# 

[**CHAPTER 1: INTRODUCTION**](#_tg9neuecco55) **1**

[1.1 Background](#_xtlxizcbxv2a) 1

[1.2. Problem Statement and Motivation](#_m7jp8jgm26j) 1

[1.3. Project Aim and Objectives](#_poitx33z0u5c) 2

[1.3.1. Aim](#_3ph53nnabpnr) 2

[1.3.2. Objectives](#_p82g7dyi69up) 2

[1.4. Overview of the Report](#_nqxhmhf20x3v) 3

[**CHAPTER 2: LITERATURE REVIEW**](#_gjp0umeygizx) **4**

[2.1. Introduction to Reinforcement Learning](#_2p5pv0xpkxal) 4

[2.2. Reinforcement Learning in Stochastic Environments & Games](#_50bfea2cun5i) 4

[2.2.1 RL in Games](#_kncjit5560gt) 4

[2.2.2. RL in Stochastic Environment](#_kuvvy47cf8kr) 5

[2.3. Proximal Policy Optimization (PPO): Strengths & Limitations](#_t91u8nlwuras) 5

[2.3.1. Exploration Enhancements in PPO](#_ebc4uorycl5l) 6

[2.3.2. Variance Control and Stability in Policy Updates](#_6q7x39qvzkxb) 6

[2.3.3. Sample Efficiency and Limitations of PPO in Low-Data Regimes](#_mbnadxpxmbsh) 7

[2.3.4. Challenges of Exploration and Community Insights](#_gocev99j1q2g) 7

[2.4. Reinforcement Learning in Gambling and Change-Based Games](#_gp8ijltai1d9) 8

[2.4.1. The UCLA Roulette Project](#_92dnrov0vedh) 8

[2.4.2. WagerWin and RL in Stochastic Card Game](#_s5qj90csaprn) 9

[2.4.3. Implication for European Roulette](#_awmuupdgb148) 9

[2.5. Comparison of PPO with Other Reinforcement Learning Algorithms](#_wxdfk67gx0om) 10

[2.6. Risk-Aware and Safety RL considerations](#_do1z255egcfc) 10

[2.7. State and Action Representations in Probability-Driven Game](#_8zt29pnc7kbg) 11

[2.8. Summary of Literatures](#_d800p3b4eoob) 12

[**CHAPTER 3: METHODOLOGY**](#_dsjery9hhtfc) **14**

[3.1. Dataset Generation](#_2ozk3nl2n5d0) 14

[3.1.1. Roulette Simulator Design](#_lfy5hfriborv) 14

[3.1.2. Data Collection process](#_rrgygmg6vkyr) 14

[3.1.3. Realism and Statistical Validation](#_1deqzebb9zes) 15

[3.2. Training and Testing Data](#_xibd32axtr4n) 16

[3.3. Model Selection](#_fwzzufif1pm3) 17

[3.3.1. Reinforcement Learning Models](#_m18m4ugqm6dm) 17

[3.3.1.1 Proximal Policy Optimization (PPO)](#_n0woc51q5jfr) 18

[3.3.1.2. Deep Q-Network (DQN)](#_37sv4lfzdgqh) 18

[3.3.2. Classical Betting Strategies](#_st0qizr1b11d) 19

[3.3.2.1. Martingale](#_3fpfepwxfqa1) 19

[3.3.2.2. Fibonacci](#_z8pbq6xu0mwo) 20

[3.3.2.3. D’Alembert](#_9fhicbzfi2ab) 20

[3.4. Evaluation Metrics](#_ypbdlocfrpmt) 20

[3.4.1. Total Reward](#_kdzh0rx1cmy0) 21

[3.4.2. Average Rewards per Spin](#_udu8ueb40mn8) 21

[3.4.3. Additional Metrics for PPO and DQN](#_ljtxwjempe71) 21

[3.5. Implementation](#_rdd859qcn1tw) 23

[3.5.1. Tools and Libraries Used](#_flivhkzedw4g) 23

[3.5.2. Coding Standards Followed](#_n3jsiud5om4w) 24

[3.5.3. Algorithm Steps](#_aa46gwftct6z) 25

[3.5.3.1. PPO Algorithm Steps](#_73ldj15btujs) 25

[3.5.3.2. DQN Algorithm Steps](#_3ws4fhjhurol) 25

[**CHAPTER 4. RESULTS**](#_i0ph7h8tsbyz) **27**

[4.1. PPO Agent Performance](#_asdrq3a7926b) 27

[4.2. Classical Strategy Comparison](#_88ku3iqyo2t0) 28

[4.2.1. Martingale Strategy](#_12fbxfjwaa6b) 29

[4.2.2. Fibonacci Strategy](#_m6ctu8kmu0uy) 29

[4.2.3. D’Alembert Strategy](#_axm8uo5q3k0b) 29

[4.2.4. Random Strategy](#_mm8kb1ie6ut7) 30

[4.3. DQN Strategy](#_ifodgvv16zym) 30

[4.4. Comparative Summary](#_lfjxwasaa9pf) 31

[**CHAPTER 5. PROJECT MANAGEMENT**](#_r27uh7up3o6d) **34**

[5.1. Project Schedule](#_uvewnt8duyh6) 34

[5.2. Risk Management](#_jdcn3vkmuhmu) 35

[5.3. Quality Management](#_9rufnkwggvxq) 36

[5.4. Ethical, Legal and Professional Considerations](#_bf46faybiap) 37

[**CHAPTER 6. DISCUSSION**](#_p3b8b6vjg6wy) **38**

[6.1. Structural Incompatibility Between RL and Roulette](#_nspzwkqzphes) 38

[6.2. Illusion of Complexity vs. Practical Resilience](#_j24tgk87smeu) 39

[6.3. Survival is not Equal to Success](#_ibu1re599ozg) 39

[6.4. Benchmarking RL in Unrewarding or Stochastic Environments](#_ubked3teyp7c) 40

[6.5. False Pattern Detection and Overfitting](#_nshzgfkfk90d) 41

[6.6. Broader Implications for Algorithmic Betting](#_wr9sypdi5tjc) 41

[**CHAPTER 7. CONCLUSION**](#_drg58qqzdo2y) **42**

[7.1. Conclusion](#_iitg6uitmo27) 42

[7.2. Limitations of the Study](#_nn46uvqlwjex) 43

[7.3. Future Enhancements](#_gf9vez9nv6cu) 43

[**CHAPTER 8. STUDENT REFLECTIONS**](#_1kq9j4dtfpl6) **45**

[**REFERENCES**](#_ivjp6hvacha) **48**

# 

# CHAPTER 1: INTRODUCTION

## 1.1 Background

European Roulette is a classic casino game which consists of the spinning wheel, a ball, and a set of numbered pockets which range from number 0 to 36 (GGpoker, 2025). The rules of the game are pretty simple and have relatively favorable odds as compared to other variants. Due to this, it can be one of the topics of interest in gambling theory and statistical modelling (Small & Tse, 2012; Blavastskyy, 2024). The roulette game is of stochastic nature and is regulated by chance. It has long been regarded as non-optimizable because of its probabilistic nature and house edge (Arend, 2024).

However, in recent years, the field of machine learning has advanced significantly and has been proven to provide good results in various fields including gambling games (Hassanniakalager & Newall, 2022; Jordan & Mitchell, 2025). The field of machine learning, especially the Reinforcement Learning (RL) , has been used extensively in exploring whether intelligent agents can be used for discovering patterns, strategies or heuristics that help to optimize performance in games of chance (Steingroever et al., 2014; Bilgin, 2020). Proximal Policy Optimization (PPO) has come out as one of the reliable and efficient RL approaches for training agents in complex environments (Schulman et al., 2017). PPO is well known for its balance of exploration and exploitation and its stability in continuous action spaces, making it a good choice for investigating strategies in stochastic games like European Roulette (Del Rio et al., 2024).

## 1.2. Problem Statement and Motivation

Despite the probabilistic nature and inherent house edge, the European Roulette game provides an engaging testbed for investigating the capabilities and limitations of the RLsystems. There are traditional betting techniques like Martingale, Fibonacci system etc which have been studied extensively (Payne et al., 2023). However, from most of the studies, use of these algorithms in case of the game of chance has been proven to be ineffective in the long run. However there is still a question which remains: *Can a reinforcement learning algorithm, particularly PPO agent trained in a constrained, realistic roulette simulation outperform traditional static betting systems in short-term cumulative return and risk-adjusted performance?*

The main motivation for conducting this research work is to push the frontiers of what reinforcement learning techniques can do in the situations that are governed by chance rather than skill. This research thus aims to bridge the gap between the theoretical machine learning and practical application of the reinforcement learning algorithm, particularly the PPO method in complicated stochastic systems like roulette games.

## 1.3. Project Aim and Objectives

### 1.3.1. Aim

The aim of this dissertation project is to design, implement and evaluate a reinforcement learning model using the Proximal Policy Optimization (PPO) algorithm to identify optimal betting strategies for the European Roulette game.

