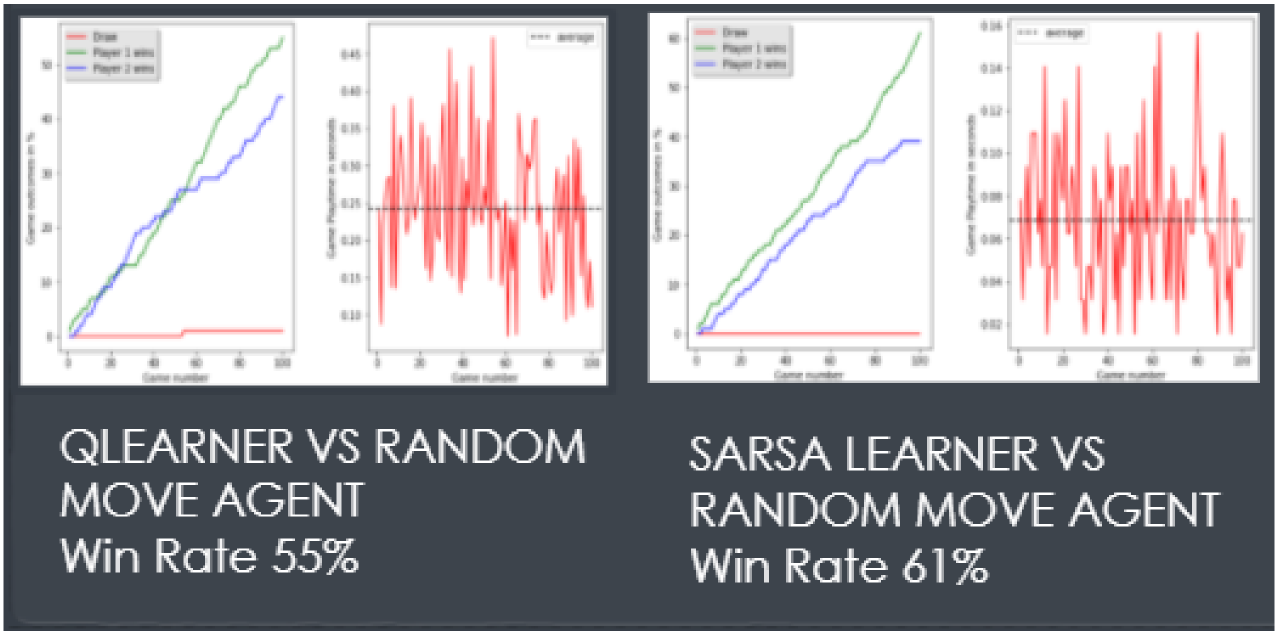
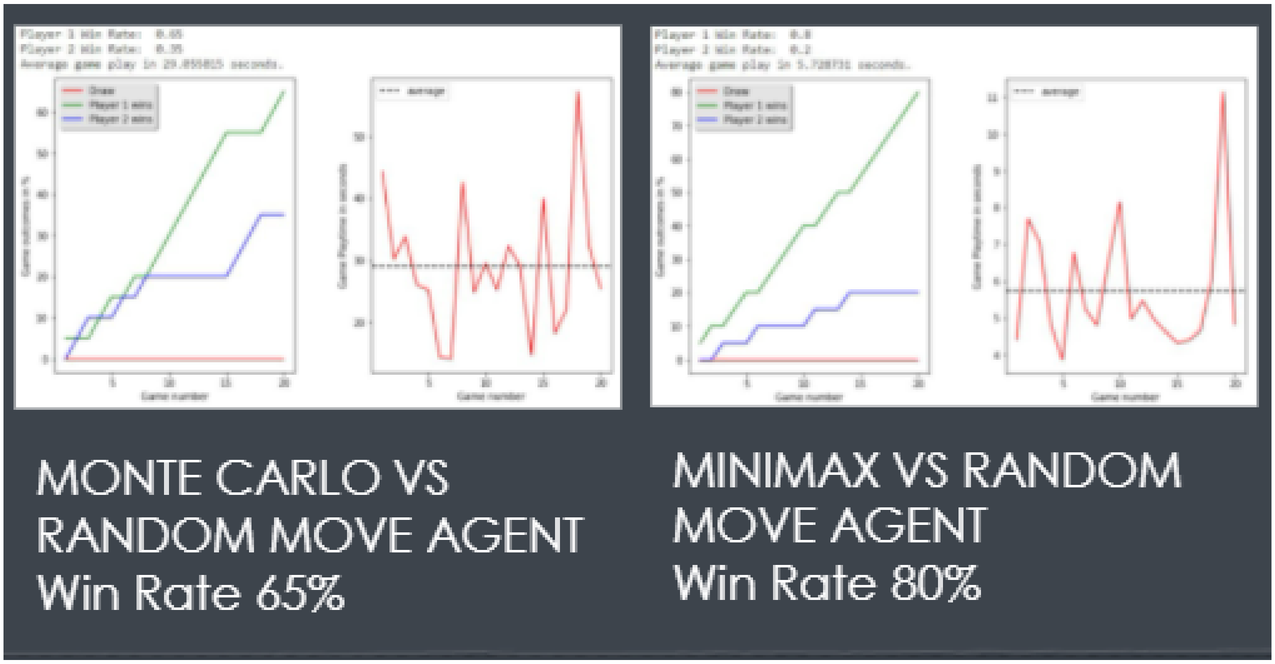


