

**College of Engineering, Environment, and Computing School of Science**

**MSc Data Science**

EECT 109

7150 CEM Data Science Project

## Project Report

**Optimizing Roulette with Proximal Policy Optimization: A Reinforcement Learning Approach**

By

**Srinath Kaithoju**

**SID: 15196796**

## 1st Supervisor:Dr.Babak Jamshidi

## 2nd Supervisor:Dr. Daniyal Haidar

Submitted in partial fulfilment of the requirements for the Degree of Master of Science in Data Science

Academic Year: 2024/25

**Declaration of Originality**

I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

**Statement of copyright**

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialize products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information please see [www.coventry.ac.uk/ipr](http://www.coventry.ac.uk/ipr) or contact [ipr@coventry.ac.uk](mailto:ipr@coventry.ac.uk).

**Statement of ethical engagement**

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking).

Signed:

Please complete all fields.

| First Name: | **Srinath** |
| --- | --- |
| Last Name: | **Kaithoju** |
| Student ID number | **15196796** |
| Ethics Application Number | **P187798** |
| 1st Supervisor Name | **Dr. Babak Jamshidi** |
| 2nd Supervisor Name | **Dr. Daniyal Haidar** |

# **CERTIFICATE OF ETHICS APPROVAL**

# **Certificate-187798_SrinathKaithoju.jpg**

# 

**Abstract:** This dissertation investigates the application of Proximal Policy Optimization (PPO), a state-of-the art reinforcement learning (RL) algorithm, to the fully stochastic environment of European Roulette. The aim of this dissertation is to determine whether RL based intelligent agents trained under realistic betting constraints can outperform the traditional strategies like Martingale Strategy, Fibonacci Strategy, and D’Alembert Strategy. A customized roulette simulator was created to produce statistically verified datasets for training and testing PPO, Deep Q-Networks (DQN), and benchmark techniques. For evaluating the performance of the algorithms, the metrics like total reward, average reward per spin, and survival duration were monitored. From the experimental results, it was found that the PPO demonstrated stable training and conservative betting patterns. However, it failed to achieve positive return because of the memory less and negative expected value of the Roulette game. Classical betting techniques, especially, D’Alembert Strategy and Fibonacci Strategy, outperformed RL agents in risk-adjusted performance. This finding highlighted a fundamental misalignment between RL assumptions and purely random gambling domains. The study comes to the conclusion that rule-based systems could outperform complex learning algorithms in settings that lack exploitable patterns. It also suggests hybrid models, hierarchical reinforcement learning, and risk-aware frameworks for further research.

**Keywords:**  Proximal Policy Optimization, Reinforcement Learning, European Roulette, Betting Strategies, Stochastic Environments

[Project Report 1](#_Toc206098433)

[1st Supervisor:Dr.Babak Jamshidi 1](#_Toc206098434)

[2nd Supervisor:Dr. Daniyal Haidar 1](#_Toc206098435)

[CHAPTER 1: INTRODUCTION 1](#_Toc206098436)

[1.1 Background 1](#_Toc206098437)

[1.2. Problem Statement and Motivation 2](#_Toc206098438)

[1.3. Project Aim and Objectives 3](#_Toc206098439)

[1.3.1. Aim 3](#_Toc206098440)

[1.3.2. Objectives 3](#_Toc206098441)

[1.4. Overview of the Report 4](#_Toc206098442)

[CHAPTER 2: LITERATURE REVIEW 5](#_Toc206098443)

[2.1. Introduction to Reinforcement Learning 5](#_Toc206098444)

[2.2. Reinforcement Learning in Stochastic Environments & Games 5](#_Toc206098445)

[2.2.1 RL in Games 5](#_Toc206098446)

[2.2.2. RL in Stochastic Environment 6](#_Toc206098447)

[2.3. Proximal Policy Optimization (PPO): Strengths & Limitations 6](#_Toc206098448)

[2.3.1. Exploration Enhancements in PPO 7](#_Toc206098449)

[2.3.2. Variance Control and Stability in Policy Updates 7](#_Toc206098450)

[2.3.3. Sample Efficiency and Limitations of PPO in Low-Data Regimes 8](#_Toc206098451)

[2.3.4. Challenges of Exploration and Community Insights 8](#_Toc206098452)

[2.4. Reinforcement Learning in Gambling and Change-Based Games 9](#_Toc206098453)

[2.4.1. The UCLA Roulette Project 9](#_Toc206098454)

[2.4.2. WagerWin and RL in Stochastic Card Game 10](#_Toc206098455)

[2.4.3. Implication for European Roulette 10](#_Toc206098456)

[2.5. Comparison of PPO with Other Reinforcement Learning Algorithms 11](#_Toc206098457)

[2.6. Risk-Aware and Safety RL considerations 11](#_Toc206098458)

[2.7. State and Action Representations in Probability-Driven Game 12](#_Toc206098459)

[2.8. Summary of Literatures 13](#_Toc206098460)

[CHAPTER 3: METHODOLOGY 15](#_Toc206098461)

[3.1. Dataset Generation 15](#_Toc206098462)

[3.1.1. Roulette Simulator Design 15](#_Toc206098463)

[3.1.2. Data Collection process 15](#_Toc206098464)

[3.1.3. Realism and Statistical Validation 16](#_Toc206098465)

[3.2. Training and Testing Data 17](#_Toc206098466)

[3.3. Model Selection 18](#_Toc206098467)

[3.3.1. Reinforcement Learning Models 18](#_Toc206098468)

[3.3.1.1 Proximal Policy Optimization (PPO) 19](#_Toc206098469)

[3.3.1.2. Deep Q-Network (DQN) 19](#_Toc206098470)

[3.3.2. Classical Betting Strategies 20](#_Toc206098471)

[3.3.2.1. Martingale Strategy 20](#_Toc206098472)

[3.3.2.2. Fibonacci Strategy 21](#_Toc206098473)

[3.3.2.3. D’Alembert Strategy 21](#_Toc206098474)

[3.4. Evaluation Metrics 21](#_Toc206098475)

[3.4.1. Total Reward 22](#_Toc206098476)

[3.4.2. Average Rewards per Spin 22](#_Toc206098477)

[3.4.3. Additional Metrics for PPO and DQN 22](#_Toc206098478)

[3.5. Implementation 24](#_Toc206098479)

[3.5.1. Tools and Libraries Used 24](#_Toc206098480)

[3.5.2. Coding Standards Followed 25](#_Toc206098481)

[3.5.3. Algorithm Steps 25](#_Toc206098482)

[3.5.3.1. PPO Algorithm Steps 25](#_Toc206098483)

[3.5.3.2. DQN Algorithm Steps 26](#_Toc206098484)

[CHAPTER 4. RESULTS 28](#_Toc206098485)

[4.1. PPO Agent Performance 28](#_Toc206098486)

[4.2. Classical Strategy Comparison 29](#_Toc206098487)

[4.2.1. Martingale Strategy 29](#_Toc206098488)

[4.2.2. Fibonacci Strategy 30](#_Toc206098489)

[4.2.3. D’Alembert Strategy 30](#_Toc206098490)

[4.2.4. Random Strategy 30](#_Toc206098491)

[4.3. DQN Strategy 30](#_Toc206098492)

[4.4. Comparative Summary 31](#_Toc206098493)

[CHAPTER 5. DISCUSSION 34](#_Toc206098494)

[5.1. Structural Incompatibility Between RL and Roulette 34](#_Toc206098495)

[5.2. Illusion of Complexity vs. Practical Resilience 35](#_Toc206098496)

[5.3. Survival is not Equal to Success 35](#_Toc206098497)

[5.4. Benchmarking RL in Unrewarding or Stochastic Environments 36](#_Toc206098498)

[5.5. False Pattern Detection and Overfitting 36](#_Toc206098499)

[5.6. Broader Implications for Algorithmic Betting 37](#_Toc206098500)

[CHAPTER 6. CONCLUSION 38](#_Toc206098501)

[6.1. Conclusion 38](#_Toc206098502)

[6.2. Limitations of the Study 39](#_Toc206098503)

[6.3. Future Enhancements 39](#_Toc206098504)

[CHAPTER 7. STUDENT REFLECTIONS 41](#_Toc206098505)

[CHAPTER 8. PROJECT MANAGEMENT 44](#_Toc206098506)

[8.1. Project Schedule 44](#_Toc206098507)

[8.2. Risk Management 45](#_Toc206098508)

[8.3. Quality Management 46](#_Toc206098509)

[8.4. Ethical, Legal and Professional Considerations 47](#_Toc206098510)

[REFERENCES 48](#_Toc206098511)

[APPENDIX I: CODE SNIPPETS 57](#_Toc206098512)

[APPENDIX-II: PROPOSAL 63](#_Toc206098513)

[APPENDIX-III: MEETING LOGS 71](#_Toc206098514)

[APPENDIX-IV: DISSERTATION PROPOSALSLIDES 73](#_Toc206098515)

APPENDIX-V: GITHUB REPOSITORY AND ONE DRIVE LINK 74

**LIST OF TABLES AND FIGURES**

| **Table/Figure** | **Title** | **Page** |
| --- | --- | --- |
| Table 1 | Description of the features of the dataset | 15 |
| Table 2 | Number of spins used in each set | 18 |
| Table 3 | Tools, Libraries, and Packages Used in the Project | 25 |
| Table 4 | Comparative summary of the test results | 31 |
| Table 5 | Risk register for the dissertation project | 46 |
| Figure 1 | Layout of the European Roulette wheel and betting table, showing numbered pockets (0–36), color distribution, and available betting options. | 1 |
| Figure 2 | Pie chart of the color distribution of simulated roulette outcomes | 17 |
| Figure 3 | Bar chart showing parity distribution of simulated roulette outcomes | 17 |
| Figure 4 | Bar chart showing number distribution of simulated roulette outcomes | 17 |
| Figure 5 | Metrics from the last four training steps of the PPO agent | 29 |
| Figure 6 | Plot of the total reward for each technique | 32 |
| Figure 7 | Bar graph showing number of spins each technique played | 32 |
| Figure 8 | Average reward per spin for all techniques | 34 |
| Figure 9 | Gantt chart showing the project timelines | 45 |

# 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). There is a single zero pockets which is colored green, 18 pockets colored red, and 18 remaining pockets which are colored black. European Roulette is a game of change and to improve randomization, the numerals are distributed non-sequentially around the wheel. Players bet on individual numbers, groupings of numbers, or attributes like color (red/ black) or parity (odd/even). The wheel is spun with a ball on it. The ball will eventually land in one of the pockets present in the spinning wheel. This will determine the outcome of the game (Abuhanna, 2015; GGpoker, 2025). In comparison to the American Roulette, the European version offers a slightly higher anticipated return for the player due to its single zero.