### 1.3.2. Objectives

To accomplish the aim of this dissertation, the key objectives that have been created are listed below:

1. To model the European Roulette game environments in a simulation platform compatible with reinforcement learning frameworks.
2. To implement a PPO-based RL agent which is capable of interacting with the roulette environment.
3. To define appropriate reward functions and training episodes and make them align with the betting conditions which are realistic.
4. To perform evaluation of the performance of the RL agent against traditional betting strategies such as Random betting, Martingale etc.
5. To perform proper analysis of the learned strategies for patterns, strengths and limitations.
6. To properly discuss the ethical and practical implications of using ML based algorithms in the context of gambling.

## 1.4. Overview of the Report

The dissertation report has been structured into various chapters. This first chapter is the introduction chapter which provides the background and states the research question, motivation behind the research, aims and objectives of the project. After this introduction chapter other chapter follows which includes:

1. Chapter 2: Literature Review: This chapter reviews previous research focusing mainly on the roulette strategy optimization, reinforcement learning, and PPO in gaming environments. From the comprehensive review of the existing literature the gaps in research have also been identified and presented in this chapter.
2. Chapter 3: Methodology: In this chapter, the description of the design of the roulette simulation has been presented along with the implementation of the PPO algorithm. In addition to this, the methodology also properly outlines the methodology for training setup of the PPO algorithm. The methodology has been presented with main focus on the reproducibility of the work.
3. Chapter 4: Implementation and Experimentation: In this part the details of the experimental setup, agent training process, tools and technologies used for coding, hyperparameter tuning and reward structure etc. has been presented.
4. Chapter 5: Results and Discussion: In this chapter of the report, the results and outcomes of the experiment have been presented which compares the outcome from the PPO algorithms with the baseline strategies. The detailed interpretation has been presented.
5. Chapter 6: Conclusion and Future work: In this chapter, the key insights obtained from the experiment has been presented along with the acknowledgement of the limitations and outline for the future research directions.

# CHAPTER 2: LITERATURE REVIEW

## 2.1. Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a field of machine learning which involves training agents to make sequential decisions by interaction with their environment and maximizing cumulative rewards (Shakya et al., 2023). The environments in the reinforcement learning can be deterministic, adversarial or fully stochastic (Zhang et al., 2020; Vinitsky et al., 2020) . In the context of this dissertation, the European Roulette is governed by pure randomness and fixed probabilistic outcomes and hence the most relevant domain for this is that of stochastic environments.

## 2.2. Reinforcement Learning in Stochastic Environments & Games

### 2.2.1 RL in Games

RL algorithms have shown good performance in various complex games which range from board games like Alpha Go to real-time strategy games like Dota 2 (OpenAI Five) (Souchleris et al., 2023). In aAlphaGo & AlphaGo Zero, the RL based agent was able to achieve superhuman performance using policy/value networks combined with Monte Carlo Tree Search (MCTS). Further improvements were seen in AlphaGo Zero where the agent learned via self-play without human data (Soucheleris et al., 2023). The OpenAI Five coordinated five RL-based agents in the complex Dota 2 game world using a modified version of PPO.This showed the potential of PPO algorithm in multip-agent,partially observable and real-time environments (Souchleris et al., 2023). These accomplishments show how RL may be used to games with unanticipated dynamics, especially when using policy-gradient-based techniques like PPO. It's crucial to differentiate between games like Roulette, which function in fully stochastic environments, and those with particle randomness and strategic intricacy, such as GO or Dota 2.

### 2.2.2. RL in Stochastic Environment

Stochastic environments are different from the environments in strategic games in a sense that the stochastic environments are characterized by probabilistic transitions, lack of long-term skill influence, and reward outcomes which are not deterministic (Kleshnina et al., 2023). Such an environment poses a significant challenge as it is very difficult to identify patterns, optimize risk, and manage uncertainty effectively. RL has been used in various stochastic domains which are driven by chance. One of such examples is the application of RL in Casino and Gambling Games. The Rlette project by Colonnese & Rakens, (2020), used PPO and other RL algorithms to simulate Casino Roulette games. Over time, the RL agents learned to minimize losses by determining that the best approach was to leave the table as soon as possible rather than attempting to adjust their betting combinations while playing. This outcome is consistent with the inherent lack of exploitable structure of the roulette game. This showed the ability of RL in reducing expected loss in stationary probabilistic settings by not falling into negative expected value traps rather than attempting to 'win' in the classical sense (Colonnesse & Rakens, 2020). Apart from gambling, another example of RL in a stochastic environment includes use in finance and tradition. The paper Jiang et al. (2017) developed a deep reinforcement learning framework for portfolio management in stochastic financial markets. The outcome showed that the RL agents could learn to balance risk and reward in volatile, non-stationary markets.

## 2.3. Proximal Policy Optimization (PPO): Strengths & Limitations

PPO is an on-policy, policy-gradient RL algorithm. It is a simple and more stable model that was created as an alternative to Trust Region Policy Optimization (TRPO). The PPO algorithms optimized by clipping policy updates to control magnitude and avoid destructive step sizes. One of the foundational work on PPO is the Schulman et al., (2017). The clipped surrogate objective-based PPO suggested in the study provided advantageous sample complexity and simplicity of implementation in both discrete and continuous action spaces (Schulman et al., 2017).

### 2.3.1. Exploration Enhancements in PPO

Various enhancements have been made to the PPO. Recent research such as IEM-PPO (Intrinsic Exploration Module PPO) improves the functionalities of PPO by incorporating uncertainty-aware intrinsic rewards for improving the exploration in continuous control tasks. With this method, the agent receives rewards for investigating epistemic uncertainty using neural approximations and using them to guide the exploration, rather than just depending on the environment rewards. In sparse-reward environments, this technique has been shown to perform better than the baseline PPO strategy. This implies that it could be helpful in situations similar to roulette, where results are typically repetitious and uninformative (Zhang et al., 2022). Similarly there are other approaches such as VIME(Variational Information Maximizing Exploration) and RND (Random Network Distillation) which have been adapted to PPO for improving the exploration under sparse feedback conditions. These methods give rewards to the agent for surprise or novelty. Due to this, it helps the methods to resist the urge to exploit noisy rewards too early in the training process.

### 2.3.2. Variance Control and Stability in Policy Updates

Another critical limitation of the PPO in stochastic environments is high policy variance. This can cause training instability or can be a cause of early and premature convergence. To alleviate this, PPO-CMA (Covariance Matrix Adaptation PPO) has been developed.

WILL ADD LATER

# CHAPTER 3: METHODOLOGY

## 3.1. Dataset Generation

To train a reinforcement learning (RL) agent for roulette, a large and representative dataset of roulette outcomes is essential. The real-world casino data was inaccessible and limited in volume. Due to this, a European roulette simulation environment was built to generate synthetic but realistic gameplay data. This simulator acts as the foundation for both training and evaluating the reinforcement learning model.

### 3.1.1. Roulette Simulator Design

A European-style roulette wheel was simulated using Python. The simulation includes all 37 pockets numbered from 0 to 36, making it consistent with the real-world version. Each number is associated with a color (0-green, red for 18 specific numbers, and black for remaining 18). This configuration in simulation mirrored the actual roulette wheel design,

### 3.1.2. Data Collection process

To produce a robust dataset, 100000 spins were simulated. For each spin, the following set of features were recorded:

Table: Description of the features of the dataset

| Feature | Description |
| --- | --- |
| Spin Number | The iteration of the current spin. |
| Number | The outcome of the spin (ranging from 0 to 36). |
| Color | The associated color of the landed number (red, black, or green). |
| Parity | Classification as even, odd, or zero. |
| Range | Categorized as low (1–18), high (19–36), or zero |

All spin results were expired in the CSV file and were used in training and analysis.

### 3.1.3. Realism and Statistical Validation

To ensure the simulation was statistically fair and aligned with theoretical roulette probabilities, various frequency analyses were performed. These statistical analysis of the simulated dataset confirmed that the roulette outcomes closely approximate theoretical predictions. In color distribution, red outcomes occurred about 48.89%, black outcomes occurred approximately 48.41%, and green (for zero) happened in 2.70% of the spins. These results exactly match the odds that ought to be expected in an evenly divided European roulette wheel, with the single green pocket accounting for 1 out of 37 potential results (approximately 2.70%). Parity was also divided evenly, odd numbers spun roughly 48.90% and even numbers roughly 48.40%, zero accounting for the remaining 2.70%. Similarly, the range-based distribution also showed that low values (1–18) occurred 48.54%, high values (19–36) occurred 48.76%, and zero also saw its forecasted frequency of 2.70%. In addition to these, the frequency of each number also followed uniform distribution. These confirm that the simulator mimics the behavior of the real roulette game and there is no noticeable skew or irregularities in the data produced. The simulator also includes internal checks to flag any category exceeding realistic frequency thresholds, further reinforcing data integrity.

| Figure 1: pie-chart of the color distribution of simulated roulette outcomes | Figure 2: Bar chart showing the parity distribution of the simulated roulette outcomes |
| --- | --- |
| Figure 3: Bar chart showing the distribution of the number from simulated outcomes following uniform distribution | |

In addition to these visualizations, a chi-square test was also conducted to see if there is any significant difference in the simulation results and the actual roulette outcomes. The significance level was set to 0.05. The p-value obtained was less than 0.01 which indicated that there is no significant difference between the simulated outcomes and the real roulette spins. Hence, the data successfully mimicked the real world roulette spins.