Figure 8.0



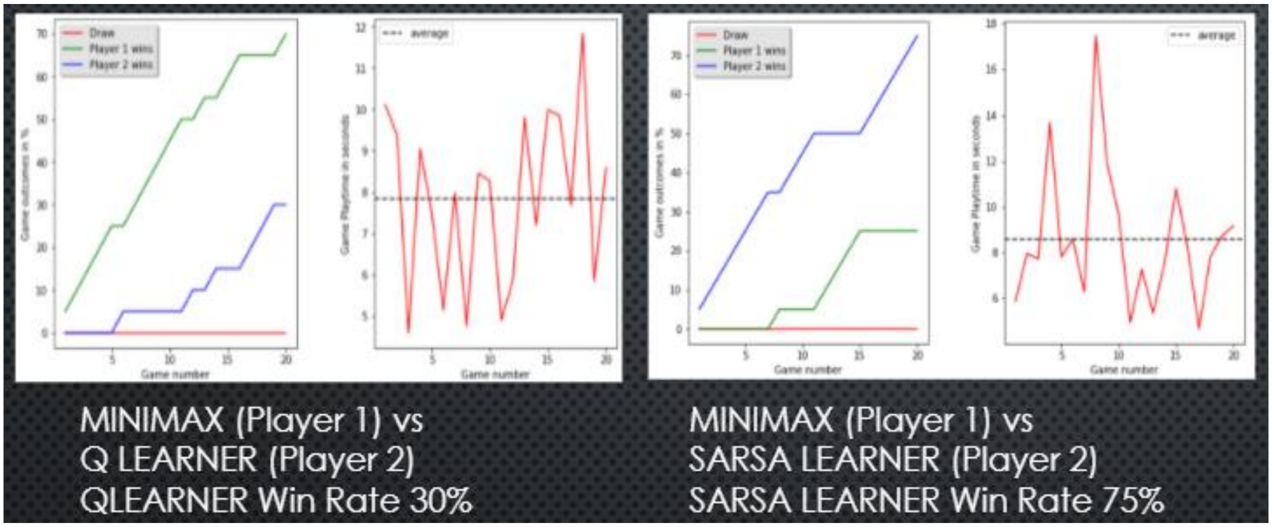
```
Player 1 Win Rate:  0.65
Player 2 Win Rate:  0.35
Average game play in 29.055815 seconds.
```

```
Player 1 Win Rate:  0.8
Player 2 Win Rate:  0.2
Average game play in 5.728731 seconds.
```

Figure 9.0

Figure 9.0 captures the results for the battles for the Monte Carlo and the MiniMax Agent with the Random Move Agent. The Minimax Agent was observed to have the highest win-rate vs the Random Move Agent, winning 80% of all games played.