Figure 1: Layout of the European Roulette wheel and betting table, showing numbered pockets (0–36), color distribution, and available betting options.

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 modeling (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 test bed for investigating the capabilities and limitations of the RLsystems. There are traditional betting techniques like Martingale Strategy, Fibonacci strategy 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 Strategy 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, hyper parameter 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 insight 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 AlphaGo & 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 algorithms 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 Roulette project by Colonnesse & 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 works 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 (Xu et al., 2025; Houthooft et al., 2016).

### 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 (Del Rio et al., 2024). To alleviate this, PPO-CMA (Covariance Matrix Adaptation PPO) has been developed. This algorithm incorporates adaptive variance control over PPO (Hämäläinen et al., 2020). In this approach, instead of having a fixed or heuristically learned variance in the action distribution of the policy, it employs a technique of modifying the covariance matrix of actions of evolutionary strategies during training and this in turn enables the agent to dynamically resize its exploration scale. This also increases convergence rates with robustness. In order to improve the convergence rates while also preserving the robustness, the agent may dynamically modify its exploration scale, increasing variance when the policy stagnates and decreasing it when it feels confident (Shen et al. 2023; Hämäläinen et al., 2020). Such adaptive variance management can be particularly useful in the cases of stochastic environments like roulette where the distribution of the reward is flat and feedback is usually noisy. In addition to this, methods like KL-penalty schedules (as employed in PPO2 and TRPO) or Trust Region based adaptations can control update magnitudes further and avoid unstable learning that is brought on by noise buildup (Palenicek, 2021; Lu et al., 2025).

### 2.3.3. Sample Efficiency and Limitations of PPO in Low-Data Regimes

PPO methods have good sample efficiency as compared to older policy-gradient methods. In order to enable repeated epochs of mini batch updates using the same trajectory data without noticeably decreasing performance, it uses clipped surrogate goals (Jin, 2025; Rahman, 2024). This speeds up learning in high-sample environments like MuJoCo and Atari (Jin, 2025; Flet-Berliac, 2021). However, there are various studies which have demonstrated that in case of the limited data or noisy environments, model-based or dynamic programming (DP) approaches may outperform PPO (Wu et al., 2023; Dogru et al., 2024). For instance, the dependence of PPO on bootstrapped advantages may result in biased updates, and value estimation becomes incorrect in contexts where there is low-signal and extremely stochastic feedback (Rahman & Xue, 2022; Li et al., 2024). In comparison to this, the model-based methods which use planning to evaluate policies (for example, Bellman equations) or explicitly makes estimation of the transition probabilities may give better performance in cases of the low-data environments (Young et al., 2022). However, these model-based techniques have higher computational complexity and less generalization across state spaces (Rai & Arun, 2002).

Thus, this dichotomy highlights the importance of tuning the parameters of PPO like batch size, learning-rate and update frequency to sparsity and stochasticity of the environment. Over fitting, policy collapse or exploitation may result from failing to accomplish this.

### 2.3.4. Challenges of Exploration and Community Insights

There are various community feedbacks and experimental research studies which have reported that entropy regularization, policy clipping thresholds, and variance tuning are very important hyper parameters in PPO, particularly in the field with unpredictable dynamics like gambling or probabilistic simulations (Rahman, 2025; Liu et al., 2019). The PPO algorithm can converge prematurely to suboptimal deterministic policies or might get stuck in local minima because of the illusory reward patterns which is caused by randomness (Del Rio et al., 2024). There are various open-source benchmarks like OpenAI Baselines and Stable-Baselines3 which suggests that the hyperparameters like entropy coefficients and clipping range need to be carefully adjusted (Pereira, 2021; Amini & Creson, 2024). While the entropy bonus may promote more persistent exploration even in late-stage training processes, a small clipping range might avoid significant policy updates which could affect the learning (Cui et al., 2025; Ma et al., 2024; Zhu & Rosendo, 2021).

In addition to this, for handling sparse-reward, stochastic games better, the hybrid frameworks which combine PPO with evolutionary algorithms or meta-learning approaches like PPO with curiosity modules or curriculum learning have been proposed in different studies (Bai et al., 2023; Jin et al., 2025).

## 2.4. Reinforcement Learning in Gambling and Change-Based Games

RL has shown good performance in games like Chess and Go (Silver et al., 2018). However, its application to games chance, like European Roulette, remains very limited as these environments offer menial strategic control. However, the ability of RL to model long-term expected returns and adapt to high-variance feedback opens the room for exploration of use of the RL algorithms in such domains as well.

### 2.4.1. The UCLA Roulette Project

One of the most relevant studies in this area is the UCLA DataRes “Roulette Ai” project (2020). This project applied various RL algorithms to casino roulette (DataRes at UCLA, 2020). The study initially focused on Q-learning technique but also made use of the RL algorithms like Proximal Policy Optimization (PPO), Deep Q-Networks (DQN), Advantage Actor-Critic (A2C), Trust Region Policy Optimization (TRPO), and Actor-Critic with Experience Replay (ACER), across large number of the simulated spins. The comparative analysis showed that ACER consistently turned out to be the best agent (DataRes at UCLA, 2020).

One of the interesting findings of the project was that the agent did not try to “beat” the house edge, but instead learned to walk away early once it understood that any attempt at playing would result in the negative average reward. Thus, this behavior shows that the utility of RL is not in maximizing absolute gains, but in learning risk-aware, loss-averse strategies which is also a realistic and practical approach in casino-like settings. The study came to the conclusion that by making use of the learned rules rather than faulty heuristics, reinforcement learning (RL) may successfully capture long-term realistic and intelligent behaviors when applied to such stochastic systems (DataRes at UCLA, 2020).

### 2.4.2. WagerWin and RL in Stochastic Card Game

Despite the fact that it is not directly related to Roulette, the WagerWin framework developed for the Chinese card game *Dou Di Zhu* provides useful insights into how the PPO algorithm can handle complex, stochastic, and partially observable games (Wang et al., 2022). Through the combination of reward shaping, state abstraction, and self-play, the WagerWin framework enabled agents to develop effective wagering strategies under uncertainty (Wang et al., 2022).

The PPO can outperform baseline models in this setting because of its ability to handle probabilistic outcomes and hidden information. Even while the card games need more strategy than roulette, they also have certain challenges like non-deterministic feedback and limited control over outcomes. This supports the notion that RL may learn rational behavior even in randomness-dominated contexts if it is calibrated properly (Wang et al., 2022).

### 2.4.3. Implication for European Roulette

RL provides a firm basis for risk assessment, loss reduction, and human-level decision-making in non-deterministic worlds even though it cannot "solve" gambling games (Yu et al., 2021). Since all of the bets at European Roulette are negative expectation, RL redirects the emphasis from profit accumulation and towards loss reduction and survival maximization (Small & Tse, 2012). PPO is best suited to this application due to its stability versus exploration balance, noting its applicability in behavioral modeling and research studies (Del Rio et al., 2024).

## 2.5. Comparison of PPO with Other Reinforcement Learning Algorithms

In the field of reinforcement learning, the PPO algorithm is usually known for providing a good balance between simplicity and performance (Del Rio et al., 2024). Despite its efficacy varies based on the domain and specific challenges, it is one of the most-successful algorithms when it comes to context that demands steady on-policy learning (Babic, 2024).

In early stages, the PPO algorithms tend to learn quickly in dynamic settings. However, because of more reliable gradient updates, Advantage Actor Critic (A2C) could outperform it over long training times (Del Rio et al., 2024). PPO has shown various advantages over model-free algorithms like Deep Deterministic Policy Gradient (DDPG) in financial applications, which frequently feature delayed or noisy rewards (Lu, 2023; Mogammadshafie et al., 2024; Schulman et al., 2017).

Technically, PPO could give poor performance when in offline or highly parallelizable settings (Li et al., 2023). In some of the situations, algorithms like Soft Actor-Critic (SAC), Twin Delayed DDPG (TD3), or model-based techniques like MuZero can provide better outcomes by utilizing richer representations and more extensive data reuse (Tan et al., 2025).Thus as per these studies, hybrid or off-policy upgrades would be more suitable for managing the high variation and constrained observability seen in stochastic games like Roulette, even if PPO is a robust general-purpose baseline (Chinellato, n.d; Fujimoto et al., 2019).

## 2.6. Risk-Aware and Safety RL considerations

Since stochastic environments like European Roulette are high-variance, negative-expectation environments, it can be very useful to apply risk-aware strategies into the training process of Proximal Policy Optimization (PPO). The Standard PPO seeks to maximize expected return (Lu et al., 2022). However, because the environment naturally favours the house over time, this can result in risky or excessively aggressive behaviours in gambling environments.

In order to tackle this issue, Safe Reinforcement Learning (Safe RL) frameworks present methods that focus on reducing the likelihood of severe losses rather than simply maximizing the average performance (Gu et al., 2024). One of the techniques for this is the Conditional Value-at-Risk (CVaR). This technique allows PPO to prioritize policies which help to avoid high-loss outcomes by focusing on the lower tail of the reward distribution (Dong & Finlay, 2025; Stachowicz & Levine, 2024). This is especially relevant in Roulette, where a sequence of poor outcomes can quickly deplete virtual rollback of an agent.

In addition to this, the UCLA Rlette Project showed that PPO-based agents may learn conservative behaviours, including stopping early after a string of losses, with the right reward shape (DataRes at UCLA, 2020). PPO may be modified to perform simulation of the realistic betting behaviors like risk aversion and bankroll management by penalizing bankrupt states or extended negative drift (DataRes at UCLA, 2020).