## 3.2. Training and Testing Data

For training the reinforcement learning agents and evaluating their performance, it was necessary to divide the simulated roulette dataset into training set and testing set. The training data would be used for training the algorithms while the testing dataset for evaluating the models on the unseen dataset so as to see how well the model generalizes on the new unseen data.

The dataset used in this study was a simulated log of fair roulette spins which contained the outcome number, color, and parity for each spin. For consistency and to preserve the temporal order of the events, the data was splitted into training and testing sets using a 70 to 30 ratio, without shuffling.

Table I: Shows the number of spins used in each set

| Dataset | Training Set | Percentage | Test Set | Percentage |
| --- | --- | --- | --- | --- |
| Roulette Spin Data | 700000 | 70% | 300000 | 30% |

This splitting of the data allowed the PPO and DQN models to learn from historical outcomes while being evaluated on the test data which was not provided during the training.

## 3.3. Model Selection

One of the most important components of this study was the selection of appropriate models. The main focus of this project was to explore the viability of reinforcement learning techniques in optimizing betting strategies for roulette. The unpredictability and lack of exploitable patterns in the roulette game are well-known.Since traditional machine learning techniques are not very suitable when it comes to modeling the stochastic data and environments, the study concentrated on the use of the state-of-the-art reinforcement learning algorithms, especially Proximal Policy Optimization (PPO) (Hu & Lauriere, 2023). In addition to this, Deep Q-Networks (DQN) was also explored. Both of these reinforcement learning techniques are well-supported and widely used in reinforcement learning research. To complement these reinforcement learning models, classical betting systems like Martingale, Fibonacci, and D’Alembert, as well as a Random agent were also used as a control for comparison.

The motivation behind using these diverse range of models was to assess the effectiveness of learned strategies in comparison to rule-based systems. All of these models were implemented within a unified, specially created roulette simulation environment that was designed to replicate real-world betting constraints as precisely as possible. This made it possible to evaluate their performance under the identical conditions in a fair and consistent manner.

### 3.3.1. Reinforcement Learning Models

Even though roulette is a game that is inherently random, the primary goal was to train the RL agents to make bets that minimize long-term losses or, ideally, detect betting trends that may result in net wins.

#### 3.3.1.1 Proximal Policy Optimization (PPO)

This algorithm was selected as the primary reinforcement learning algorithm in this study. It is a policy-gradient method. In this method, the algorithm directly learns the probability distribution over actions (Del Rio et al., 2024). The PPO consists of the clipped objective function which is used for limiting the amount that the policy can change with each update. This process of clipping improves the stability of the training and it also prevents the model from making very big changes that can cause unpredictable performance (Zhu & Rosendo, 2021).

In this dissertation, the PPO agent was trained within a custom roulette environment which simulates the outcome of a spin, calculates payouts, and enforces betting rules such as progression limits, maximum bet sizes etc. The environment was implemented using the Gymnasium framework. This framework provided a very structured way to define observation and action spaces.

The current bet amount, the progression count, and the past five spin results made up the observation space for the PPO model. This information was used as the input for the PPO agent at every timestep. For designing the action space, a comprehensive list of the possible bets which included straight numbers, red or black, odd or even, dozens, columns and special combinations like trios and baskets was used. Each of the action types represented a distinct bet type and value. The PPO agent was trained over one hundred thousand timesteps using historical roulette spin data. During the training, the model showed steady progress in terms of policy loss, entropy loss, and learning stability. The explained variance, which measures how well the value function forecasted future rewards, stayed very close to zero or even slightly negative. This suggested that because the roulette results were random, the model had trouble identifying recurring patterns. Despite these outputs, the training process provided valuable information and insights into how the PPO models interact with certain environments. The agent showed an ability to adjust bets based on the previous outcomes and also maintained a good degree of internal logic, even if it didn’t seem to explain the variance in the data effectively.

#### 3.3.1.2. Deep Q-Network (DQN)

The second RL algorithm used in the dissertation was DQN. This algorithm is different from the PPO algorithms in that it learns a value function instead of a policy.DQN works by estimating the expected future reward of taking each action from a given state and then performs selection of the action which has the highest anticipated value. This algorithm depends on techniques like experience replay and target networks for stabilizing the learning process.

For this project, DQN was configured with a buffer size of around 50,000. The other hyperparameters like a learning rate was set to 0.0001, and a batch size was set to 32. These hyperparameters were selected for the training of DQN based on standard practices. Like PPO, DQN was also trained in the same roulette environment using the same state and action definitions.

Training the DQN agent required more careful handling of the experience relay buffer and turned out to be more sensitive to hyperparameter adjustments. As the training process progressed, the initial exploration of the random acts of the agent progressively gave way to exploitation. However, the learning process of the DQN was also similar to that of the PPO approach and it was not very good at finding the patterns in the data as the stochastic and independent nature of the roulette outcomes offered very little in terms of long-term dependencies or exploitable features.

### 3.3.2. Classical Betting Strategies

To benchmark the performance of the reinforcement learning models, various classical betting strategies were also implemented and evaluated in this dissertation. These algorithms were also tested under identical conditions as the RL models. These classical strategies are very popular algorithms in gambling literature and work by following certain predefined rules instead of learning from the environment like RL algorithms. These algorithms, unlike RL, don't adapt or improve over time. However, they are very predictable and simple which makes them useful for comparison.

#### 3.3.2.1. Martingale

This is one of the most popular classical betting techniques in which the bet is doubled after every loss so as to recover the previous losses with a single win. Once the player wins the bet, the bet amount is reset to the base bet value. This betting technique works well in short losing streaks. However, this can also cause bet amounts to grow exponentially after each loss, which are very risky under financial or casino-imposed limits (Victor, 2015; Turner, 1998).

In the test environment for this project, the Martingale agent was made to consistently bet on the red and follow the doubling rule after each loss. The agent achieved one of the best results among all the strategies in terms of minimizing average loss per spin.

#### 3.3.2.2. Fibonacci

The Fibonacci is another classical technique which was built in this project. This betting strategy increments the size of bet as per the Fibonacci sequence after losses and in case of the win it steps back two positions. In this the bet size is increased more slowly than Martingale approach and thus, helps to manage risk while also aiming to recover the previous losses. The Fibonacci agent performed similarly to the Martingale strategy in terms of average losses.

#### 3.3.2.3. D’Alembert

This approach works by increasing the bet by one unit after a loss and decreasing it by one unit after a win. This strategy works on the assumption that wins and losses will balance out in the long-run. Thus, this model is more conservative than the Martingale or Fibonacci technique.

## 3.4. Evaluation Metrics

In order to assess the performance of models, appropriate metrics which adheres to the principles of reinforcement learning had to be used in this project. Reinforcement learning models like PPO and DQN, are evaluated on their ability to maximize rewards through sequential decision-making, in contrast to typical supervised learning tasks where classification accuracy is the main metric used. As a result, the primary metric used in this project was Total Reward. In addition to this, other secondary indicators which are relevant to the training and evaluation of the RL agents were also used. Each of the metrics used in this project are discussed below:

### 3.4.1. Total Reward

It is calculated as the sum of all rewards that are collected by the agent over a complete episode. In the context of the roulette, it is the net profit or loss resulting from the betting decisions which are made during the gameplay (Muslimani et al., 2025). It is formally calculated as:

(1)

In the equation:

* *rt* is the reward gained by agent at each timestep *t*
* *T* is the total number of spins conducted in the test episode

This metric gives a clear measurement of how good each model performs under the rules of the roulette game. Higher the reward better is the performance of the model.

### 3.4.2. Average Rewards per Spin

The Average Reward per Spin was also calculated so that the models could be fairly compared independent of test duration. It is calculated using the formula:

(2)

These metrics help in normalizing performance and determining the average profit or loss from each bet throughout all test spins.

### 3.4.3. Additional Metrics for PPO and DQN

In order to evaluate the behavior and convergence of reinforcement learning models, additional internal metrics were further monitored, especially during training. The additional metrics monitored are discussed below:

1. **Policy Gradient Loss (PPO):** In PPO algorithm, this metric indicates how much the policy (i.e. the decision strategy of the agent) is being updated. If the value of this metric is stable and consistently negative, then it indicates that the model is learning appropriate improvements without abrupt shifts. This aids in ensuring that training remains stable and effective (Huang et al., 2020).
2. **Entropy Loss (PPO):**  This metric measures how random the behavior of the agent is. When the entropy value is high, the agent is still exploring the patterns in the data and when the entropy is low, it becomes more certain and deterministic in its decisions. In the training process, the entropy should decrease gradually to indicate agent converging on an optimal strategy (Tim Lou, 2025; Verstraete, 2024).
3. **Value Function Loss (PPO and DQN):** This metric measures how well the model forecast rewards. A lower value of loss indicates that the model is learning properly. However, in this study, explained variance was taken into account, indicating the accuracy of value projections. Values approaching 0 or negative indicate that the value network may not be accurately reflecting the environment (Farahmand et al., 2016).
4. **Approximate KL Divergence (PPO):** This metric gives the measurement of how much the new policy is different from the old policy during the training updates. PPO tries to maintain this within a safe range (usually less than a clipping threshold), enabling steady learning and preventing overfitting or destabilization (Palenicek, 2021; Kobayashi, 2021).
5. **Q-value Estimation (DQN):** The ability of the model to forecast long-term rewards for every action is evaluated by measuring the projected Q-values for the DQN agent. Consistently unstable or overestimated Q-values might be an indication of inadequate exploration or problems with the learning dynamics (Zhang et al., 2024; Cini et al., 2020).
6. **Exploration Rate (DQN):** DQN models make use of an epsilon greedy strategy to maintain a balance between exploration and exploitation. The agent performs exploration early in the training process before settling into more predictable behavior if the exploration rate is monitored (Wang et al., 2023).

