Modelling a DRL agent to play the game of Rock Paper and Scissors

Jammula Durga Bala Sathvik [1] , N.Sushma [2] , Panchami Raghav [3]
Dept of Computer Science & EngineeringAmrita
School of Engineering, BengaluruAmrita
Vishwa Vidyapeetham India
sathvik6916[1],sushma6139[2],panchamiraghav[3] @gmail.com

***Acknowledgement -** We would like to thank Amrita School of Engineering, Bengaluru for giving us this opportunity. We would also like to express our gratitude to our mentor Dr. Amudha for guiding us throughout this study.*

***Abstract**: Reinforcement learning (RL) has emerged as a powerful technique for training agents to make intelligent decisions in complex environments. In this study, we present a novel approach to developing a Rock Paper Scissors game using the Deep Q-Network (DQN) algorithm and a model-based approach. The objective of our research is to create a stimulated experience by training an RL agent to play Rock Paper Scissors optimally against human players. To achieve this, we utilize a combination of deep neural networks, experience replay, and the DQN algorithm. The DQN algorithm enables the agent to learn a policy that maximizes its expected reward by approximating the action-value function using a deep neural network as a function approximator.*

Keywords— Game Optimization, RL, DQN, Simulated Experience.

I. INTRODUCTION

For generations, people have enjoyed playing the game of rock, paper, scissors because it requires players to think ahead and outsmart their opponents. The desire to create intelligent beings that can play this game at a high level has increased with the development of artificial intelligence. Machine learning's reinforcement learning (RL) branch offers a promising method for teaching agents how to make the best judgements possible in challenging situations.

In this work, we create a Deep Reinforcement Learning (DRL) agent to play the game of Rock, Paper, Scissors, which allows us to explore the fascinating realm of RL. Our goal is to develop an agent that can learn from its mistakes and make decisions that will increase its chances of success when competing against human players.

Another crucial element of our methodology, experience replay, enables our agent to draw lessons from a wide range of prior encounters. In this method, events are kept in a replay buffer and randomly selected during training. By doing this, the agent can explore various tactics and break connections between related events, resulting in more effective and robust learning.
Through our research, we hope to demonstrate DRL's modelling and gaming abilities for the game of Rock, Paper, Scissors. Our agent uses the DQN algorithm, experience replay, and the strength of deep neural networks to learn to adapt and make the best judgements possible based on its prior experiences. In the end, we hope that our work encourages future research into the field of intelligent game playing and advances RL approaches.

Game theory and decision-making are two areas in which Rock Paper Scissors fits. The study of strategic decision-making in circumstances involving several participants or players is known as game theory, and it is a subfield of mathematics and economics. Players in the game of Rock, Paper, or Scissors must select one of three outcomes (rock, paper, or scissors) in an effort to outsmart their rivals and secure a favourable result. The game incorporates elements of psychology, probability, and strategy, making it an engaging topic for game theory analysis and research.

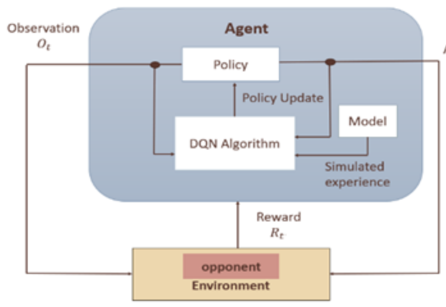


Fig1: RL Formulation

Fig1 gives us a detailed overview of the RL formulation visually. It is explained in detail below

A. Environment

The environment for the Rock, Paper, and Scissors game is a simulated game arena where the agent competes against an opponent by selecting one of three possible moves: rock, paper, or scissors. There are no specific parameters required for initializing the environment in the Rock, Paper, and Scissors game.
During each round of the game, the environment provides the agent with the current state of the game, including information about the previous moves made by both the agent and the opponent. The state space in the game includes:
• Agent's Previous Moves: A record of the agent's previous actions (rock, paper, or scissors) throughout the game.
• Opponent's Previous Moves: A record of the opponent's previous actions (rock, paper, or scissors) throughout the game.
• Game History: A record of the past actions and outcomes of the game.

