ShortDeckAI
Interim Report

DT228/4
BSc in Computer Science

Harry O'Donnell

C20379081

Mr. Brendan Tierney

School of Computer Science

Technological University, Dublin

12/04/24

Abstract

"Games are playgrounds in which to understand how minds—human and artificial—learn on their own to achieve goals"(1)

Based on David Siler's notion, I'm focused on the strategic domain of Short Deck poker, an imperfect information game. The project's core goal is the development of a web application which enables user to test their skills against an AI agent that's been trained through reinforcement learning.

Not only is it an exploration into the world of decision making under incomplete knowledge of its environment but ShortDeckAI is about providing a platform for users to freely play against a strategically sound player without having to sacrifice their money.

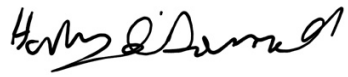
Poker enthusiasts will no longer have to improve their skills through online cash games, which aren't designed to educate the players but instead focuses on the players spending.

This platform will aim to educate players on how to improve their overall game and decision making in various situations without the stress of sacrificing their hard earned money.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:



Harry O'Donnell

12/04/24

Acknowledgements

I would like to thank my supervisor Mr. Brendan Tierney for his extensive guidance and advice over the past few months. He provided crucial advice throughout our weekly meetings which led to the improvement of many features in my final project.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The use of Artificial Intelligence (AI) in games has been a long journey, not just within the scope of perfect information games such as Chess or Go but also in the complex domain of imperfect information games like Poker. My final year project will explore the complexities of this imperfect information game with a particular emphasis on the Short Deck variant of Poker. Imperfect information games are due to the incomplete knowledge of the complete environment and are an ideal testing ground for reinforcement learning approaches.

The projects foundation is developing an AI agent to play an imperfect information game using reinforcement learning. The agent will be trained through iterative self-play with the goal of progressively converging towards the Nash equilibrium. The Nash equilibrium represents the ideal strategic balance in which "a player does not gain anything from deviating from their initially chosen strategy". (2)

This project is unique since, it's not only about developing a sophisticated AI agent but also a user-focused web application. The application acts as a link between the user who wants to improve their poker skills and the AI's complex algorithm. It's intended to provide an interactive environment where users may interact with AI to learn from its sophisticated tactics and enhance their gameplay in situations that occur in real time.

The web application is crucial as it turns the advances in AI research into a practical and useful tool for poker players to improve their skills against a trained AI agent. The application allows players to play freely and gain real and engaging benefits in their poker playing skills through repetition against the agent and the feedback received from the application.

In the following sections, I will explore the project's background and detail the research and methodology that paved the way for this project. It also outlines the goals and objectives, scope, and anticipated challenges and solutions for a successful project.

1.1. Project Background

"As a game of incomplete information and uncertainty, poker is a prime application of the game theory concepts and decision making skills essential to trading. While traders make risk decisions based on the limited information they get from the markets, poker players make decisions based on hidden information as well, taking into account factors such as expected value and probability distributions".(3)

The world of imperfect information games, where uncertainty is at the forefront and decisions must be made based on incomplete and unavailable data, finds parallels in the world of trading, finance and many other real world scenarios and applications. In these scenarios, just like at the poker table, decisions depend on hidden community cards and unknown moves by your opponent. This comparison highlights the value of game theory and decision making skills which are vital in many real world applications.

My initial inspiration for a project of this type; using reinforcement learning, came when watching a video on DeepMind's development of AlphaGo, who is an AI that beat a human world champion in Go. From there, I began my exploration into reinforcement learning and ended up watching David Silver's ten part YouTube series from University College London

on reinforcement Learning (4). I spent the next few months exploring potential applications of reinforcement learning that I wished to explore for my final year project. This laid the groundwork for my current exploration into the mechanics of poker.

There has been plenty of research done in recent years into machine learning and artificial intelligence techniques such as reinforcement learning and neural networks in imperfect information games, specifically poker. Metas AI lab built a computer poker player by name of pluribus in 2019 which beat 5 human poker professionals in the game of no-limit Texas Hold'em. (5) The primary challenge with training poker AI's is the size of the game space, in the full game of Heads-Up Limit Hold'em (HULHE) there are 3.19 X 10^14 information states, which is impossible to compute without significant time and resources. In order to compute the optimal strategy for a large scale game, the game size must be reduced. Reducing the size of the game is primarily done with the use of abstraction techniques, which makes the game more manageable and the optimal strategy computable for that simplified game. Once the game has been abstracted, the abstracted game is mapped back to the original game and the strategy is an approximate Nash equilibrium of the original game. The best method to reduce the size of the game is by action abstraction and card abstraction. There has been a lot of research into abstraction techniques, such as bucketing the similar hands together, which works by putting a hands into buckets according to hand strength.