## 3.5. Implementation

The implementation of this project was done using Python programming within a Jupyter Notebook environment. This setup provided a flexible and interactive workspace which was perfect for testing behavior of the betting agents, experimenting with the reinforcement learning models and performing results visualization in real time. The rationale for choosing Python programming was that it has a robust ecosystem of machine learning and deep learning packages. Besides, it is also easy to use and has high readability (Raschka et al., 2020).

### 3.5.1. Tools and Libraries Used

The tools and libraries that have been used in the project are tabulated below:

Table 1 - Tools, Libraries, and Packages Used in the Project with Description, Rationale, and Version

| **Library/Tool** | **Description and Reason Behind Use** | **Version** |
| --- | --- | --- |
| Python | It was the core programming language used developing the codes for the different agents. | 3.12 |
| Jupyter Notebook | IDE is used for coding because it provides an interactive coding environment which is suitable for modular development and testing. | 6.5.4. |
| Gymnasium | It is a framework which provides methods for building custom reinforcement learning environments. | 1.1.1. |
| Stable Baselines3 | Reinforcement learning library which provides methods for implementing PPO and DQN agents | 2.6.0 |
| NumPy | This library was used for array operations | 1.24.4 |
| Pandas | Used for data loading | 2.0.3 |
| Matplotlib | This library was used for making necessary visualization plots | 3.7.1. |
| Seaborn | Used for improving the quality of the plots | 0.12.2 |
| Scikit-learn | For splitting the dataset into training and testing set | 1.3.0 |

Anaconda and pip were used to manage all packages, and version consistency was preserved to guarantee reproducibility.

### 3.5.2. Coding Standards Followed

To make sure that the code was consistent and written in professional manner, the following coding standards were followed:

* **PEP-8:** All Python code was written as per the guidelines provided by the official Python Enhancement Proposal 8 (PEP 8) guide. This included consistent use of indentation, proper variable and function naming, spacing, adhering to standard line lengths etc (Rossum, et al., 2001).
* **Modular Structure:** The code has been written in a modular manner. The necessary classes and functions have been created. This helped to ensure reusability and readability.
* **Proper Code Documentation:** Proper comments and docstrings were added to the code to enhance its readability. All classes, methods and functions included docstrings for proper documentation. The use of inline comments also further added to the code clarity.
* **Error Handling:** Proper error handling was added wherever necessary to enable graceful handling of code during the cases of exceptions and crashes.

This approach used in the coding process helped in streamlining the development process and also helped to minimize the chance of errors or inconsistencies.

### 3.5.3. Algorithm Steps

#### 3.5.3.1. PPO Algorithm Steps

The PPO agent was implemented using the Stable Baselines3 library and trained on the simulated roulette data in a custom roulette environment. The simplified steps followed by the PPO agent are listed below:

**Step 1:** Initialize the PPO agent with policy architecture and learning parameters.

**Step 2:**  Define the custom roulette environment which consists of the state, action space, and reward system.

**Step 3:** Reset the environment and get the initial state.

**Step 4:** Perform interaction with the environment by selecting actions using the current policy.

**Step 5:** Collect the reward and the next state after each action.

**Step 6**: Store the transitions (state, action, reward, next state)

**Step 7:** Perform updates of the policy network periodically using collected experiences, with clipped objective function to avoid large updates.

**Step 8:** Repeat the interaction and training loop for a predefined number of timesteps (100,000 steps in this case).

**Step 9:** Perform evaluation of the training policy on the test environment and calculate total reward.

The use of the clipped surrogate objective of the PPO helped to ensure a stable training process and prevented policy divergence, particularly when the environment of roulette has very high-variance.

#### 3.5.3.2. DQN Algorithm Steps

DQN (Deep Q-Network) makes use of a different technique. This algorithm focuses on estimating the value of actions. The steps taken for the implementation of the DQN agent is as follows:

**Step 1:** Initialize the DQN agent with a neural network for predicting the Q-values, replay buffer, and other parameters like learning rate.

**Step 2:**  Reset the custom roulette environment developed in above PPO algorithm to start an episode..

**Step 3:** At each step, perform selection of action using an epsilon-greedy policy:

* Select a random action (i.e. exploration) with probability epsilon.
* Or else, select the action which has the highest predicted Q-value (exploitation).

**Step 4:** Perform execution of the action and look for the resulting reward and new state.

**Step 5:** Store the transitions like state, action, reward and next state in the replay buffer

**Step 6**: Perform random sampling of mini-batches from the buffer and use them to train the network by minimizing the difference between predicted and target Q-values.

**Step 7:** Perform update of the target network at fixed intervals. This helps to stabilize the training process.

**Step 8:** For the requirement number of timesteps (100,000 steps in this study), repeat this loop

**Step 9:** Throughout the episode, test the trained agent and record its total reward.

# CHAPTER 4. RESULTS

This section presents the results obtained from evaluating various roulette betting strategies. This included classical progression systems and a reinforcement learning agent which is trained using the Proximal Policy Optimization (PPO) algorithms. All the experiments were conducted using the historical roulette dataset obtained via realistic roulette simulations. The primary objective was to evaluate the agent's ability to maximize long-term profits while adhering to real-world betting constraints.

## 4.1. PPO Agent Performance

The PPO agent was trained over 100,352 timesteps, with a total of 49 iterations and 480 policy updates. Various training stability indicators were monitored. These indicators included KL divergence (≈0.012–0.015), policy gradient loss (≈–0.037 to –0.044), and entropy decay (from –2.89 to –2.62) which showed somewhat healthy optimization behavior. But, the explained variance remained slightly negative This suggested that the model was struggling with accurately predicting returns.

|  |  |
| --- | --- |
|  |  |

Figure 4: Metrics from the last four training steps of the PPO agent

The test evaluation performed on the test dataset resulted in following outcome:

Table: Test results from the PPO agent

| **Total Rounds Played** | 12,732 |
| --- | --- |
| **Total Cumulative Reward** | -278.50 |

While PPO showed learning stability, it still struggled to achieve positive returns. This is likely due to the high-variance and memory-less nature of the roulette game, where even intelligent exploration fails to find the patterns and gain competitive edge.

## 4.2. Classical Strategy Comparison

To benchmark the PPO agent, four traditional strategies were implemented. Fibonacci, D'Alembert, Martingale, and a Random Baseline were all included. Every Strategy was subjected to the identical environmental rules and placed a constant bet on red.

### 4.2.1. Martingale Strategy

The exponential bet growth worked briefly but it then collapsed during the losing streaks. Surprisingly, with this technique the agent was able to play longer rounds than the PPO algorithm and also the total loss at the end was lower than the PPO algorithm. The test results for this agent is provided below:

Table: Test results from the Martingale strategy

| **Total Rounds Played** | 15,660 |
| --- | --- |
| **Total Cumulative Reward** | –97.40 |

### 4.2.2. Fibonacci Strategy

The slower linear progression helped to control the risk better than Martingale. However, this technique also ended in the loss. The number of rounds played by this strategy was comparatively lower in number as compared to the Martingale approach. The test results are provided below:

Table: Test results from the Fibonacci strategy

| **Total Rounds Played** | 2,018 |
| --- | --- |
| **Total Cumulative Reward** | –31.10 |

### 4.2.3. D’Alembert Strategy

The results from this strategy were also similar to that of the Fibonacci strategy. In fact it was slightly better. The linear progression proved the most resilient, offering the smallest cumulative loss. The test results are presented below:

Table: Test results from the D’Alembert Strategy

| **Total Rounds Played** | 2,047 |
| --- | --- |
| **Total Cumulative Reward** | –21.80 |

### 4.2.4. Random Strategy

This strategy acted as a baseline, placing bets randomly without any logic or memory. Since the approach is unstructured, it led to the worst performance as expected and resulted in the highest cumulative loss among all agents. This result emphasized the ineffectiveness of roulette betting that is only based on chance and supports the need of even basic strategic approaches. The test results are presented below:

Table: Test results from the Random Strategy

| **Total Rounds Played** | 12,163 |
| --- | --- |
| **Total Cumulative Reward** | –648.00 |

## 4.3. DQN Strategy

The DQN agent was another reinforcement learning based model that was trained using a value-based reinforcement learning approach over 100000 steps. This agent also attempted to learn Q-values for roulette betting actions via trail-and-error exploration. This agent also showed stable training with exploration rate reaching around 0.05 and loss decreasing to 0.44. However, the agent was also unable to develop a successful policy.