The different computer agents were also trained in battles against the MiniMax computer agent. The results of those battles were captured for win-rate and efficiency i.e. average time to play the game. Figure 10.0 captures the results for the battles for the Q Learner and the Sarsa Learner Agent against the MiniMax Agent.

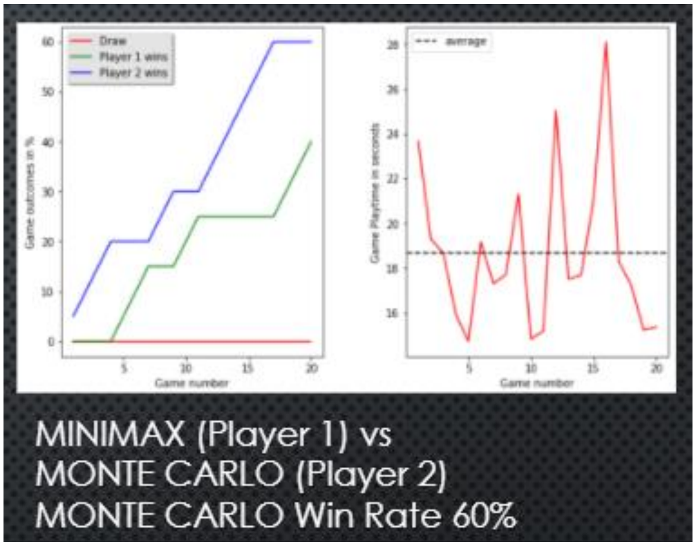


MINIMAX (Player 1) vs
Q LEARNER (Player 2)
QLEARNER Win Rate 30%

MINIMAX (Player 1) vs
SARSA LEARNER (Player 2)
SARSA LEARNER Win Rate 75%

Figure 10.0

Figure 11.0 captures the results for the battles between the Monte Carlo Agent and the MiniMax Agent.



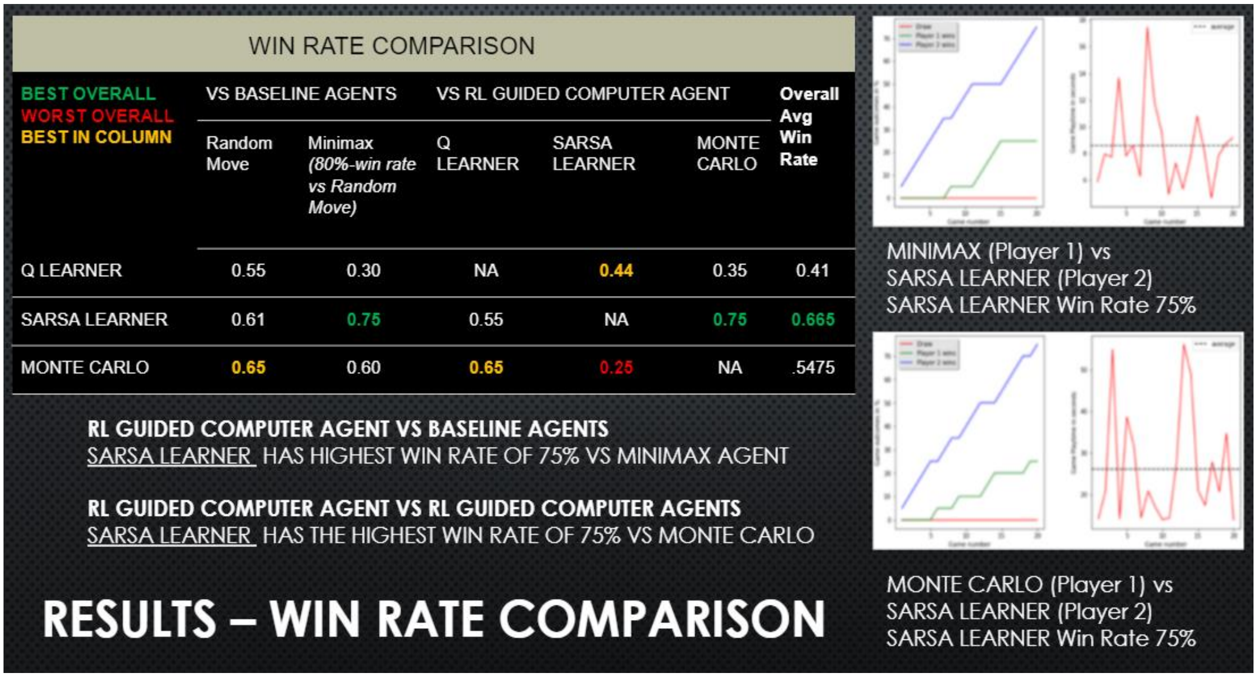
MINIMAX (Player 1) vs
MONTE CARLO (Player 2)
MONTE CARLO Win Rate 60%