B. State Space

The state space represents the information available to the agent at a given point in the game. In this implementation, the state space is represented by the state variable, which is a NumPy array. The shape of the state array is (1, 9), indicating that it has one dimension with nine elements. The elements of the state array correspond to different aspects of the game state, including:

- Win count: Number of times the agent has won the game.
- Tie count: Number of times the game has ended in a tie.
- Loss count: Number of times the agent has lost the game.
- Win rate trend: Trend indicator (0 or 1) based on the moving average of win rates.
- Tie rate trend: Trend indicator (0 or 1) based on the moving average of tie rates.
- Loss rate trend: Trend indicator (0 or 1) based on the moving average of loss rates.
- Win rate moving average: Moving average of the win rate over a specified window of previous games.
- Tie rate moving average: Moving average of the tie rate over a specified window of previous games.

Loss rate moving average: Moving average of the loss rate over a specified window of previous games.

C. Action Space

The action space represents the available choices or actions that the agent can take in the game. In this implementation, the action space is represented by the action space variable, which is a list containing three elements: [0, 1, 2]. Each element corresponds to a specific action that the agent can choose:

0: Represents the action of playing "rock".

1: Represents the action of playing "paper".

2: Represents the action of playing "scissors".

D. Reward Function

The reward function is based on the outcome of each round in the Rock, Paper, and Scissors (RPS) game. The reward is calculated as follows:

If the agent wins the round (its action beats the opponent's action), a positive reward of +1 is assigned.

If the round ends in a tie (both the agent and the opponent play the same move), a reward of 0 is assigned.
If the agent loses the round (opponent's action beats the agent's action), a negative reward of -1 is assigned.

II. LITERATURE SURVEY

Daeyeol et.al [1] investigates the neural basis of economic decision-making processes in primates using a modified version of the Q-learning algorithm in a rock-paper-scissors game. The study found that the monkeys' choice behaviour was consistent with a reinforcement learning model that incorporated statistical biases, and that these biases were reflected in the activity of dopamine neurons in the brain. The results suggest that dopamine neurons play a key role in reward-based learning and decision-making processes during rock-paper-scissors games.

Franco et.al [2] presents a study on how an external super-agent can exploit the behavior of adaptive agents in a game of Rock-Paper-Scissors to predict favorable moments to play against one of the other players. The two adaptive agents use a reinforcement learning algorithm to adapt their behavior in response to changes in the behavior of the other player.

David et.al [3] specifically mentions Shapley's variant of the rock-paper-scissors game as one of the simple games where the model-free algorithm and most other adaptive processes fail to converge. The paper explores this game and shows that by extending stochastic approximation results to multiple timescales, each player can learn at a different rate, allowing for convergence in this game as well.

Gregory et.al [4] mentions how humans learn in strategic environments using experimental data from the Rock-Paper-Scissors game. The authors show that humans can decode their opponents' strategies and outperform the random response by 17%. They also find that the strategy-based approach to learning is superior to the action-based approach and that humans are better at learning separate components of an opponent's strategy than recognizing the strategy. The paper provides insights into human learning and behavior in strategic situations and offers new extensions to existing learning models.

Zhijan et.al [5] investigates how humans make decisions in non-cooperative strategic interactions using the Rock-Paper-Scissors game. While classic Nash equilibrium theory predicts randomization of actions to avoid exploitation, evolutionary game theory predicts cyclic motions. The authors observe persistent cyclic motions in a laboratory experiment and explain them using a microscopic model of win-lose-tie conditional response.