## 2.7. State and Action Representations in Probability-Driven Game

Building efficient representations for states and actions is a major challenge in probability-driven settings like European Roulette (Blavatskyy, 2024). Roulette works mostly in static and memoryless settings, in comparison to the strategy games which have evolving and information rich states. The remaining betting capacity of the player, previous results, and current bankroll are usually very relevant state variables, which can lead to a low-complexity state space (Henrique Pereira, 2025; Blavatskyy, 2024; Mehdiyev, 2025).

Due to minimal variation and strong stochasticity of states, the traditional tabular methods such as basic Q-learning, is ineffective as these techniques rely heavily on bootstrapping (Ferguson, 2025). Roulette usually doesn’t have the distinct, repeated state changes which these techniques depend on (Blavatskyy, 2024). Deep policy gradient techniques like PPO are therefore much more appropriate since they may work across continuous or abstracted state spaces (Zhu et al., 2021). Instead of modelling particular bet types, researchers have suggested meta-actions like “exit table”, “double bet”, or “switch strategy” to improve agent reasoning in such contexts. This higher-level abstraction enables the agent to assess long-term risk and survival, rather than merely immediate payoffs (Yuan et al., 2024; Zhang et al., 2023; Restack, 2023).

Furthermore, strategies such as intrinsic motivation and reward shaping, as shown in models like RARSMS for Dou Di Zhu, show that making use of the unique reward signals can significantly increase learning in a low-reward context (Luo & Tan, 2023). Thus, building a well-structured state-action space and reward function is important for using the full potential of PPO in Roulette.

## 2.8. Summary of Literatures

| **Section** | **Title** | **Key Insight** | **Relevance to PPO in Roulette** |
| --- | --- | --- | --- |
| 2.1 | Introduction to Reinforcement Learning | RL learns through interaction and reward feedback. | Roulette’s stochastic nature aligns with RL learning paradigms. |
| 2.2. | RL in Stochastic Environments & Games | PPO has proven success in unpredictable games (e.g., Dota 2, Go); handles randomness better than Q-learning. | Demonstrates PPO’s suitability for fully stochastic, pattern less games like Roulette. |
| 2.3 | Proximal Policy Optimization (PPO) | PPO ensures stable learning through clipped updates, exploration bonuses, and variance control; performs well in sparse and noisy environments. | Well-suited for high-variance, low-signal settings like Roulette; adjustments improve convergence. |
| 2.4 | RL in Gambling & Chance-Based Games | PPO agents in Roulette learn to quit early, minimizing losses rather than seeking wins. | PPO captures loss-averse and risk-sensitive strategies in negative-expectation environments. |
| 2.5 | Comparison with Other RL Algorithms | PPO balances performance and simplicity but may be outperformed by hybrid or off-policy methods in some setups. | PPO is a strong baseline; hybrid approaches could further optimize performance in Roulette. |
| 2.6 | Risk-Aware and Safety RL | Techniques like CVaR and reward shaping steer PPO towards conservative play and bankroll preservation. | Enables PPO to avoid ruin and simulate realistic gambling behaviour. |
| 2.7 | State & Action Representations | Abstract meta-actions and intrinsic rewards improve learning in sparse-reward, static state spaces. | Enhances PPO’s effectiveness by allowing it to reason long-term and reduce action granularity. |

# 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 game play 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 colour (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 sets of features were recorded:

Table 1: 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 colour 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 behaviour 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 2: pie-chart of the color distribution of simulated roulette outcomes | Figure 3: Bar chart showing the parity distribution of the simulated roulette outcomes |
| --- | --- |
| Figure 4: 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, colour, and parity for each spin. For consistency and to preserve the temporal order of the events, the data was split into training and testing sets using a 70 to 30 ratio, without shuffling.

Table 2: 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 modelling 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 Strategy, Fibonacci Strategy, and D’Alembert Strategy, 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 Strategy

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 agent based on Martingale Strategy 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 Strategy

The Fibonacci Strategy 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 Strategy and thus, helps to manage risk while also aiming to recover the previous losses. The Fibonacci Strategy based agent performed similarly to the Martingale strategy in terms of average losses.

#### 3.3.2.3. D’Alembert Strategy

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 Strategy or Fibonacci Strategy.

## 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 behaviour 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 behaviour 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 behaviour 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 behaviour 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 3 - 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 behaviour. But the explained variance remained slightly negative This suggested that the model was struggling with accurately predicting returns.

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

Figure 5: Metrics from the last four training steps of the PPO agent

The test evaluation performed on the test dataset showed that the PPO agent was able to play for 12,732 number of spins and the total cumulative reward obtained by the agent was -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 Strategy, D'Alembert Strategy, Martingale Strategy, and a Random Strategy as 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 total rounds played by this strategy was 15,660 with the cumulative loss of -97.40 at the end.

### 4.2.2. Fibonacci Strategy

The slower linear progression helped to control the risk better than Martingale Strategy. However, this technique also ended in the loss with a cumulative reward of -31.10 which was significantly higher than the previous strategies. However, the number of rounds played by this strategy was comparatively lower in number (2,018) as compared to the Martingale Strategy (15,660).

### 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 with the cumulative loss of -21.80 only and the number of rounds played was around 2,047 which is slightly higher than the Fibonacci Strategy. The linear progression proved the most resilient, offering the smallest cumulative loss.

### 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.

## 4.3. DQN Strategy

The DQN agent was another reinforcement learning based model that was trained using a value-based reinforcement learning approach over 100\

000 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.

## 4.4. Comparative Summary

Table 4: Comparative summary of the test results

| **Strategy** | **Type** | **Rounds Played** | **Total Reward** | **Average Reward per Spin** | **Outcome** |
| --- | --- | --- | --- | --- | --- |
| PPO Agent | RL | 12,732 | –278.50 | –0.0218 | Terminated (Max Loss) |
| DQN Agent | RL | 1,198 | –126.90 | –0.1059 | Terminated (Max Loss) |
| Martingale Strategy | Classical | 15,660 | –97.40 | –0.0062 | Terminated (Max Loss) |
| Fibonacci Strategy | Classical | 2,018 | –31.10 | –0.0154 | Terminated (Max Loss) |
| D’Alembert Strategy | Classical | 2,047 | –21.80 | –0.0107 | Terminated (Max Loss) |
| Random Strategy | Classical | 12,163 | –648.00 | –0.0533 | Terminated (Max Loss) |

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

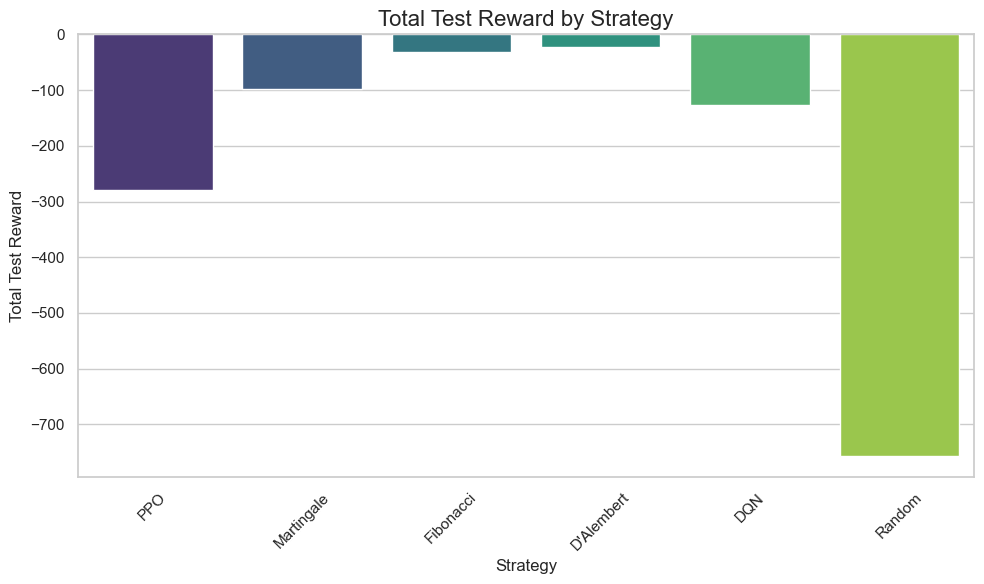


Figure 6: Plot of the total reward for each technique

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

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

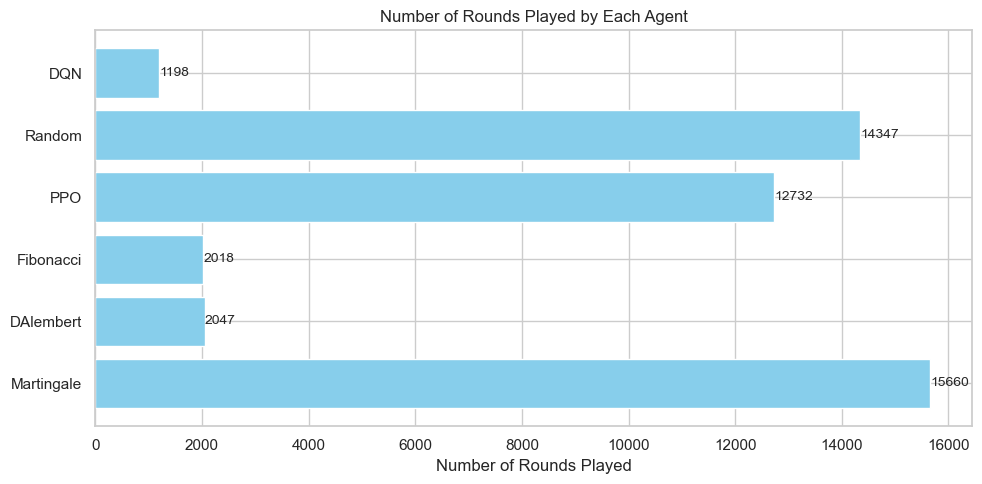


Figure 7: Bar Graph showing the number of spins each technique played before termination