On testing, the agent terminated quickly after only 1,198 steps, accumulating a moderate loss. This performance of this agent also lagged behind all the classical strategies. The test results for this agent is provided below:

Table: Test results from the DQN agent

| **Total Rounds Played** | 1,198 |
| --- | --- |
| **Total Cumulative Reward** | –126.90 |

## 

## 

## 4.4. Comparative Summary

Table: Comparative summary of the test results

| **Strategy** | **Rounds Played** | **Total Reward** | **Outcome** |
| --- | --- | --- | --- |
| PPO (RL) | 12,732 | –278.50 | Terminated (Max Loss) |
| Martingale | 15,660 | –97.40 | Terminated (Max Loss) |
| Fibonacci | 2,018 | –31.10 | Terminated (Max Loss) |
| D’Alembert | 2,047 | –21.80 | Terminated (Max Loss) |
| Random | 12,163 | –648.00 | Terminated (Max Loss) |

The plot of the total reward for each of the technique is provided below:

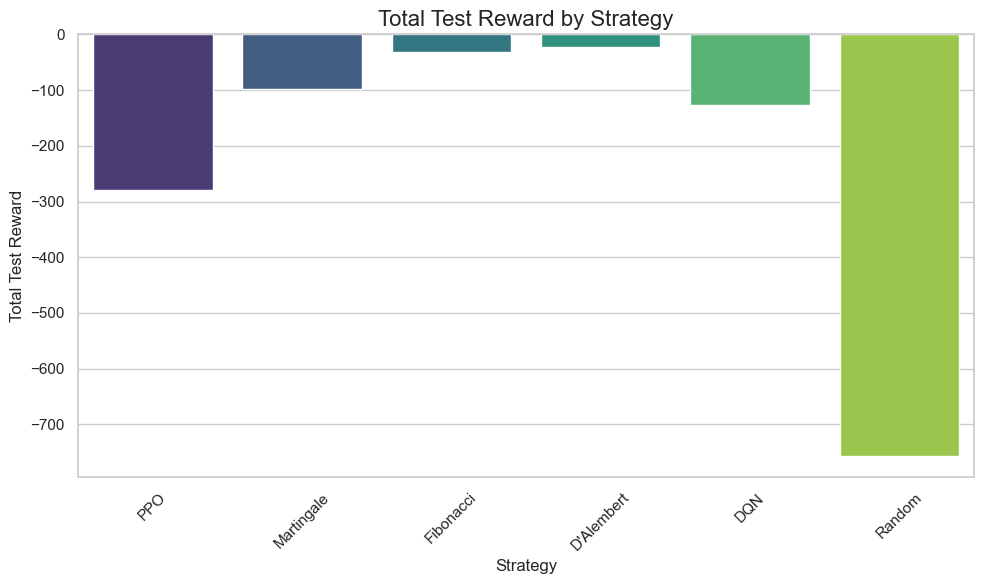


Figure 5: Plot of the total reward for each techniques

The plot shows that all the techniques incur losses. However, the classical techniques like Fibonacci and D'Alembert seem to incur minimal overall loss.

The bar graph of the total rounds played by all techniques are given below:

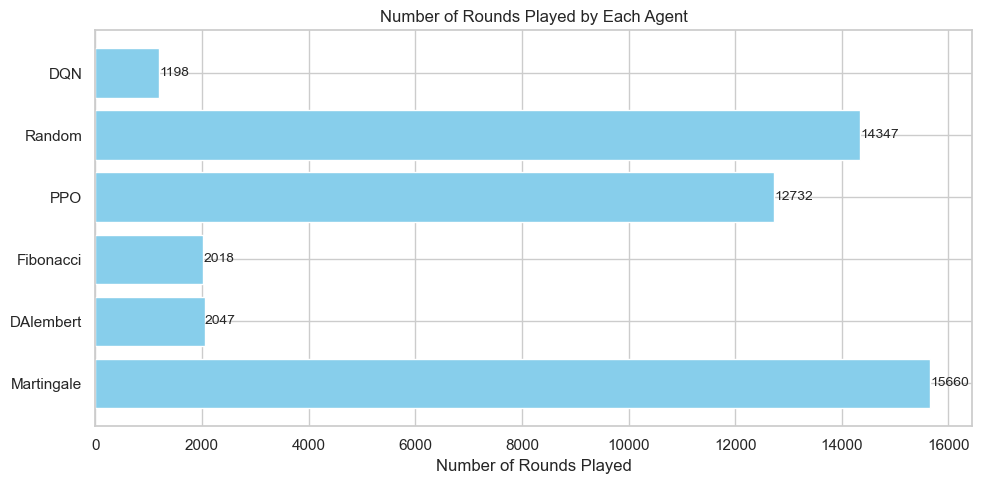


Figure 6: Bar Graph showing the number of spins each technique played before termination

The bar chart compares the number of rounds played by each agent before termination. It highlights the behavioral characteristics and risk profiles of different betting strategies. From the graph it is clear that Martingale strategy survived the longest (15,660 rounds). This may be because of its aggressive doubling strategy, which enables quick loss recovery but eventually causes collapse over protracted losing streaks (Victor, 2015). One of the surprising findings of this experiment was that the Random agent also lasted quite long (14,347 rounds). This might be due to its flat betting structure, which slows down the rate of loss despite lacking any intelligence or pattern.

The PPO agent was able to manage a moderate number of plays of around 12,732 rounds. This indicates that it likely converged to conservative betting behaviors which avoided high-risk actions. However, it was still not able to consistently gain an advantage in a roulette environment. In comparison to this, Fibonacci and D’Alembert both make use of the mild progression systems (Ethier, 2010). These terminated earlier (around 2,000 rounds), as their steady increase in stakes still led to eventual losses under unfavorable streaks.

The DQN agent had the shortest number of plays (1,198 rounds). This reflects its difficulty in learning a value-based policy in a game with no exploitable patterns or predictable structure. This result highlights the limitations of reinforcement learning in high-variance, chance-driven situations like roulette, where complex algorithms fail to beat simple, rule-based methods in terms of survival or profitability.

For more normalized comparison, the average reward per spin was also calculated and the bar graph was plotted which can be seen below:

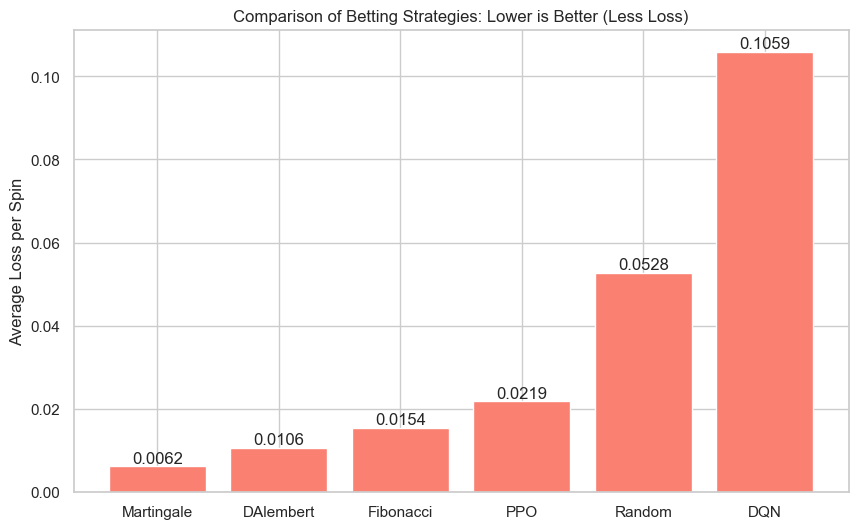


Figure 7: Average reward per spin for all techniques

The average reward for each round shows that Martingale performed best (-0.0062), with the smallest loss every spin, followed by D'Alembert and Fibonacci technique, which also managed losses efficiently. PPO (-0.0218) performed moderately, outperforming Random and DQN but underperforming traditional techniques. DQN got the weakest score (-0.1059), demonstrating a challenge in a stochastic environment. Overall, structured classical techniques outperformed both RL agents and random betting when it came to reward efficiency.