Algorithm	Reference Papers	Model based/ Model Free	On policy/ off-Policy
Q-Learning	[1]	Model Free	Off-Policy
WOLF	[2]	Model Free	Off-Policy
SFP	[3]	Model Free	Off-Policy
WFP	[4]	Model Free	On-Policy
Monte Carlo	[6]	Model Based	

Table1: Summary of key works in RL-based inventory management

The above Table 1 describes about some of the algorithms used in RL-based inventory management works.

III. GAPS AND CHALLENGES

- Lack of Exploration-Exploitation Balance
- Limited State Representation

• Generalization and Robustness

By addressing these gaps and challenges, the project can be enhanced to create a more robust and effective DRL agent for playing the Rock, Paper, and Scissors game.

IV. TIMELINE

Our study adds to the existing body of knowledge by proposing a novel RL framework for inventory optimisation. Figure 2 depicts a timeline diagram of the stages of our project.

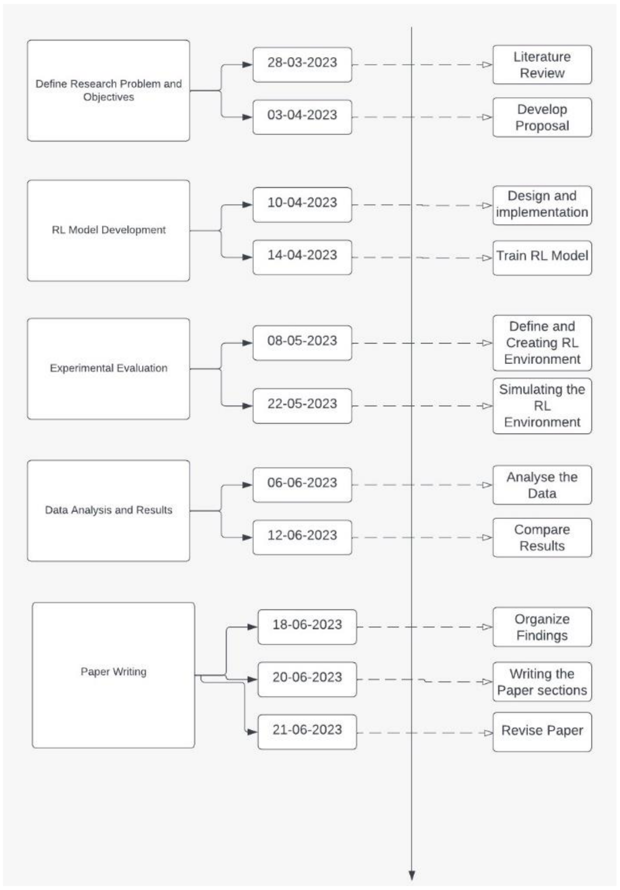


Fig 2: Timeline of the stages of the project

V. SYSTEM ARCHITECTURE

The project's system architecture is made up of several key components that work together to optimize the game through Reinforcement Learning (RL). Datageneration, RL model development, environment simulation, and performance evaluation are among these components. The high-level system architecture for the Rock Paper Scissors game using Deep Q-Learning can be divided into the following stages:

1. Data Generation:

The environment simulates the Rock Paper Scissors game and generates data by playing multiple episodes. Each episode consists of the agent interacting with the environment, selecting actions, receiving rewards, and transitioning between different game states. The data generated includes the current state, selected action, received reward, and the next state.

2. RL Model Development:

A deep neural network model, such as a Q-Network, is developed to approximate the Q-values for state-action pairs. The model takes the current state as input and outputs the Q-values for each possible action. The model is trained using the generated data from the environment to learn the optimal policy for playing the Rock Paper Scissors game. The training process involves updating the model's weights using techniques like Q-Learning or Deep Q-Learning.