The bar chart compares the number of rounds played by each agent before termination. It highlights the behavioural 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 behaviours 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 Strategy and D’Alembert Strategy 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 unfavourable 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:

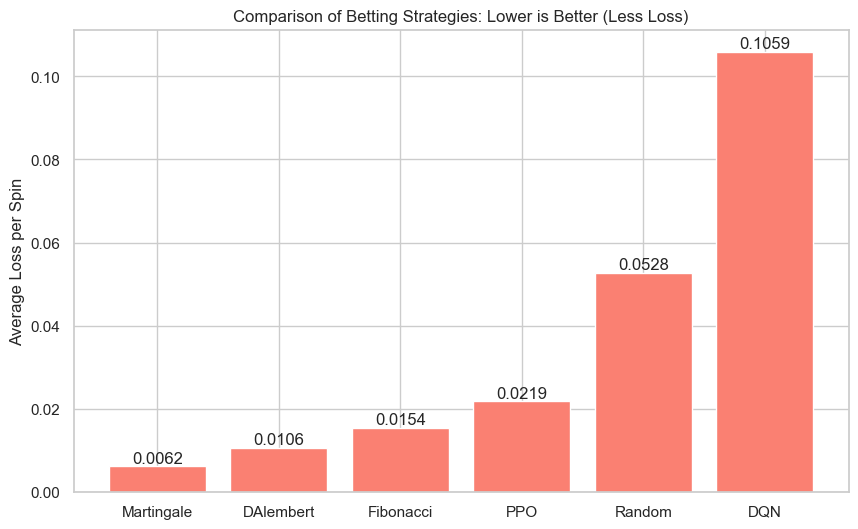


Figure 8: Average reward per spin for all techniques

The average reward for each round shows that Martingale Strategy performed best (-0.0062), with the smallest loss every spin, followed by D'Alembert Strategy and Fibonacci Strategy, 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. 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 Strategy, Fibonacci Strategy, D’Almenbert Strategy 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.

## 5.1. 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 behaviours 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.

## 5.2. 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 Strategy), 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 behaviours 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.

## 5.3. 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 Strategy based 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 understandings 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 behaviour 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.

## 5.4. 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.

## 5.5. 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.,

## 5.6. 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 make the adaptation of the ML techniques meaningless.

# CHAPTER 6. CONCLUSION

## 6.1. 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 behaviours, 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 Strategy and Fibonacci Strategy, 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.

## 6.2. 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 behaviours, 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.

## 6.3. 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 Strategy 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 modelling 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 7. 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 Strategy and Fibonacci Strategy 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 simpler betting strategies, even after experimenting with various parameters and running the models for thousands of steps.

One interesting outcome was that strategies like Martingale Strategy and Fibonacci Strategy 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, troubleshooting 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.

# CHAPTER 8. PROJECT MANAGEMENT

## 8.1. 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:

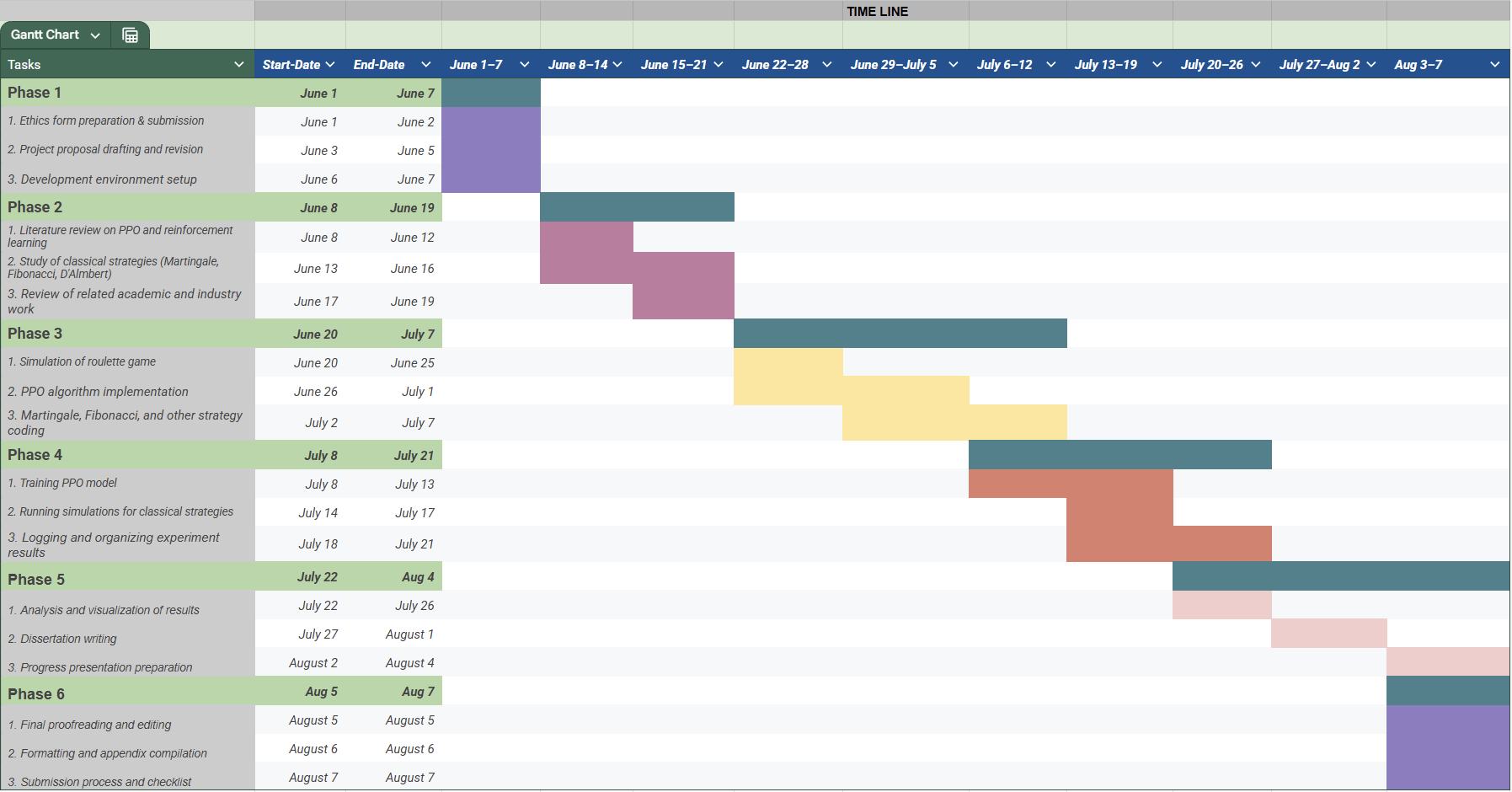


Figure 9: Gantt chart showing the project timelines created during planning phase

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.

## 8.2. 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 5: 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.

## 8.3. 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.

## 8.4. 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.

# REFERENCES

Abuhanna, D. (2015). Roulette: More than just a Chance.

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

Amini, M., & Creson, A. (2024). Cross-Domain Generalizability in Image Feature Extraction. *LU-CS-EX*.

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.

Babic, M. (2024). *Analysis and Evaluation of Reinforcement Learning Algorithms for a Continuous Control Problem* (Doctoral dissertation, Hochschule für Angewandte Wissenschaften Hamburg).

Bai, H., Cheng, R., & Jin, Y. (2023). Evolutionary reinforcement learning: A survey. *Intelligent Computing*, *2*, 0025.

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.

Chinellato, D. Dealing with Long-Term Constraints in a Hybrid Learning and Optimization Method.

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

Cui, G., Zhang, Y., Chen, J., Yuan, L., Wang, Z., Zuo, Y., ... & Ding, N. (2025). The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*.

DataRes at UCLA (2020, July 27). *Rlette - casino roulette through reinforcement learning*. Medium. https://ucladatares.medium.com/rlette-casino-roulette-through-reinforcement-learning-67e865843f0d

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*.

Dogru, O., Xie, J., Prakash, O., Chiplunkar, R., Soesanto, J., Chen, H., ... & Huang, B. (2024). Reinforcement learning in process industries: Review and perspective. *IEEE/CAA Journal of Automatica Sinica*, *11*(2), 283-300.

Dong, S. C., & Finlay, J. R. (2025). Adaptive Insurance Reserving with CVaR-Constrained Reinforcement Learning under Macroeconomic Regimes. *arXiv preprint arXiv:2504.09396*.

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).

Ferguson, S. (2025). *Exploring One-Step Fixed Horizon Q Learning in Tabular Stochastic Environments* (Doctoral dissertation).

Flet-Berliac, Y. (2021). *Sample-efficient deep reinforcement learning for control, exploration and safety* (Doctoral dissertation, Université de Lille).

Fujimoto, S., Meger, D., & Precup, D. (2019, May). Off-policy deep reinforcement learning without exploration. In *International conference on machine learning* (pp. 2052-2062). PMLR.

GGpoker. (2025). *Roulette - Understanding Roulette*. Welcome to LWC communities! https://help.ggpoker.com/article/Roleta-Entendendo-a-Roleta-1717475260871

Gu, S., Yang, L., Du, Y., Chen, G., Walter, F., Wang, J., & Knoll, A. (2024). A review of safe reinforcement learning: Methods, theories and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Hämäläinen, P., Babadi, A., Ma, X., & Lehtinen, J. (2020, September). PPO-CMA: Proximal policy optimization with covariance matrix adaptation. In *2020 IEEE 30th International Workshop on Machine Learni ng for Signal Processing (MLSP)* (pp. 1-6). IEEE.

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

Henrique Pereira, D. (2025). Victoria: Beating the House Using the Principles of Statistics and Randomness.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., & Abbeel, P. (2016). Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, *29*.

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.

Jin, W., Du, H., Zhao, B., Tian, X., Shi, B., & Yang, G. (2025). A comprehensive survey on multi-agent cooperative decision-making: Scenarios, approaches, challenges and perspectives. *arXiv preprint arXiv:2503.13415*.