# CHAPTER 5. PROJECT MANAGEMENT

## 5.1. Project Schedule

To successfully accomplish the primary objectives of this project, which was mainly to build and evaluate a Proximal Policy Optimization(PPO) algorithm applied for optimizing betting strategies in roulette, a properly planned and detailed project schedule was developed at the beginning of the project. The project was structured into different stages with measurable milestones. For structuring the project tools like Gantt chart were used. The gantt chart for the project is provided below:

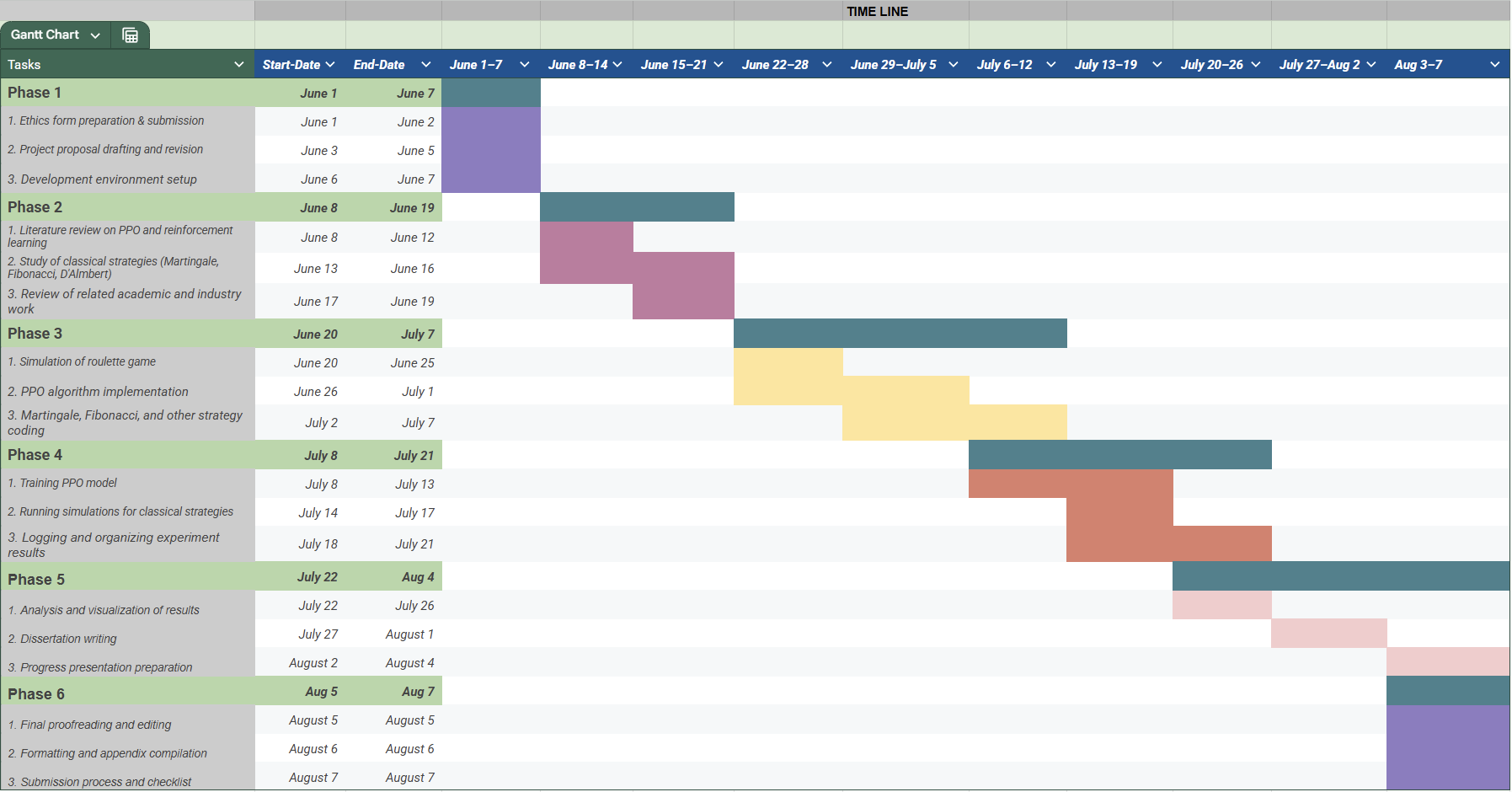


Figure 8: Gantt chart showing the project timelines created during planning phase

Each of the tasks were allocated with a fixed start date and end date as seen from the Gantt chart. The aim was to adhere strictly to this timeline to complete the project within the given deadline. However, during the project, a few minor deviations from the planned time line occurred. The training and optimization phase of the PPO algorithm required additional time because of the challenges in attaining consistent PPO performance. Time was carefully reallocated without compromising the final deliverables. Weekly meetings with the supervisor were conducted to review milestones, debug and correct issues, and refine project objectives. These meetings with the supervisor helped to ensure regular feedback and direction, which proved very important in meeting project goals on time.

## 5.2. Risk Management

Risk management is one of the most important parts of project planning to identify and handle any potential disruptions which could occur during the course of the project. Because the data and environment in this dissertation research are self-contained and reproducible, the risks were comparatively minimal. However, there were few technical and logistical risks which were identified:

Table: Risk register for the dissertation project.

| **S.N** | **Risks** | **Likelihood** | **Impact** | **Implemented Strategy for Mitigation** |
| --- | --- | --- | --- | --- |
| 1 | Simulation code or model files corruption | Low | Medium | Maintain backups of the codes by saving in Google drives and Github. |
| 2 | Hardware failure or system crash | Medium | Medium | Backup all the files and generated data into an external drive. |
| 3 | Extended model training time | Medium | High | Using smaller batch sizes and training steps can reduce the number of episodes. |
| 4 | Implementation bugs or PPO convergence issues | Medium | Medium | The components needed to be validated incrementally and loggings and visualization tools can be used for debugging. |

The most significant challenge which was encountered during the project was the problems related to the algorithm convergence. The training of the PPO model was sensitive to hyperparameters like learning rate and clipping range. During some of the initial training, it was found that the policy updates were unstable and the reward allocation was inconsistent. Handling this issue required iterative adjustments and additional experimentation. To manage this, open-source PPO implementations were studied, necessary adjustments were done as per the knowledge gained from the study and training process was thoroughly validated to ensure the correctness of the implementation.

Besides, no other risks materialized to disrupt the timeline. However, there were minor delays in the project timeline because of the tuning challenges.In general, proactive risk management supported the preservation of project quality and continuity.

## 5.3. Quality Management

One of the main focuses was to make sure that the quality of the implementation and overall research process was maintained throughout the course of this project. To make sure that the quality of the project was maintained, various quality assurance measures were adopted:

1. Regular Supervisor Feedback: Meetings with the supervisor were conducted on a weekly basis which provides ongoing review and constructive feedback on the algorithm design and experimental approach. These feedbacks from the supervisor helped to identify the issues early and optimize and revise the work continuously.
2. Code Review and Testing: Modular approach was opted for implementing the PPO and other algorithms. The simulation codes were also modularized. All the codes written were tested thoroughly to ensure that the functionality of the algorithms were correct. Standard protocols were followed to train and evaluate the models like splitting training and testing set in a ratio of 7:3, performing data encoding and necessary enrichments.
3. Documentation: For the codebase proper and detailed documentation were maintained which included commenting the codes, using docstrings to provide information of the modules and functions used in the code, documenting package requirements in the requirements.txt, maintaining README files describing the setup, usage and configuration parameters, documenting the findings and methodology in text cells of the jupyter notebooks etc.
4. Version control: Git was used as a tool for version control for tracking changes in the code updates and reverting back to stable versions in cases when the code failed.
5. Reproducibility: Experiments were properly logged with exact parameters and random seeds. This helped to enable replication of results.

## 5.4. Ethical, Legal and Professional Considerations

This project properly follows all relevant ethical, legal, and professional guidelines. Since the project made use of the synthetic roulette spin dataset which was generated via simulation rather than real user or sensitive data, there were not many considerations needed towards privacy and confidentiality.

**Data Ethics:** The roulette game was created programmatically by making use of well-understood probability distributions and casino rules. No personal or sensitive data was involved during the course of this project. This eliminated risk related to data misuse of bias.

**Legal Compliance:** The simulation and algorithms developed for the project were solely for academic purposes and did not involve gaming with real money. Thus, this avoids legal restrictions which are related to gambling activities. Apart from this, all the tools used for developing the algorithms and simulations were open-source tools and didn’t require any licensing which further mitigated the legal risks related licensing of the necessary software tools.

**Professional Integrity:** All sources that were used for performing background research, implementing algorithms, and comparative techniques were properly cited. This helped to maintain academic honesty. The work was carried out independently while adhering to university regulations and receiving the proper supervision.

For the project, the ethical approval was obtained as a part of the research oversight process of the university. The documentation is included in the Appendix. Regular consultations with the supervisor help to ensure alignment with the professional standards.

# CHAPTER 6. DISCUSSION

The experiment performed in this dissertation aimed to evaluate and optimize roulette betting strategies using the PPO algorithm and benchmark it performance by comparing it with other reinforcement learning technique like DQN and classical betting strategies like Martingale, Fibonacci, D’Almenbert and Random approach. The main objective was to perform thorough assessment of the effectiveness of these strategies in maximizing long-term rewards while also adhering to practical betting constraints. The results of the experiment clearly demonstrated that despite the sophistication of reinforcement learning techniques, classical betting systems still showed more robust performance in terms of risk management, loss minimization and round longevity.

## 6.1. Structural Incompatibility Between RL and Roulette

From the experiment, fundamental issues were revealed by the underperformance of PPO and DQN. The issue is that RL, by design, requires either temporal correlations or exploitable feedback loops for learning effective policies. However, these are not available in case of the Roulette. Each of the spins in the Roulette is independent and this independence of spins removes any sequential advantage. Because of this the RL agents cannot learn effectively from the past outcomes, which makes experience-based learning mostly ineffective. Despite the fact that PPO was able to maintain stable training dynamics, its policy optimization tended to favor conservative, loss-minimizing behaviors over properly profitable ones. The algorithm didn’t seem to fail from the technical perspective, it simply had nothing learnable from the stochastic data from Roulette to optimize the agent.