Figure 11.0

The Sarsa Learner was observed to have the highest win-rate vs the MiniMax Agent, winning 75% of all games played.

Win-Rate Comparison Results

The results of the battles of different reinforcement learning guided computer agents with baseline agents (Random Move Agent and MiniMax Agent) were evaluated for win-rate. Figure 12.0 captures the results of the evaluation.



WIN RATE COMPARISON

BEST OVERALL WORST OVERALL BEST IN COLUMN	VS BASELINE AGENTS		VS RL GUIDED COMPUTER AGENT			Overall Avg Win Rate
	Random Move	*Minimax (80%-win rate vs Random Move)*	Q LEARNER	SARSA LEARNER	MONTE CARLO	
Q LEARNER	0.55	0.30	NA	0.44	0.35	0.41
SARSA LEARNER	0.61	0.75	0.55	NA	0.75	0.665
MONTE CARLO	0.65	0.60	0.65	0.25	NA	.5475

RL GUIDED COMPUTER AGENT VS BASELINE AGENTS

SARSA LEARNER HAS HIGHEST WIN RATE OF 75% VS MINIMAX AGENT

RL GUIDED COMPUTER AGENT VS RL GUIDED COMPUTER AGENTS

SARSA LEARNER HAS THE HIGHEST WIN RATE OF 75% VS MONTE CARLO

RESULTS – WIN RATE COMPARISON

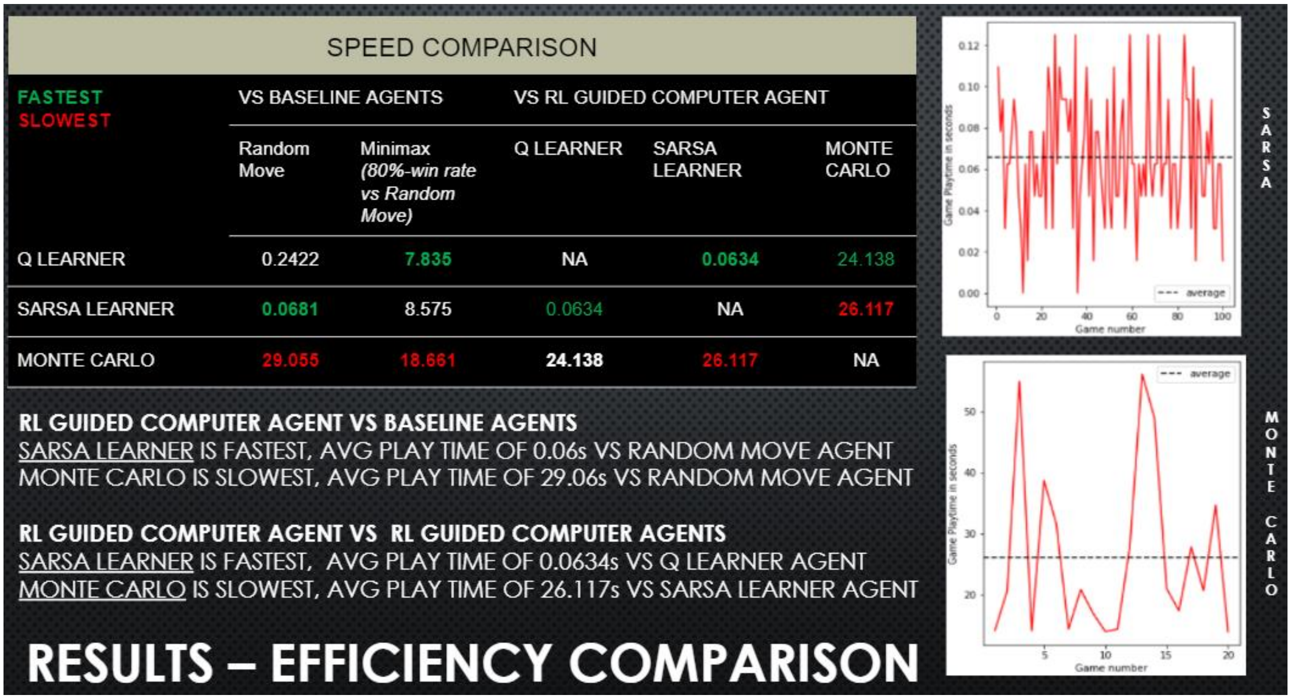
MINIMAX (Player 1) vs
SARSA LEARNER (Player 2)
SARSA LEARNER Win Rate 75%

MONTE CARLO (Player 1) vs
SARSA LEARNER (Player 2)
SARSA LEARNER Win Rate 75%

Figure 12.0

Efficiency (Average Play Time) Comparison Results

The results of the battles of different reinforcement learning guided computer agents with baseline agents (Random Move Agent and MiniMax Agent) were also evaluated for efficiency (average time to play the game). Figure 13.0 captures the results of the evaluation.



FASTEST SLOWEST	VS BASELINE AGENTS		VS RL GUIDED COMPUTER AGENT		
	Random Move	Minimax *(80%-win rate vs Random Move)*	Q LEARNER	SARSA LEARNER	MONTE CARLO
Q LEARNER	0.2422	7.835	NA	0.0634	24.138
SARSA LEARNER	0.0681	8.575	0.0634	NA	26.117
MONTE CARLO	29.055	18.661	24.138	26.117	NA

SPEED COMPARISON