Jin, Y. (2025). *Advanced Deep Reinforcement Learning for Real-Time Strategy Games* (Doctoral dissertation, Queen Mary University of London).

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

Liu, Z., Li, X., Kang, B., & Darrell, T. (2019). Regularization matters in policy optimization. *arXiv preprint arXiv:1910.09191*.

Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., & Foerster, J. (2022). Discovered policy optimisation. *Advances in Neural Information Processing Systems*, *35*, 16455-16468.

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.

Li, Z., Chen, T., Hong, Z. W., Ajay, A., & Agrawal, P. (2023, July). Parallel $ Q $-Learning: Scaling Off-policy Reinforcement Learning under Massively Parallel Simulation. In *International Conference on Machine Learning* (pp. 19440-19459). PMLR.

Li, T., Yang, G., & Chu, J. (2024). Uncertainty-based bootstrapped optimization for offline reinforcement learning. *International Journal of Machine Learning and Cybernetics*, 1-18.

Lu, C. I. (2023). Evaluation of deep reinforcement learning algorithms for portfolio optimisation. *arXiv preprint arXiv:2307.07694*.

Lu, H., Fang, L., Zhang, R., Li, X., Cai, J., Cheng, H., ... & Ma, P. (2025). Alignment and Safety in Large Language Models: Safety Mechanisms, Training Paradigms, and Emerging Challenges. *arXiv preprint arXiv:2507.19672*.

Luo, Q., & Tan, T. P. (2023). Rarsmsdou: Master the game of doudizhu with deep reinforcement learning algorithms. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *8*(1), 427-439.

Ma, H., Sima, K., Vo, T. V., Fu, D., & Leong, T. Y. (2024, June). Reward shaping for reinforcement learning with an assistant reward agent. In *Forty-first international conference on machine learning*.

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

Mehdiyev, S. (2025). Data-Driven Chess Typology: A Computational Approach: Machine Learning Classification of Player Types Based on Game Data.

Mohammadshafie, A., Mirzaeinia, A., Jumakhan, H., & Mirzaeinia, A. (2024, July). Deep reinforcement learning strategies in finance: Insights into asset holding, trading behavior, and purchase diversity. In *World Congress in Computer Science, Computer Engineering & Applied Computing* (pp. 449-463). Cham: Springer Nature Switzerland.

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.

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.

Pereira, B. A. B. (2021). *Deep Reinforcement Learning for Robotic Manipulation Tasks* (Master's thesis, Universidade de Aveiro (Portugal)).

Rahman, M. M. (2024). *Enhancing Policy Optimization for Improved Sample Efficiency and Generalization in Deep Reinforcement Learning* (Doctoral dissertation, Purdue University).

Rahman, B. (2025). PPO-BR: Dual-Signal Entropy-Reward Adaptation for Trust Region Policy Optimization. *arXiv preprint arXiv:2505.17714*.

Rahman, M. M., & Xue, Y. (2022, December). Bootstrap advantage estimation for policy optimization in reinforcement learning. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 234-239). IEEE.

Rao, B. D., & Arun, K. S. (2002). Model based processing of signals: A state space approach. *Proceedings of the IEEE*, *80*(2), 283-309.

Restack. (2025, March 6). *Roulette practice for AI simulations*. Restackio. https://d2wozrt205r2fu.cloudfront.net/p/ai-driven-simulation-answer-roulette-practice-cat-ai

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*.

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

Shen, Y., Gao, X., & Liang, Z. (2023, May). Covariance Matrix Adaptation for Multi-Agent Proximal Policy Optimization. In *2023 35th Chinese Control and Decision Conference (CCDC)* (pp. 4847-4852). IEEE.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140-1144.

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

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

Stachowicz, K., & Levine, S. (2024). Racer: Epistemic risk-sensitive rl enables fast driving with fewer crashes. *arXiv preprint arXiv:2405.04714*.

Tan, J. L., Taha, B. A., Aziz, N. A., Mokhtar, M. H. H., Mukhlisin, M., & Arsad, N. (2025). A Review of Reinforcement Learning Evolution: Taxonomy, Challenges and Emerging Solutions. *International Journal of Advanced Computer Science & Applications*, *16*(1).

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.

Wu, G., Fang, W., Wang, J., Ge, P., Cao, J., Ping, Y., & Gou, P. (2023). Dyna-PPO reinforcement learning with Gaussian process for the continuous action decision-making in autonomous driving. *Applied Intelligence*, *53*(13), 16893-16907.

Xu, Z., Wang, J., Xu, X., Yu, P., Huang, T., & Yi, J. (2025). A Survey of Reinforcement Learning-Driven Knowledge Distillation: Techniques, Challenges, and Applications.

Young, K., Ramesh, A., Kirsch, L., & Schmidhuber, J. (2022). The benefits of model-based generalization in reinforcement learning. *arXiv preprint arXiv:2211.02222*

Yu, C., Liu, J., Nemati, S., & Yin, G. (2021). Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, *55*(1), 1-36.

Yuan, C., Al Forhad, M. A., Bansal, R., Sidorova, A., & Albert, M. V. (2024). Multi-agent dual level reinforcement learning of strategy and tactics in competitive games. *Results in Control and Optimization*, *16*, 100471.

Wang, H., Wu, H., & Lai, G. (2022). Wagerwin: An efficient reinforcement learning framework for gambling games. *IEEE Transactions on Games*, *15*(3), 483-491.

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, S., Shen, L., & Han, L. (2021). Learning meta representations for agents in multi-agent reinforcement learning. *arXiv preprint arXiv:2108.12988*.

https://doi.org/10.48550/arXiv.2108.12988

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.

Zhu, J., Wu, F., & Zhao, J. (2021, December). An overview of the action space for deep reinforcement learning. In *Proceedings of the 2021 4th international conference on algorithms, computing and artificial intelligence* (pp. 1-10)

# APPENDIX I: CODE SNIPPETS

| ***#Code Snippet for Data Generation***  *#Importing necessary libraries*  *import random*  *import pandas as pd*  *import matplotlib.pyplot as plt*  *import seaborn as sns*  *import plotly.graph\_objects as go*  *import pandas as pd*  *class EuropeanRoulette:*  *"""*  *I'm simulating a European roulette wheel with the option to introduce bias.*  *Bias lets me favor certain numbers more often, which is useful for training RL models.*  *"""*  *def \_\_init\_\_(self, bias\_type=None, bias\_target=None, bias\_strength=0.1):*  *"""*  *Parameters:*  *- bias\_type: 'number', 'color', or 'range'*  *- bias\_target: depends on type (e.g., 17, 'red', or (1,12))*  *- bias\_strength: how strong the bias is (0.1 = 10% of the time, override random)*  *"""*  *self.numbers = list(range(37))*  *self.colors = self.\_assign\_colors()*  *# Save bias settings*  *self.bias\_type = bias\_type*  *self.bias\_target = bias\_target*  *self.bias\_strength = bias\_strength*  *def \_assign\_colors(self):*  *"""*  *I'm assigning colors based on real roulette coloring.*  *"""*  *red = {1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36}*  *return {*  *num: 'green' if num == 0 else 'red' if num in red else 'black'*  *for num in self.numbers*  *}*  *def spin(self):*  *"""*  *Simulate a spin, with an optional bias injected based on the user setting.*  *"""*  *apply\_bias = self.bias\_type is not None and random.random() < self.bias\_strength*  *if apply\_bias:*  *# I'm applying bias depending on the chosen type*  *if self.bias\_type == 'number':*  *result = self.bias\_target # Always return the biased number*  *elif self.bias\_type == 'color':*  *# I’ll filter all numbers of that color and pick one randomly*  *eligible = [n for n in self.numbers if self.colors[n] == self.bias\_target]*  *result = random.choice(eligible)*  *elif self.bias\_type == 'range':*  *# Range bias (e.g., (1, 12))*  *low, high = self.bias\_target*  *eligible = [n for n in self.numbers if low <= n <= high]*  *result = random.choice(eligible)*  *else:*  *result = random.choice(self.numbers) # fallback*  *else:*  *# No bias applied — just a fair spin*  *result = random.choice(self.numbers)*    *return result, self.colors[result]*  *# we can tweak these to generate different datasets*  *num\_spins = 1000000*  *# Example 1: Fair roulette (no bias)*  *wheel = EuropeanRoulette()*  *# I'll use this list to store each spin result*  *spin\_data = []*  *for i in range(num\_spins):*  *number, color = wheel.spin()*  *spin\_data.append({*  *"Spin": i + 1,*  *"Number": number,*  *"Color": color*  *})*  *# Convert to DataFrame and save to CSV*  *df = pd.DataFrame(spin\_data)*  *print("Frequency Analysis:\ n")*  *# Color distribution*  *color\_counts = df['Color'].value\_counts(normalize=True)*  *print("Color Frequency (%):")*  *print(color\_counts \* 100)*  *plt.figure(figsize=(6,6))*  *plt.pie(color\_counts, labels=color\_counts.index, colors=['red', 'black', 'green'], autopct='%1.1f%%', startangle=140)*  *plt.title("Color Distribution Pie Chart")*  *plt.axis('equal') # Keep the pie chart circular*  *plt.show()*  *# Prepare labels and values*  *labels = color\_counts.index.tolist()*  *values = color\_counts.values.tolist()*  *colors =['red', 'black', 'green']* |
| --- |
| *# Create pie chart*  *fig = go.Figure(*  *go.Pie(*  *labels=labels,*  *values=values,*  *hole=0.1,*  *pull=[0.05] \* len(values), # "Explode" effect*  *marker=dict(colors=colors, line=dict(color='white', width=2)),*  *textinfo='label+percent',*  *)*  *)*  *# Tilt effect (simulate 3D by rotating view)*  *fig.update\_layout(*  *title="Color Distribution Pie Chart",*  *showlegend=False,*  *margin=dict(t=80, l=50, r=50, b=50),*  *height=600,*  *width=600,*  *)*  *# This simulates a tilted effect with camera*  *fig.update\_traces(rotation=120, direction='clockwise')*  *# Show it*  *fig.show()*  *# Even/Odd distribution*  *parity = df['Number'].apply(lambda x: 'even' if x != 0 and x % 2 == 0 else 'odd' if x != 0 else 'zero')*  *parity\_counts = parity.value\_counts(normalize=True)*  *print("\nParity Frequency (%):")*  *print(parity\_counts \* 100)*  *# plotting parity*  *df['Parity'] = df['Number'].apply(*  *lambda x: 'even' if x != 0 and x % 2 == 0 else 'odd' if x != 0 else 'zero'*  *)*  *# Step 2: Plot barplot*  *plt.figure(figsize=(6, 4))*  *sns.countplot(x='Parity', data=df, order=['zero', 'even', 'odd'], palette='Set2')*  *# Step 3: Label the plot*  *plt.title("Parity Distribution (Even, Odd, Zero)")*  *plt.xlabel("Parity")*  *plt.ylabel("Count")*  *plt.tight\_layout()*  *plt.show()*  *# Low/High number range*  *low\_high = df['Number'].apply(lambda x: 'low' if 1 <= x <= 18 else 'high' if 19 <= x <= 36 else 'zero')*  *range\_counts = low\_high.value\_counts(normalize=True)*  *print("\nLow/High Range Frequency (%):")*  *print(range\_counts \* 100)*  *plt.figure(figsize=(16,6))*  *sns.countplot(x='Number', data=df, palette='viridis')*  *plt.title("Frequency of Each Roulette Number")*  *plt.xlabel("Number")*  *plt.ylabel("Count")*  *plt.show()*  *from scipy.stats import chisquare*  *# Expected probabilities*  *expected\_probs = [18/37, 18/37, 1/37] # black, red, green*  *observed\_counts = df['Color'].value\_counts().reindex(['black', 'red', 'green'], fill\_value=0).values*  *expected\_counts = [p \* len(df) for p in expected\_probs]*  *# Perform chi-square test*  *chi2\_stat, p\_value = chisquare(f\_obs=observed\_counts, f\_exp=expected\_counts)*  *print(f"\nChi-Square Statistic: {chi2\_stat:.4f}")*  *print(f"P-value: {p\_value:.4f}")*  *if p\_value < 0.05:*  *print("Roulette appears to be biased (statistically significant)")*  *else:*  *print("Roulette appears fair (no significant bias)")*  *# Check for potential unrealistic distributions*  *def check\_threshold(label, dist, threshold=0.50):*  *for k, v in dist.items():*  *if v > threshold:*  *print(f"Warning: {label} '{k}' exceeds {threshold\*100:.0f}% — check for unrealistic bias!")*  *return*  *print("Simulation is realsitic")*  *check\_threshold("Color", color\_counts)*  *check\_threshold("Parity", parity\_counts)*  *check\_threshold("Range", range\_counts)*  *bias\_name = wheel.bias\_type if wheel.bias\_type else "fair"*  *df.to\_csv(f"roulette\_spin\_data\_{bias\_name}.csv", index=False)*  *print(f"\n Data saved to roulette\_spin\_data\_{bias\_name}.csv")* |