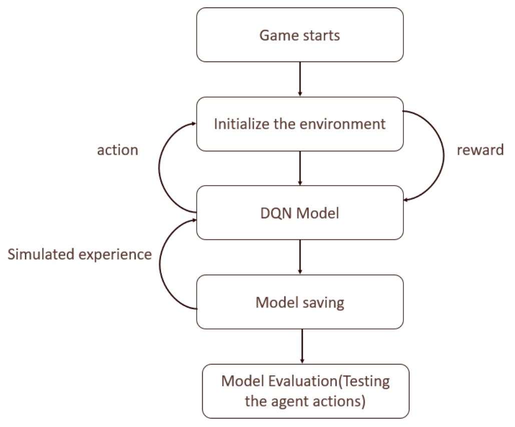
3. Environment Simulation:

The trained RL model is used to simulate the Rock Paper Scissors game. The model takes the current state as input and selects the action with the highest Q-value. The environment responds with the opponent's move, and the process continues until the game is completed.

4. Performance Evaluation:

The performance of the RL model is evaluated by playing multiple episodes of the Rock Paper Scissors game. Metrics such as win rate, tie rate, and loss rate are calculated to assess the effectiveness of the RL model. The RL model's performance can be compared against baselines or human performance to gauge its capabilities.

Throughout these stages, iterative improvements can be made to enhance the RL model's performance. This may include adjusting hyperparameters, modifying the model architecture, or employing more advanced RL algorithms. By following this architecture, the RL model can be trained and evaluated to effectively play the Rock Paper Scissors game, demonstrating the capabilities of Deep Q-Learning in game playing scenarios.



Game starts
Initialize the environment
action
reward
DQN Model
Simulated experience
Model saving
Model Evaluation(Testing the agent actions)

Fig 3: System Architecture

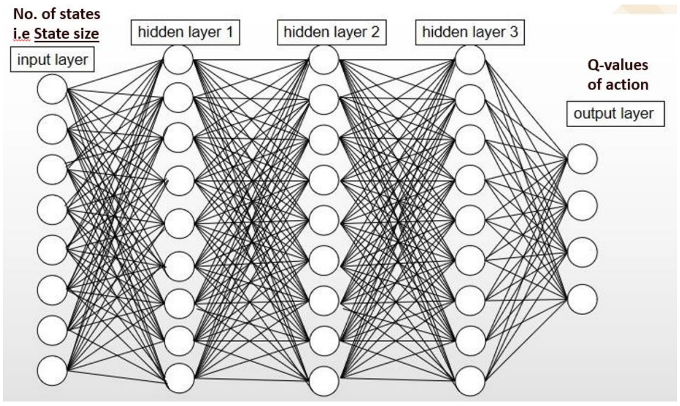


Fig4: DRL Architecture

In Fig 4 shows the architecture which is a feedforward neural network. It has three hidden layers, followed by an output layer with several units equal to the number of actions in the environment (which is 3 for the Rock, Paper, and Scissors game). The activation function used in the hidden layers is ReLU (Rectified Linear Unit).

Input:

The input to the DQN architecture is the state representation. The state representation includes various statistics about win, tie, and loss rates, as well as trends and moving averages. These features are provided as input to the Q-network.

Hidden Layer:

In the hidden layers of the Q-network, the input is transformed through a series of linear transformations followed by the activation function (ReLU). These transformations enable the network to learn complex patterns and representations from the input features, allowing it to approximate the Q-values for different actions given the state.

Output Layer:

The output of the DQN is a vector representing the Q-values for each action. The Q-values estimate the expected cumulative rewards the agent can achieve by taking each action in the current state. The action with the highest Q-value is selected as the agent's predicted action, guiding its decision-making process.

During the training process, the DQN learns to update its weights and biases to minimize the mean squared error between the predicted Q-values and the target Q-values. This update is performed using the Adam optimizer with a specified learning rate.

VI. IMPLEMENTATION

Initialize the Environment:
It initializes the game parameters such as action space, opponent player mode, counts for opponent and agent moves, window size for moving averages, and the initial state. The environment provides methods for resetting the game state, taking a step based on an action, and calculating rewards and termination status.