RL GUIDED COMPUTER AGENT VS BASELINE AGENTS
SARSA LEARNER IS FASTEST, AVG PLAY TIME OF 0.06s VS RANDOM MOVE AGENT
MONTE CARLO IS SLOWEST, AVG PLAY TIME OF 29.06s VS RANDOM MOVE AGENT

RL GUIDED COMPUTER AGENT VS RL GUIDED COMPUTER AGENTS
SARSA LEARNER IS FASTEST, AVG PLAY TIME OF 0.0634s VS Q LEARNER AGENT
MONTE CARLO IS SLOWEST, AVG PLAY TIME OF 26.117s VS SARSA LEARNER AGENT

RESULTS – EFFICIENCY COMPARISON

Figure 13.0

Team Member Contributions

Member	Contribution
Abhishek Bais	● Performed background research on playing Connect Four using different AI approaches ● Implemented a reusable and easy to extend framework to play the game using different algorithms by refactoring the game into classes, have it accept different game playing algorithms as agents/ players, trained the agents via battles against other agents ● Implemented the Sarsa learning algorithm ● Designed a scheme to evaluate different reinforcement learning guided computer agents for win-rate by evaluating battles over 'N' games. ● Implemented visuals to capture the evaluation results ● Performed hyper parameter tuning for the Sarsa Learning algorithm, Re-evaluated results, captured best results ● Performed comparative study of Sarsa Learner Agent with baseline agents
Haley Feng	● Performed background research on playing Connect Four using different AI approaches ● Implemented the Q learning algorithm ● Designed a scheme to evaluate different reinforcement learning guided computer agents for efficiency by evaluating battles over 'N' games. ● Performed hyper parameter tuning for the Q Learning algorithm, Re-evaluated results, captured best results ● Performed comparative study of Q Learner Agent with baseline agents
Princy Joy	● Performed background research on playing Connect Four using different AI