| ***#Code For Algorithms***  *import random*  *import gymnasium as gym*  *from gymnasium import spaces*  *import numpy as np*  *import pandas as pd*  *from sklearn.model\_selection import train\_test\_split*  *from stable\_baselines3 import PPO*  *from stable\_baselines3 import DQN*  *from stable\_baselines3.common.env\_checker import check\_env*  *from stable\_baselines3.common.env\_util import make\_vec\_env*  *from stable\_baselines3.common.callbacks import EvalCallback*  *import matplotlib.pyplot as plt*  *import seaborn as sns*  *# reading the dataset*  *df = pd.read\_csv('roulette\_spin\_data\_fair.csv')*  *# checking into first five rows of the dataset to check if the data has been loaded successfully*  *df.head()*  *# Define a function to convert color to a number*  *def color\_to\_number(color):*  *if color == 'red':*  *return 1*  *elif color == 'black':*  *return 0*  *elif color == 'green':*  *return 2*  *else:*  *return -1 # in case of unexpected value*  *# Convert color to numeric format*  *df['Color'] = df['Color'].apply(lambda x: color\_to\_number(x))*  *# divide the datset into trin and test set in ratio 7:3. 70% for training and remaining for testing*  *train\_df, test\_df = train\_test\_split(df, test\_size=0.3, shuffle=False) #setting suffle false to kee the sequence intact*  *# Action encoding*  *action\_list = []*  *bet\_type\_to\_id = {}*  *bet\_id = 0*  *bet\_types = ["straight", "color\_red", "color\_black", "odd", "even", "low", "high",*  *"dozen1", "dozen2", "dozen3", "column1", "column2", "column3",*  *"trio\_0\_1\_2", "trio\_0\_2\_3", "basket", "color\_green"]*  *for bt in bet\_types:*  *if bt == "straight":*  *for num in range(37):*  *action\_list.append((bt, num))*  *elif bt in ["trio\_0\_1\_2", "trio\_0\_2\_3", "basket", "color\_green"]:*  *action\_list.append((bt, 0))*  *else:*  *action\_list.append((bt, -1))*  *if bt not in bet\_type\_to\_id:*  *bet\_type\_to\_id[bt] = bet\_id*  *bet\_id += 1*  *action\_to\_index = {a: i for i, a in enumerate(action\_list)}*  *index\_to\_action = {i: a for i, a in enumerate(action\_list)}*  *import random*  *import gymnasium as gym*  *from gymnasium import spaces*  *import numpy as np*  *class RouletteEnv(gym.Env):*  *metadata = {"render.modes": ["human"]}*  *def \_\_init\_\_(self, df=None, max\_bet=500, max\_win=10000, max\_loss=-10000, use\_historical=True):*  *super().\_\_init\_\_()*  *# Load dataset (spin history) if provided*  *self.df = df.reset\_index(drop=True) if df is not None else None*  *self.use\_historical = use\_historical*  *# Set how many steps (spins) this episode will run*  *self.max\_steps = len(df) if df is not None else 1000*  *self.current\_step = 0 # Start at first spin*  *# Define red and black numbers (used for color bets)*  *self.reds = {1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36}*  *self.blacks = {2, 4, 6, 8, 10, 11, 13, 15, 17, 20, 22, 24, 26, 28, 29, 31, 33, 35}*  *# Number of possible actions (bets the agent can place)*  *self.action\_space = spaces.Discrete(len(action\_list))*  *# Observation includes last 5 outcomes + current bet + progression count*  *low = np.array([0]\*5 + [0, 0], dtype=np.float32)*  *high = np.array([36]\*5 + [max\_bet, 10], dtype=np.float32)*  *self.observation\_space = spaces.Box(low=low, high=high, dtype=np.float32)*  *# Starting bet amount*  *self.base\_bet = 10*  *self.current\_bet = self.base\_bet*  *# Max allowed bet (used to prevent runaway losses)*  *self.max\_bet = max\_bet*  *# Maximum winning and loss limits*  *self.max\_win = max\_win*  *self.max\_loss = max\_loss*  *# Count how many losses in a row (used for doubling bets)*  *self.progression\_count = 0*  *# Keep last 5 spin results for observation*  *self.history = [0] \* 5*  *# Track total profit/loss during episode*  *self.total\_profit = 0*  *def reset(self, seed=None, options=None):*  *# Reset the environment for a new episode*  *super().reset(seed=seed)*  *self.current\_step = 0*  *self.current\_bet = self.base\_bet*  *self.progression\_count = 0*  *self.history = [0] \* 5 # Clear spin history*  *self.total\_profit = 0 # Reset total accumulated profit/loss*  *obs = self.\_get\_obs() # Return initial state*  *return obs, {}*  *def step(self, action\_idx):*  *# Get the bet type and value from the action index*  *bet\_type, bet\_number = index\_to\_action[action\_idx]*  *# Get the spin result: use historical data if available*  *if self.use\_historical and self.current\_step < len(self.df):*  *result = self.df.loc[self.current\_step, "Number"]*  *else:*  *result = random.randint(0, 36) # Simulate a new spin*  *# Calculate how much the agent wins (or loses)*  *payout = self.\_get\_payout(bet\_type, bet\_number, result)*  *# Invalid bet*  *if payout is None:*  *reward = -2 \* self.current\_bet # Penalty*  *# Win*  *elif payout > 0:*  *reward = payout \* self.current\_bet*  *# Loss*  *else:*  *reward = -self.current\_bet*  *# Track cumulative profit/loss*  *self.total\_profit += reward*  *# Normalize reward to keep PPO stable*  *reward = np.clip(reward / 100.0, -1.0, 1.0)*  *# Betting strategy (like Martingale): double bet on loss*  *if reward > 0:*  *self.current\_bet = self.base\_bet*  *self.progression\_count = 0*  *else:*  *self.progression\_count += 1*  *self.current\_bet = min(self.current\_bet \* 2, self.max\_bet)*  *if self.progression\_count > 5:*  *self.current\_bet = self.base\_bet*  *self.progression\_count = 0*  *# Update the spin history*  *self.history.pop(0)*  *self.history.append(result)*  *self.current\_step += 1*  *# Check if the episode is over:*  *# - Reached max number of steps*  *# - Agent has won too much*  *# - Agent has lost too much*  *terminated = (*  *self.current\_step >= self.max\_steps or*  *self.total\_profit >= self.max\_win or*  *self.total\_profit <= self.max\_loss*  *)*  *obs = self.\_get\_obs()*  *return obs, reward, terminated, False, {}*  *def \_get\_obs(self):*  *# Create the observation (state): last 5 numbers + bet info*  *return np.array(self.history + [self.current\_bet, self.progression\_count], dtype=np.float32)*  *def \_get\_payout(self, bet\_type, bet\_number, result):*  *# Check payout rules for different types of bets*  *if bet\_type == "straight":*  *return 35 if bet\_number == result else 0*  *elif bet\_type == "color\_red":*  *return 1 if result in self.reds else 0*  *elif bet\_type == "color\_black":*  *return 1 if result in self.blacks else 0*  *elif bet\_type == "color\_green":*  *return 35 if result == 0 else 0*  *elif bet\_type == "odd":*  *return 1 if result != 0 and result % 2 == 1 else 0*  *elif bet\_type == "even":*  *return 1 if result != 0 and result % 2 == 0 else 0*  *elif bet\_type == "low":*  *return 1 if 1 <= result <= 18 else 0*  *elif bet\_type == "high":*  *return 1 if 19 <= result <= 36 else 0*  *elif bet\_type == "dozen1":*  *return 2 if 1 <= result <= 12 else 0*  *elif bet\_type == "dozen2":*  *return 2 if 13 <= result <= 24 else 0*  *elif bet\_type == "dozen3":*  *return 2 if 25 <= result <= 36 else 0*  *elif bet\_type == "column1":*  *return 2 if result in {1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34} else 0*  *elif bet\_type == "column2":*  *return 2 if result in {2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35} else 0*  *elif bet\_type == "column3":*  *return 2 if result in {3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36} else 0*  *elif bet\_type == "trio\_0\_1\_2":*  *return 11 if result in {0, 1, 2} else 0*  *elif bet\_type == "trio\_0\_2\_3":*  *return 11 if result in {0, 2, 3} else 0*  *elif bet\_type == "basket":*  *return 8 if result in {0, 1, 2, 3} else 0*  *else:*  *return None # Invalid bet*  *#!pip install "shimmy>=2.0"*  *# creating environment*  *train\_env = RouletteEnv(train\_df, use\_historical=True)*  *test\_env = RouletteEnv(test\_df, use\_historical=True)*  *# ---- Train PPO ----*  *model = PPO("MlpPolicy", train\_env, verbose=1)*  *model.learn(total\_timesteps=100\_000)*  *# ---- Test/Evaluate the trained PPO agent ----*  *obs, \_ = test\_env.reset() # Reset the environment to start evaluation*  *total\_reward\_ppo = 0 # Track total cumulative reward*  *rounds\_played\_by\_ppo = 0 # Track how many games/steps agent played*  *for \_ in range(len(test\_df)):*  *action, \_ = model.predict(obs, deterministic=True)*  *action = int(action) # Convert action to integer*  *obs, reward, done, truncated, info = test\_env.step(action)*  *total\_reward\_ppo += reward*  *rounds\_played\_by\_ppo += 1*  *# Explicitly check if win/loss limit triggered episode termination*  *if test\_env.total\_profit >= test\_env.max\_win:*  *print("PPO Agent Terminated: Maximum winning limit reached.")*  *break*  *elif test\_env.total\_profit <= test\_env.max\_loss:*  *print("PPO Agent Terminated: Maximum loss limit reached.")*  *break*  *if done:*  *break # Exit if environment signals end of episode*  *# Report results*  *print(f"Total test reward (PPO): {total\_reward\_ppo}")*  *print(f"Total rounds played by PPO Agent: {rounds\_played\_by\_ppo}")*  *class MartingaleAgent:*  *def \_\_init\_\_(self, env, base\_bet=10, max\_bet=500):*  *self.env = env*  *self.base\_bet = base\_bet*  *self.current\_bet = base\_bet*  *self.max\_bet = max\_bet*  *self.progression\_count = 0*  *self.action = None*  *for i, (bet\_type, bet\_num) in enumerate(env.action\_list if hasattr(env, 'action\_list') else []):*  *if bet\_type == "color\_red":*  *self.action = i*  *break*  *if self.action is None:*  *self.action = list(index\_to\_action.keys())[list(index\_to\_action.values()).index(("color\_red", -1))]*  *def run\_episode(self):*  *obs, \_ = self.env.reset()*  *total\_reward = 0*  *done = False*  *rounds\_played = 0 # Track how many games/steps were played*  *while not done:*  *obs, reward, terminated, truncated, info = self.env.step(self.action)*  *total\_reward += reward*  *rounds\_played += 1*  *if self.env.total\_profit >= self.env.max\_win:*  *print("Terminated: Maximum winning limit reached.")*  *break*  *elif self.env.total\_profit <= self.env.max\_loss:*  *print("Terminated: Maximum loss limit reached.")