Setting the Agent:
The agent maintains a replay memory buffer, gamma discount factor, epsilon exploration factor, learning rate, and tau for target network update. It has a Q-network and a target network, both implemented using a deep neural network model with multiple dense layers. The agent has methods for selecting actions, storing experiences in the replay memory, performing experience replay, and updating the target network.

Target Network and Local Network:
The target network and local network are instances of the agent's Q-network. The target network is periodically updated by blending the weights of the local network using the tau parameter. This helps stabilize the learning process.

Policy Architecture:
The Q-network takes the state as input and outputs Q-values for each action. It consists of multiple dense layers with ReLU activation.

Training Loop:
The main training loop is implemented in the main function. It initializes the environment and the agent, defines the number of episodes and trial length. Within each episode, the agent interacts with the environment by selecting actions, performing steps, and updating its Q-network. The agent's action selection follows an epsilon-greedy exploration strategy. Performance metrics such as win rate, tie rate, loss rate, player move rates, average reward, and average Q-values are calculated and printed after each episode.

Model Saving:
The trained model can be saved for future use. Once the model is saved, it can be loaded and reused in the future using the appropriate methods provided by the deep learning framework being used.

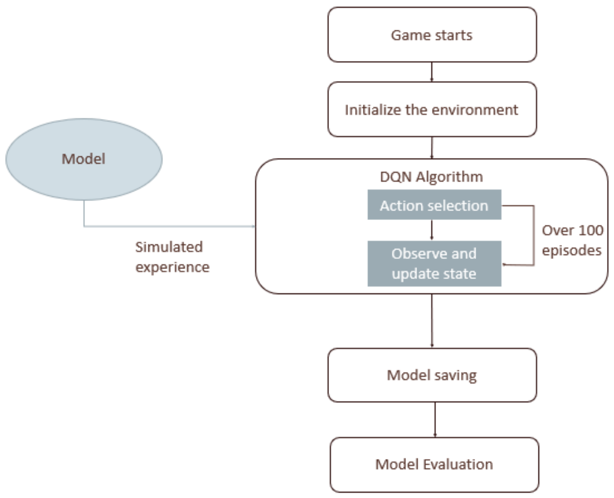


Fig 5: A High-Level System Architecture

1. **Replay Memory Size:** This hyperparameter determines the size of the replay memory buffer, which stores the agent's experiences for experience replay.

2. **Discount Factor (Gamma):** The discount factor controls the importance of future rewards compared to immediate rewards.
3. **Exploration Rate (Epsilon):** Epsilon determines the balance between exploration and exploitation in the agent's action selection strategy.
4. **Minimum Exploration Rate (Epsilon Min):** This hyperparameter sets the lower bound for the exploration rate (epsilon). Once epsilon reaches this minimum value, the agent will continue to exploit the learned policy rather than exploring further.
5. **Epsilon Decay Rate:** Epsilon decay rate determines the rate at which the exploration rate (epsilon) decreases over time.
6. **Learning Rate:** The learning rate determines the step size, or the amount of adjustment made to the Q-network weights during each iteration of the optimization process.
7. **Tau:** Tau is the parameter used for updating the target network weights based on a fraction of the local network weights.

VII. RESULTS

During the training loop of the model, we plot the accumulated profits over the demand scenarios, the increasing trend indicates the learning progress and improvement of the agent over time. Once the model is training for the defined number of episodes it has to be saved. The saved model is further used for testing and evaluating the agent based on the actions it performs and rewards it gains over time. As the agent trains through the episodes, it gradually learns to make better decisions, optimize inventory management, and increase profits. The increasing plot suggests that the agent's policy becomes more effective as training progresses, and it gains a better understanding of the demand patterns and optimal ordering strategies.