	approaches ● Implemented the Monte Carlo algorithm ● Performed hyper parameter tuning for the Monte Carlo algorithm, Re-evaluated results, captured best results ● Performed comparative study of Monte Carlo Agent with baseline agents
Shannon Phu	● Performed background research on playing Connect Four using different AI approaches ● Implemented the Connect Four game design and user interface ● Implemented the MiniMax algorithm ● Performed comparative study of MiniMax Agent with Random Move Agent

Conclusions

In conclusion,

1. We successfully simulated a real-life gaming experience for a human player against different computer agents. We implemented five different computer agents namely Random Move Agent, MiniMax Agent, Monte Carlo Agent, Q Learner Agent, and a Sarsa Learner Agent that learnt to play the game. Of these, the Random Move Agent and Minimax Agent were none-reinforcement learning guided agents while the Monte Carlo, Q Learner and Sarsa Learner were reinforcement learning guided agents. To visualize the game in action, a graphic interface with a virtual Connect Four board and coins was also implemented.

2. We compared and contrasted the performance of different reinforcement learning guided computer agents on win-rate and efficiency i.e. average time to play the game. The performance of these agents were measured against baseline agents namely the Random Move Agent and the MiniMax Agent.

We found the Sarsa Learning Agent to be the best computer agent with the highest win-rate of 75% against both baseline agents and the highest average win-rate of 66.5% against baseline and other reinforcement learning guided computer agents. It was also the most efficient agent with an average play time of 0.06 secs, 435 times faster than the Monte Carlo Agent. Recall the Monte Carlo Agent formed the building block of the AlphaGo program.

In the future, we would like to explore other reinforcement learning computer algorithms such as DQN and multi-armed bandits. It would be interesting to see how they compare against the evaluated computer agents in this project.

Appendix

1. The Team Github repository: https://github.com/Team-Equality-RL-Project/connect-4

References

[1] G. Press, The Brute Force of IBM Deep Blue and Google DeepMind, Forbes, Feb 7, 2018. Accessed on: November 28, 2021. [Online]. Available: https://www.forbes.com/sites/gilpress/2018/02/07/the-brute-force-of-deep-blue-and-deep-learning/?sh=64a74ecd49e3

[2] S.J. Park, Google's AlphaGo AI trained on 30 million moves, AJUDaily, Feb 2, 2016. Accessed on: November 28, 2021. [Online]. Available: https://www.ajudaily.com/view/20160202145621103

[3] R. Karlsson, How to use reinforcement learning to play Tic-tac-toe, towards data science, May 15, 2020. Accessed on: November 28, 2021. [Online]. Available: https://towardsdatascience.com/how-to-play-tic-tac-toe-using-reinforcement-learning-9604130e56f6

[4] J. Juul, 255168 ways of playing Tic-tac-toe, The Ludologist, December 28, 2003. Accessed on: November 28, 2021. [Online]. Available: https://www.jesperjuul.net/ludologist/2003/12/28/255168-ways-of-playing-tic-tac-toe/

[5] S. Maha, Connect 4 with reinforcement learning, GitHub, August 20, 2017. Accessed on: November 28, 2021. [Online]. Available:https://github.com/ShekMaha/connect4-reinforcement-learning

[6] K. Galli, Connect 4-Python, GitHub, August 21, 2019. Accessed on: November 28, 2021. [Online]. Available:https://github.com/KeithGalli/Connect4-Python