*  *break*  *if terminated or truncated:*  *done = True*  *print(f"Total rounds played by MartingaleAgent: {rounds\_played}")*  *return total\_reward, rounds\_played*  *# ---- Create environments ----*  *train\_env = RouletteEnv(train\_df, use\_historical=True)*  *test\_env = RouletteEnv(test\_df, use\_historical=True)*  *# Inject action\_list and index\_to\_action into env for agent (since your env does not expose these by default)*  *train\_env.action\_list = action\_list*  *train\_env.index\_to\_action = index\_to\_action*  *test\_env.action\_list = action\_list*  *test\_env.index\_to\_action = index\_to\_action*  *# Running Martingale agent*  *martingale\_agent = MartingaleAgent(test\_env)*  *total\_reward\_martingale, rounds\_played\_by\_martingale = martingale\_agent.run\_episode()*  *print(f"Total test reward for Martingale: {total\_reward\_martingale}")*  *class FibonacciAgent:*  *def \_\_init\_\_(self, env, base\_bet=10, max\_bet=500):*  *self.env = env*  *self.base\_bet = base\_bet*  *self.max\_bet = max\_bet*  *self.fib\_sequence = [1, 1]*  *self.current\_index = 0*  *self.action = None*  *for i, (bet\_type, bet\_num) in enumerate(env.action\_list if hasattr(env, 'action\_list') else []):*  *if bet\_type == "color\_red":*  *self.action = i*  *break*  *if self.action is None:*  *self.action = list(index\_to\_action.keys())[list(index\_to\_action.values()).index(("color\_red", -1))]*  *def \_next\_bet\_amount(self):*  *while self.current\_index >= len(self.fib\_sequence):*  *self.fib\_sequence.append(self.fib\_sequence[-1] + self.fib\_sequence[-2])*  *return min(self.fib\_sequence[self.current\_index] \* self.base\_bet, self.max\_bet)*  *def run\_episode(self):*  *obs, \_ = self.env.reset()*  *total\_reward = 0*  *done = False*  *self.current\_index = 0*  *rounds\_played = 0 # Track how many games/steps were played*  *while not done:*  *bet\_amount = self.\_next\_bet\_amount()*  *self.env.current\_bet = bet\_amount*  *obs, reward, terminated, truncated, info = self.env.step(self.action)*  *total\_reward += reward*  *rounds\_played += 1*  *if reward > 0:*  *self.current\_index = max(0, self.current\_index - 2)*  *else:*  *self.current\_index += 1*  *if self.env.total\_profit >= self.env.max\_win:*  *print("Terminated: Maximum winning limit reached.")*  *break*  *elif self.env.total\_profit <= self.env.max\_loss:*  *print("Terminated: Maximum loss limit reached.")*  *break*  *if terminated or truncated:*  *done = True*  *print(f"Total rounds played by FibonacciAgent: {rounds\_played}")*  *return total\_reward, rounds\_played*  *fibonacci\_agent = FibonacciAgent(test\_env)*  *total\_reward\_fib, rounds\_played\_by\_fibo = fibonacci\_agent.run\_episode()*  *print(f"Total test reward for Fibonacci: {total\_reward\_fib}")*  *class DAlembertAgent:*  *def \_\_init\_\_(self, env, base\_bet=10, max\_bet=500):*  *self.env = env*  *self.base\_bet = base\_bet*  *self.max\_bet = max\_bet*  *self.current\_bet = base\_bet # Start betting with the base amount*    *# Choose a fixed bet action — here, always bet on "color\_red"*  *self.action = None*  *# Find the action index corresponding to "color\_red" in the environment's action list*  *for i, (bet\_type, bet\_num) in enumerate(env.action\_list if hasattr(env, 'action\_list') else []):*  *if bet\_type == "color\_red":*  *self.action = i*  *break*  *# If not found, fallback to default way to get the "color\_red" action index*  *if self.action is None:*  *self.action = list(index\_to\_action.keys())[list(index\_to\_action.values()).index(("color\_red", -1))]*  *def run\_episode(self):*  *# Reset the environment to start a new episode (new game)*  *obs, \_ = self.env.reset()*  *total\_reward = 0 # Keep track of total wins/losses during this episode*  *done = False*  *self.current\_bet = self.base\_bet # Start with base bet amount*  *rounds\_played = 0 # Track how many games/steps were played*  *while not done:*  *self.env.current\_bet = self.current\_bet # Set the current bet in the environment*  *# Make the bet (action) and get the result from environment*  *obs, reward, terminated, truncated, info = self.env.step(self.action)*  *total\_reward += reward # Add win/loss reward to total*  *rounds\_played += 1*  *if reward > 0:*  *# If we win, decrease bet size by one base unit (but never go below base bet)*  *self.current\_bet = max(self.base\_bet, self.current\_bet - self.base\_bet)*  *else:*  *# If we lose, increase bet size by one base unit (but don't exceed max allowed)*  *self.current\_bet = min(self.current\_bet + self.base\_bet, self.max\_bet)*  *# Check if maximum win/loss limit has been reached*  *if self.env.total\_profit >= self.env.max\_win:*  *print("winnnn",self.env.total\_profit)*  *print("Terminated: Maximum winning limit reached.")*  *break*  *elif self.env.total\_profit <= self.env.max\_loss:*  *print("Terminated: Maximum loss limit reached.")*  *break*  *# Check if the episode ended (no more spins)*  *if terminated or truncated:*  *done = True*  *print(f"Total rounds played by DAlembertAgent: {rounds\_played}")*  *return total\_reward, rounds\_played*  *# Running D’Alembert agent*  *dalembert\_agent = DAlembertAgent(test\_env)*  *total\_reward\_dalembert, rounds\_played\_by\_dalembert = dalembert\_agent.run\_episode()*  *print(f"Total test reward for D’Alembert: {total\_reward\_dalembert}")*  *import random*  *import numpy as np*  *class RandomAgent:*  *def \_\_init\_\_(self, env, seed, base\_bet=10):*  *self.env = env*  *self.base\_bet = base\_bet*  *self.total\_reward = 0*  *self.total\_bet = 0*  *self.seed = seed*  *# Enforce seed for reproducibility*  *random.seed(self.seed)*  *np.random.seed(self.seed)*  *self.env.reset(seed=self.seed)*  *if hasattr(self.env.action\_space, 'seed'):*  *self.env.action\_space.seed(self.seed)*  *def run\_episode(self):*  *obs, \_ = self.env.reset(seed=self.seed)*  *done = False*  *self.total\_reward = 0*  *self.total\_bet = 0*  *rounds\_played = 0 # Track number of rounds played*  *while not done:*  *# Force environment to bet flat before step*  *self.env.current\_bet = self.base\_bet*  *current\_bet = self.env.current\_bet*  *action = self.env.action\_space.sample()*  *obs, reward, terminated, truncated, info = self.env.step(action)*  *# Immediately reset bet to base after the environment tries to double it*  *self.env.current\_bet = self.base\_bet*  *self.env.progression\_count = 0*  *# Keep environment's clipped reward to avoid reward explosion*  *self.total\_reward += reward*  *self.total\_bet += current\_bet*  *rounds\_played += 1 # Increment round counter*  *# Check for max win/loss condition manually*  *if self.env.total\_profit >= self.env.max\_win:*  *print("RandomAgent Terminated: Maximum winning limit reached.")*  *break*  *elif self.env.total\_profit <= self.env.max\_loss:*  *print("RandomAgent Terminated: Maximum loss limit reached.")*  *break*  *if terminated or truncated:*  *done = True*  *print(f"Total rounds played by RandomAgent: {rounds\_played}")*  *return self.total\_reward, rounds\_played*  *random\_env = RouletteEnv(df=test\_df, use\_historical=True)*  *agent = RandomAgent(random\_env, seed=10)*  *total\_reward\_random, rounds\_played\_by\_random = agent.run\_episode()*  *print(f"Total test reward (Random Agent): {total\_reward\_random:.5f}")*  *print(f"Total rounds played: {rounds\_played\_by\_random}")*  *# Training the DQN Agent*  *dqn\_model = DQN(*  *policy="MlpPolicy",*  *env=train\_env,*  *verbose=1,*  *learning\_rate=1e-4,*  *buffer\_size=50000,*  *learning\_starts=1000,*  *batch\_size=32,*  *tau=1.0,*  *gamma=0.99,*  *train\_freq=1,*  *target\_update\_interval=500,*  *exploration\_fraction=0.1,*  *exploration\_final\_eps=0.05,*  *tensorboard\_log="./dqn\_roulette\_tensorboard/"*  *)*  *# Train for a number of timesteps*  *dqn\_model.learn(total\_timesteps=100\_000)*  *# Evaluate on test set*  *obs, \_ = test\_env.reset()*  *total\_reward\_dqn = 0*  *rounds\_played\_by\_dqn = 0 # Track how many games/steps were played*  *for \_ in range(len(test\_df)):*  *action, \_ = dqn\_model.predict(obs, deterministic=True)*  *obs, reward, done, truncated, info = test\_env.step(int(action))*  *total\_reward\_dqn += reward*  *rounds\_played\_by\_dqn += 1*  *# Check environment termination conditions*  *if test\_env.total\_profit >= test\_env.max\_win:*  *print("Terminated: Maximum winning limit reached.")*  *break*  *elif test\_env.total\_profit <= test\_env.max\_loss:*  *print("Terminated: Maximum loss limit reached.")*  *break*  *if done or truncated:*  *break*  *# Return and print standard metrics*  *print(f"Total rounds played by DQN: {rounds\_played\_by\_dqn}")*  *print(f"Total test reward (DQN): {total\_reward\_dqn}")*  *results = {*  *"PPO": total\_reward\_ppo,*  *"Martingale": total\_reward\_martingale,*  *"Fibonacci": total\_reward\_fib,*  *"D'Alembert": total\_reward\_dalembert,*  *"DQN": total\_reward\_dqn,*  *"Random": total\_reward\_random*  *}*  *sns.set(style="whitegrid")*  *# Create lists*  *strategies = list(results.keys())*  *rewards = list(results.values())*  *# Plot*  *plt.figure(figsize=(10, 6))*  *sns.barplot(x=strategies, y=rewards, palette="viridis")*  *# Add titles and labels*  *plt.title("Total Test Reward by Strategy", fontsize=16)*  *plt.xlabel("Strategy", fontsize=12)*  *plt.ylabel("Total Test Reward", fontsize=12)*  *plt.xticks(rotation=45)*  *plt.tight\_layout()*  *plt.show()*  *results = {*  *"PPO": total\_reward\_ppo,*  *"DQN": total\_reward\_dqn,*  *"Martingale": total\_reward\_martingale,*  *"Fibonacci": total\_reward\_fib,*  *"DAlembert": total\_reward\_dalembert,*  *"Random": total\_reward\_random*  *}*  *rounds\_played\_dict = {*  *"PPO": rounds\_played\_by\_ppo,*  *"DQN": rounds\_played\_by\_dqn,*  *"Martingale": rounds\_played\_by\_martingale,*  *"Fibonacci": rounds\_played\_by\_fibo,*  *"DAlembert": rounds\_played\_by\_dalembert,*  *"Random": rounds\_played\_by\_random*  *}*  *# Compute average reward (loss) per spin for each strategy*  *avg\_rewards = {*  *k: (results[k] / rounds\_played\_dict[k]) if rounds\_played\_dict[k] > 0 else 0*  *for k in results.keys()*  *}*  *# Convert to positive loss values for plotting*  *loss\_magnitudes = {k: -v for k, v in avg\_rewards.items()} # negative reward means loss*  *# Sort by magnitude of loss (lowest loss first)*  *sorted\_items = sorted(loss\_magnitudes.items(), key=lambda x: x[1])*  *strategies = [k for k, v in sorted\_items]*  *loss\_values = [v for k, v in sorted\_items]*  *import matplotlib.pyplot as plt*  *plt.figure(figsize=(10,6))*  *bars = plt.bar(strategies, loss\_values, color='salmon')*  *plt.ylabel('Average Loss per Spin')*  *plt.title('Comparison of Betting Strategies: Lower is Better (Less Loss)')*  *# Add numeric labels on top of each bar*  *for bar, loss in zip(bars, loss\_values):*  *plt.text(bar.get\_x() + bar.get\_width()/2, bar.get\_height(), f"{loss:.4f}",*  *ha='center', va='bottom')*  *plt.show()*  *# Extract rounds played values in the same order as strategies for consistency*  *rounds\_played\_values = [rounds\_played\_dict[s] for s in strategies]*  *plt.figure(figsize=(10, 5))*  *bars = plt.barh(strategies, rounds\_played\_values, color='skyblue')*  *plt.xlabel('Number of Rounds Played')*  *plt.title('Number of Rounds Played by Each Agent')*  *# Add numeric labels next to bars*  *for bar, rounds\_played in zip(bars, rounds\_played\_values):*  *plt.text(bar.get\_width() + 1, bar.get\_y() + bar.get\_height()/2, f"{rounds\_played}",*  *va='center', fontsize=10)*  *plt.tight\_layout()*  *plt.show()* |
| --- |