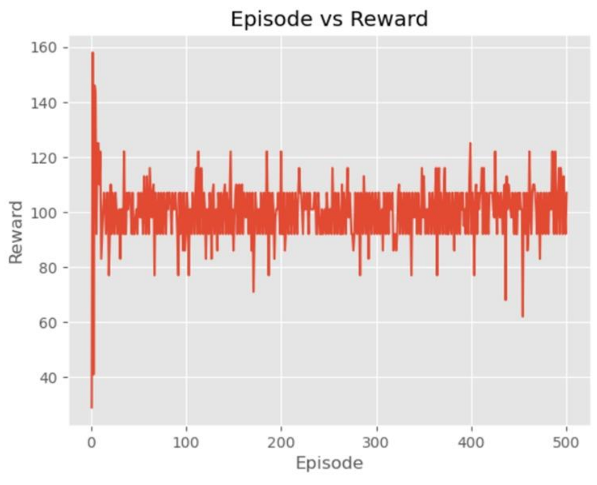


Fig 6: Accumulated profits over 500 episodes

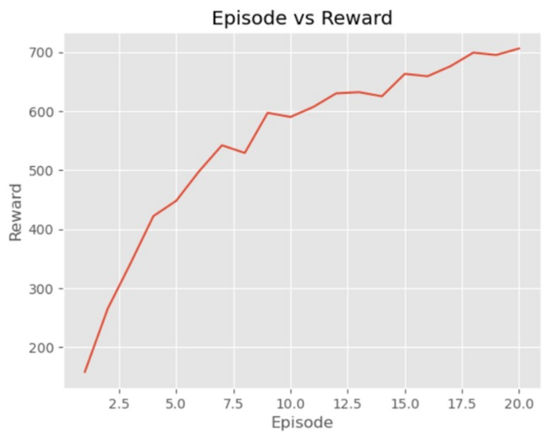


Fig 7 Episode vs Reward graph for 20 episodes

VIII. CONCLUSION AND FUTURE SCOPE

The project showcases the use of a Model-Based DQN algorithm, where the environment itself acts as the opponent. The environment keeps track of win, tie, and loss rates, and the agent learns to make decisions based on its observations and previous experiences. However, there are gaps and challenges in the project that can be addressed to enhance its effectiveness. These include improving the exploration-exploitation balance, expanding the state representation, enhancing the opponent model, optimizing hyperparameters, evaluating performance metrics, ensuring generalization and robustness, and incorporating visualization and analysis.

To model a Deep Reinforcement Learning (DRL) agent to play Rock, Paper, Scissors, future research will investigate multi-agent scenarios to train the agent against multiple opponents, improve opponent modelling techniques to improve the agent's performance, investigate transfer learning to apply the agent's knowledge to related tasks, refine exploration-exploitation strategies for optimal decision-making, and expand the game with variations and dynamics.

IX. REFERENCES

[1] Daeyol Lee, Benjamin P.McGreevy, Dominic J.Barraclough , "Learning and decision making in monkeys during a rock–paper–scissors game", sciencedirect,2005.

[2] Franco Salvetti, Paolo Patelli, Simone Nicolo, " Chaotic time series prediction for the game, Rock-Paper-Scissors",Sciencedirect,2006.

[3] David S. Leslie, E.J.Collins, "CONVERGENT MULTIPLE-TIMESCALES REINFORCEMENT LEARNING ALGORITHMS IN NORMAL FORM GAMES", 2003,Vol.13,No.4,1231-1251.

[4] Gregory Chernov, "How to Learn to Defeat Noisy Robot in Rock-Paper-Scissors Game: An Exploratory Study",2020 HSE Economic Journal.

[5] ZhijanWang,Bin Xu,Hai-Jun Zhou , "Social cycling and conditional responses in the Rock-Paper-Scissors game",2014,Scopus .

[6] Bikramjit Banerjee, Jing Peng, "Strategic best-response learning in multiagent systems", Journal of Experimental & Theoretical Artificial Intelligence,Vol.24,No.2,June 2012,139-160.