DQN also performed poorly. Unlike PPO, it relied on Q-value estimates, which is practically worthless in a situation where no activities consistently result in increased future value. This demonstrates a wider conceptual limitation: value-based RL algorithms are not suitable for flat, reward-agnostic settings. Even deep approximators don't identify any structure where none exists.

Thus, the limitations observed in this study clearly suggests that using RL algorithms for optimizing the game of pure chance may not yield good results and doesn’t seem to depend on what algorithm tuning approach is used or how long the model is trained. The results indicate domain misalignment.

## 6.2. Illusion of Complexity vs. Practical Resilience

One of the key findings of this dissertation is that, in many situations, simpler classical techniques showed superior reward efficiency and survival, even if they were mechanically simple. This does not mean that these systems are inherently effective, but it does draw significant attention to an important nuance: strategy robustness is not the same as intelligence.

Classical systems, especially those with linear or moderate progression (e.g., D’Alembert), have been designed for managing risks in a structured way. While these techniques have issues and flaws mathematically over the long term, they provide consistent, bounded behaviors which can outperform ML models which are constrained by limited state observability and lack of pattern detectability.

This brings up an important point: Rl agents which are not explicitly limited by domain knowledge may default to overfitting randomness, while simpler heuristics which, ironically, ignore the “learning” process, might demonstrate better resilience. Therefore, the failure of the RL agents may not be due to the implementation, but rather to misguided adaptation in a system which penalizes the identification of false pattern recognition.

## 6.3. Survival is not Equal to Success

Survival time is the amount of time a strategy can play before it suffers from huge loss leading to termination. It is one of the topics that receives a lot of attention in gambling strategy research (Chapman & Getzen, 2011). This statistic, nevertheless, may be deceptive. A longer survival time does not always indicate a superior plan, especially if it masks an impending collapse.

In this study, the Martingale system survived the longest number of episodes or spins. The technique was able to achieve this via the aggressive risk exposure that ultimately guaranteed loss. Apart from this, the PPO agent also showed moderately long playtime. This reflects risk aversion rather than strategic superiority of the algorithm. One of the main understanding from the experiment was also that, measuring performance only using longevity may confuse failure delay with strategic value. Thus, it is necessary to create a more critical approach like normalized metrics (e.g. reward per spin) and qualitative behavior in a more analytical manner. It is necessary to understand, if the strategy changes significantly, or is it only surviving via statistical noise or bankroll padding.

## 6.4. Benchmarking RL in Unrewarding or Stochastic Environments

Through the experiment it was found that RL algorithms like PPO and DQN performed worse than the classical techniques. It was sometimes found that the DQN sometimes performed even worse than random strategy. This raises important questions about how we perform evaluation of the RL in environments like Roulette. In such unpredictable settings, common performance measures like total reward seem less useful.

When every action leads to similar negative outcomes, it is very difficult for the RL agents to tell whether a policy to which they converge is actually good or just randomly lucky. This findings of the dissertation clearly highlights a key challenge: RL methods require environments where actions can lead to some meaningful differences in the results. This work adds to the increasing amount of evidence that certain domains are inherently unsuited for reward-driven learning and that RL assessment has to be environment-aware.

## 6.5. False Pattern Detection and Overfitting

One of the key limitations of the RL agents is that they work on the assumptions that the winning pattern can be found. This assumption works well in games which have definite patterns like Go or Poker. But in roulette, which is meant to be purely random, trying to find patterns in noise might lead to misdirected policy updates.

As a result, the agents in such cases might overfit to short-term streaks or become very cautious avoiding to take actions or moves that are very risky. Due to this, the agents may converge to a risk-averse but ineffective strategy. In this instance, the RL models fail as there is nothing valuable to learn from the data. This is very important distinction, especially for individuals who could interpret these findings as a technological issue rather than a more serious issue with applying RL to situations which depend on chance, like roulette.,

## 6.6. Broader Implications for Algorithmic Betting

The more general implication is that, in absence of the structural bias or external knowledge, reinforcement learning is not well adapted to zero-skill gambling settings. The findings clearly imply that no amount of learning can make up for the negative expected value in fair games like roulette. In actuality, a performance of the system may suffer the more it tries to adjust to noise.

Thus, in a nutshell, this study not only performs critical analysis of the effectiveness of RL in gambling but also suggests caution against the broader trend of overextending machine learning into domains where the fundamental limitations of the environment makes the adaptation of the ML techniques meaningless.

# CHAPTER 7. CONCLUSION

## 7.1. Conclusion

This dissertation performed an investigation of whether modern reinforcement learning techniques, particularly the Proximal Policy Optimization (PPO) could be used for developing viable betting strategies in a gambling environment which is fully stochastic and memoryless like European Roulette. The PPO performance was compared with a value-based RL model like DQN and also with several traditional rule-based betting strategies.

The findings reveal several important insights:

1. Reinforcement learning agents, especially PPO, were able to show stable learning behaviors, but they struggled to achieve positive results because of the stochastic and high-variance nature of roulette. Despite being technically sound, these agents could not extract meaningful structure from random data, thus this emphasizes a misalignment between environmental characteristics and algorithmic expectations.
2. Traditional betting systems, especially D'Alembert and Fibonacci, showed better risk-adjusted performance. Although they also generated net losses (a common occurrence in negative-expectation games), their average reward per spin and survival duration were superior to that of RL agents.
3. The performance of PPO was better than the DQN and Random betting approaches. This shows some signs of conservative and risk-averse betting patterns of the algorithm. But its performance was still lower than the traditional betting techniques in terms of the cumulative reward. This indicates that intelligent learning alone cannot compensate for structural randomness.
4. Overfitting of the noise was seen in RL models, which misinterpreted random streaks as patterns. This raises a question whether learning algorithms should be used in settings without long-term dependencies or predictable feedback.
5. Overall, the study concludes that reinforcement learning even in its most robust form (i.e. PPO), is not well suitable when it comes to optimizing the strategies in games that are purely driven by chance like European Roulette. The agents which learnt to rescue risks and loss outperformed those which looked for improving the predictive value.

## 7.2. Limitations of the Study

Despite the rigorous design and execution of the experiment, this study had some of the limitations. The limitations of the study are discussed below:

1. Synthetic Environment: While the simulation of European Roulette was validated statistically using the different visualizations and chi-squared test, it still remains a model of the real-world system. Real-word casino behaviors, biases or subtle irregularities were not considered in this study.
2. Limited Action Abstractions: Despite being extensive, the action space lacked sophisticated meta-strategies that may have reduced losses, such as conditional exit strategies or dynamic risk scaling.
3. Training Time Limitations: Because of limited computing power, the PPO and DQN agents were only trained for a set number of steps (100,000). Training these models for a larger number of steps might improve the performance of the models. However, it is very unlikely to make a big difference in a game like roulette.

## 7.3. Future Enhancements

The future enhancements of the dissertation are provided below:

1. Hybrid models with Rule Integration

In future, the focus can be put on building the hybrid models which combine reinforcement learning with rule-based systems. For instance, integrating classical betting strategies like Martingale within PPO or adding safety layers which can enforce bankroll management could improve the survivability of the RL agent.

1. Use of Meta-Actions and Hierarchical RL

In future versions of the experiment, meta-actions such as “exit”, “switch strategy” etc. could be added which can enable agents to simulate higher-level decision-making. By making use of the Hierarchical reinforcement Learning (HRL) frameworks can also offer a more meaningful structure in the stochastic environments (Pateria, et al., 2021).

1. Safe and Risk-Aware RL Frameworks

The PPO algorithm can be extended using the Safe RL methods like CVaR-PPO or Constrained policy Optimization (CPO) techniques (Mead et al., 2025; Achiam et al., 2017). These techniques can help in better modeling risk aversion and bankroll preservation, which are important in gambling environments.

1. Exploring Alternative RL Algorithms

In future, other model-based RL, evolutionary strategies like Genetic Algorithms etc could be explored and it might provide further new insights.

1. Expansion to Other Gambling or Financial Domains

The knowledge gained from this research may be used in other stochastic domains, such as bitcoin trading, sports betting, or stock market simulation, where reinforcement learning may discover more structural patterns and be more applicable.

# CHAPTER 8. STUDENT REFLECTIONS

Working on this dissertation study has been a very rewarding experience for me. This dissertation helped me to improve my technical skills, specifically it improved my Python programming skills and skills related to development of the AI models. Besides, technical skills in this dissertation also helped me improve my problem solving approach. My main goal was to see if modern reinforcement learning methods like PPO and DQN could be used to improve betting in the game of roulette. Along the way, I also looked at traditional strategies like Martingale and Fibonacci to compare how well each one performed when it came to minimizing the loss.

The idea began with a simple question: can artificial intelligence outperform classic betting systems and yield better results? Since roulette is a game of chance and highly stochastic, I knew from the very beginning that it wouldn’t be easy to find patterns in the roulette data. But I was very curious to see if reinforcement learning agents could still learn to make better decisions over time, even with such random outcomes. Besides my curiosity, the other motivation behind carrying out this dissertation on this topic was that there was not much literature or research done in the past on this topic. Thus, I felt like this work of mine would provide the base or foundation for future researchers.