# APPENDIX-II: PROPOSAL

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

# APPENDIX-III: MEETING LOGS

| **Meeting** | **Discussion / Feedback** | **Action Taken** | **File / Code Reference** | **Outcome** |
| --- | --- | --- | --- | --- |
| 1 | Simulate ≥10k spins, check fairness vs real European roulette. | Implemented Chi-Square GOF (18/37, 18/37, 1/37), 50% threshold cap for Color/Parity/Range. | DataGenerator.ipynb — Chi-Square test, check\_threshold() | p > 0.05 → “Roulette appears fair” |
| 1 | Pie chart colors must match roulette (red/black/green). | Fixed explicit color mapping in Matplotlib & Plotly. | DataGenerator.ipynb — colors=['red','black','green'] | Visuals match exactly |
| 1 | Add casino constraints: max bet, progression limit. | Added bet doubling cap: min(current\_bet \* 2, max\_bet). Progression reset after N losses. | RouletteEnv in RL notebook | Progression realistic, avoids runaway bets |
| 2 | Add max win & max loss limits. | Episode ends if total\_profit >= max\_win or total\_profit <= max\_loss. | RouletteEnv in RL notebook | Simulates real casino termination rules |
| 1 & 2 | Compare Martingale Strategy, Fibonacci Strategy, D’Alembert Strategy with PPO, DQN, Random. | Implemented all strategies; evaluated on ≥10k spins; plotted results. | RL training notebook | All strategies benchmarked; Martingale/Fibonacci lowest loss this run |
| 2 | “Average loss per betting amount” and “no significant difference in avg loss per spin” between Random & Fibonacci. | Calculated per-spin avg reward plotted bar chart; noted all negative due to house edge. | RL training notebook — final evaluation | Confirms small differences; all negative EV |
| 1 & 2 | Threshold check change from 55% to 50%. | Updated threshold in fairness check. | check\_threshold(threshold=0.50) | More sensitive bias detection |
| 2 | Ensure Chi-square is also used for biased wheel check. | Validated Chi-square for all datasets (fair & biased). | DataGenerator.ipynb — Chi-Square test section | Detects deviations from expected distribution |
| Future | Optional: avg loss per unit bet & std dev; biased wheel RL test; train PPO/DQN longer. | Not yet implemented. | N/A | Planned enhancements |

# APPENDIX-IV: DISSERTATION PROPOSALSLIDES

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

**APPENDIX-V: GITHUB REPOSITORY AND ONE DRIVE LINK**

**GitHub Link**

[**https://github.com/SrinathMLOps/Dissertation**](https://github.com/SrinathMLOps/Dissertation)

**One Drive Link**

[**https://livecoventryac-my.sharepoint.com/:f:/r/personal/kaithojus\_uni\_coventry\_ac\_uk/Documents/Dissertation-main?csf=1&web=1&e=ZtYZdh**](https://livecoventryac-my.sharepoint.com/:f:/r/personal/kaithojus_uni_coventry_ac_uk/Documents/Dissertation-main?csf=1&web=1&e=ZtYZdh)