Using Python and the Gymnasium module to create my own roulette simulation was one of the most educational experiences I've had. Using Python and the Gymnasium module to create my own roulette simulation was one of the most educational experiences I've had. I had to create a configuration that adhered to actual casino rules, such as putting restrictions on the agent's maximum bet or the number of consecutive times it may double the amount bet. This made me realize how crucial it is to properly plan the reinforcement learning environment since even minor adjustments can have a big impact on the entire training process.

In addition to this, training the PPO and DQN agents was another important step in the dissertation. PPO seemed to train more smoothly, and I could clearly see its learning progress via different metrics like policy loss and entropy. At each step, I would know how the model is doing. However, I noticed that the model still struggled to make accurate predictions about long-term rewards. In comparison to this, DQN agents were a bit harder to train, particularly with managing things like memory buffers and exploration settings. Neither agent consistently outperforms the more simple betting strategies, even after experimenting with various parameters and running the models for thousands of steps.

One interesting outcome was that strategies like Martingale and Fibonacci performed comparatively better than the intelligent agents. This was a very surprising finding for me. Even though they are very simple and don’t rely on learning from data, they lost less money on average compared to the AI Models. This made me realize that in cases, especially in betting games where games are driven by luck, simple human-made strategies can be just as effective or even better than machine learning approaches.

This dissertation also made me realize some of the challenges in using reinforcement learning in settings where results are very independent and unpredictable. Because each spin in roulette is independent of the previous one, the agent has limited information from which to learn. Due to this, it is very difficult for the models to get better over time. It served as a reminder that machine learning is not always the most effective solution for all the problems.

During the course of the project, I faced different challenges as well. One of the main challenges was setting up the training process, trouble shooting issues and bugs in the code, and trying to comprehend the unexpected results. I became more careful and patient as a result of these issues. I discovered how important it is to try things gradually and to be more adaptable when something doesn’t work.

Finally, writing and preparing the dissertation report helped me a lot to make sense of everything I did. It gave me a chance to reflect on the results., explain why the models performed the way they did and make a comparative assessment of these AI models with the traditional approaches in a very clear, transparent and honest way. It also increased my understanding of how crucial it is to communicate research in a way that other people can comprehend and expand upon.

In conclusion, this dissertation project helped me gain much more than just academic credit. It helped to improve my technical coding skills and also my analytical and problem solving skills. This also introduced me to the real-world uses of reinforcement learning, and taught me how to deal with complex unpredictable sophisticated systems. Even though the AI models experimented in this study didn’t outperform the traditional approach at the end, I gained a much more deeper understanding of machine learning which I think is my biggest achievement of this dissertation.

# REFERENCES

Achiam, J., Held, D., Tamar, A., & Abbeel, P. (2017, July). Constrained policy optimization. In *International conference on machine learning* (pp. 22-31). PMLR.

Arend, R. J. (2024). A New Typology of Uncertainty (for Decision-Making). In *Uncertainty in Strategic Decision Making: Analysis, Categorization, Causation and Resolution* (pp. 247-312). Cham: Springer Nature Switzerland.

Bilgin, E. (2020). *Mastering reinforcement learning with python: build next-generation, self-learning models using reinforcement learning techniques and best practices*. Packt publishing ltd.

Blavatskyy, P. (2024). A puzzle of roulette gambling. *Journal of Risk and Uncertainty*, *69*(2), 219-234.

Chapman, Z., & Getzen, T. (2011). Bet doubling in gambling and investing. *The Journal of Risk Finance*, *12*(4), 291-305.

Cini, A., D'Eramo, C., Peters, J., & Alippi, C. (2020). Deep reinforcement learning with weighted Q-Learning. *arXiv preprint arXiv:2003.09280*.

Del Rio, A., Jimenez, D., & Serrano, J. (2024). Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments. *IEEE Access*.

Ethier, S. N. (2010). Betting Systems. In *The Doctrine of Chances: Probabilistic Aspects of Gambling* (pp. 275-316). Berlin, Heidelberg: Springer Berlin Heidelberg.

Farahmand, A. M., Barreto, A. M., & Nikovski, D. N. (2016, December). Value-aware loss function for model learning in reinforcement learning. In *13th European Workshop on Reinforcement Learning (EWRL)* (Vol. 1, No. 2, p. 36).

GGpoker. (2025). *Roulette - Understanding Roulette*. Welcome to LWC communities! https://help.ggpoker.com/article/Roleta-Entendendo-a-Roleta-1717475260871#:~:text=European%20Roulette%20is%20a%20classic,ball%20will%20land%20on%20correctly.

Hassanniakalager, A., & Newall, P. W. (2022). A machine learning perspective on responsible gambling. *Behavioural Public Policy*, *6*(2), 237-260.

Hu, R., & Lauriere, M. (2023). Recent developments in machine learning methods for stochastic control and games. *arXiv preprint arXiv:2303.10257*.

Huang, R., Yu, T., Ding, Z., & Zhang, S. (2020). Policy gradient. In *Deep Reinforcement Learning: Fundamentals, Research and Applications* (pp. 161-212). Singapore: Springer Singapore.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255-260.

Kleshnina, M., Hilbe, C., Šimsa, Š., Chatterjee, K., & Nowak, M. A. (2023). The effect of environmental information on evolution of cooperation in stochastic games. *Nature Communications*, *14*(1), 4153.

Kobayashi, T. (2021, May). Proximal policy optimization with relative pearson divergence. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 8416-8421). IEEE.

Mead, H., Costen, C., Lacerda, B., & Hawes, N. (2025). Return Capping: Sample-Efficient CVaR Policy Gradient Optimisation. *arXiv preprint arXiv:2504.20887*.

Muslimani, C., Johnstonbaugh, K., Chandramouli, S., Booth, S., Knox, W. B., & Taylor, M. E. (2025). Towards Improving Reward Design in RL: A Reward Alignment Metric for RL Practitioners. *arXiv preprint arXiv:2503.05996*.

Palenicek, D. (2021). A survey on constraining policy updates using the KL divergence. In *Reinforcement learning algorithms: Analysis and applications* (pp. 49-57). Cham: Springer International Publishing.

Pateria, S., Subagdja, B., Tan, A. H., & Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, *54*(5), 1-35.

Rossum, G. van, Warsaw, B., & Coghlan, A. (2001, July 5). *PEP 8 – style guide for python code*. Python Enhancement Proposals (PEPs). https://peps.python.org/pep-0008/

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Souchleris, K., Sidiropoulos, G. K., & Papakostas, G. A. (2023). Reinforcement learning in game industry—Review, prospects and challenges. *Applied Sciences*, *13*(4), 2443.

Shakya, A. K., Pillai, G., & Chakrabarty, S. (2023). Reinforcement learning algorithms: A brief survey. *Expert Systems with Applications*, *231*, 120495.

Small, M., & Tse, C. K. (2012). Predicting the outcome of roulette. *Chaos: an interdisciplinary journal of nonlinear science*, *22*(3).

Tim Lou, P. (2025, January 19). *What does entropy measure? an intuitive explanation*. Towards Data Science. https://towardsdatascience.com/what-does-entropy-measure-an-intuitive-explanation-a7f7e5d16421/

Turner, N. E. (1998). Doubling vs. constant bets as strategies for gambling. *Journal of Gambling Studies*, *14*(4), 413-429.

Verstraete, F. (2025, April 14). *Mastering proximal policy optimization (PPO) in Reinforcement learning with code*. Medium. <https://medium.com/@felix.verstraete/mastering-proximal-policy-optimization-ppo-in-reinforcement-learning-230bbdb7e5e7>

Victor, O. O. (2015). *Martingale Theory* (Master's thesis, Eastern Mediterranean University (EMU)-Doğu Akdeniz Üniversitesi (DAÜ)).

Vinitsky, E., Du, Y., Parvate, K., Jang, K., Abbeel, P., & Bayen, A. (2020). Robust reinforcement learning using adversarial populations. *arXiv preprint arXiv:2008.01825*.

Wang, X., Yang, Z., Chen, G., & Liu, Y. (2023). A reinforcement learning method of solving Markov decision processes: an adaptive exploration model based on temporal difference error. *Electronics*, *12*(19), 4176.

Zhang, H., Chen, H., Xiao, C., Li, B., Liu, M., Boning, D., & Hsieh, C. J. (2020). Robust deep reinforcement learning against adversarial perturbations on state observations. *Advances in Neural Information Processing Systems*, *33*, 21024-21037.

Zhang, Y., Liu, J., Li, C., Niu, Y., Yang, Y., Liu, Y., & Ouyang, W. (2024, March). A perspective of q-value estimation on offline-to-online reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 38, No. 15, pp. 16908-16916).

Zhang, J., Zhang, Z., Han, S., & Lü, S. (2022). Proximal policy optimization via enhanced exploration efficiency. *Information Sciences*, *609*, 750-765.

Zhu, W., & Rosendo, A. (2021). A functional clipping approach for policy optimization algorithms. *IEEE Access*, *9*, 96056-96063.