This project will focus on the Short Deck Poker variant. Short Deck Poker is a simplified version of the traditional 52 card deck by removing 2 to 5, changing the strategic depth of the game. This variant is gaining popularity in Asia and has strong strategic properties, providing new perspectives and unique challenges for AI development.

Within this context, the AI agent will play against itself and attempt to improve their overall strategy. I will be using a reinforcement learning algorithm called counterfactual regret minimisation (CFR) to train the agent, it analyses previous actions and estimates their positive or negative regret for making that decision, and adjust the strategy to minimise those regrets and making better decisions in the future.

This platform allows the user to interact with and learn from the AI agent and refine their decision-making skills in Short Deck poker, thereby bridging the gap between poker strategies and practical user application.

1.2. Project Description

This project explores the world of artificial intelligence (AI) in the context of poker, concentrating on the growingly popular Short Deck variation. The goal of the project is twofold: first, to create an AI agent using reinforcement learning techniques that can master this difficult variation of poker, and second, to develop a web application that acts as a bridge between the AI and poker players.

The user will be required to login / register their profile to gain access to the web application. Once logged in, the user will arrive at the homepage which will display their previous results and analyse the games played, which can be used by the user to improve their overall performance. The user will also have the ability to customise their profile by adding a

nickname and choose an avatar to represent themselves, both of which will be displayed whilst playing poker against the agent. Additionally, the user can access a detailed view of all the hands played in the previous game, along with the data that led to the AI making its various decisions.

Finally, the user can play Short Deck Poker against the AI agent, the agent is the heart of the project. The user will have the freedom to directly engage with the AI agent, with the standard poker actions such as call, raise, check and fold. At the beginning of each round each player is dealt two cards and a series of community cards are dealt in the centre of the table, the players will take turns taking a variety of actions with the aim of winning the pot of chips. At the end of each round, if neither player has folded, the system will determine the winner based off the player cards and the community cards combined, the winner will win the pot.

Whilst playing poker, the user will also have the ability to view the previous hand to review their actions and AI's actions to potentially learn from their strategic mistakes and the process that led to the AI's decisions.

1.3. Project Aims and Objectives

The aim of this project is to create a robust AI agent that can play the Short Deck variant of poker to a very high standard through reinforcement learning. The agents ability will serve as a sophisticated opponent for players to sharpen and improve their overall skills by playing against the agent in the web application.

- Develop a much simpler version of poker such as three card poker. I aim to do this to get an idea of how the game engine and the reinforcement learning algorithm will work.
- Develop the full Short Deck game engine.
- Implement the reinforcement leaning algorithm, counterfactual regret minimisation (CFR).
- Develop the abstraction to simplify the game for it to be computable by my laptop
- Train the AI agent through self-play.
- Develop the poker web application for the users to play against the agent.
- Provide the capability for the users to see their performance history using visualisations and performance metrics.
- Develop a view for user to view the previous hands played against the AI, it will supply a breakdown of the decision making process, which is valuable to the user.
- Do human testing on the agent through the web application and analyse the performance of the agent against the humans and test the functionality of the web application.
- Develop a very simple AI agent that can play agents the trained agent to analyse the agents performance.

1.4. Project Scope

The scope of the project is defined by the specific goals and objectives its set out to achieve within the constraints of time and resources. This project isn't meant to be a revolutionary poker web application or an attempt to solve Short Deck poker. It's meant to combine both of these components to provide the users with great insight into the AI's thinking, through the use of the web application.

This project will focus on the following key areas:

Login & Register

Upon visiting the web application, the user must be prompted to either login to their previously registered account or register a new profile. This profile will store key information in the database which will allow the user to track their results and history and customise their player profile.

User Performance Analyses

The homepage of the web application will include metrics to track and analyse the users gameplay history and results, which can be used by the user to progress and evolve their playstyle and overall ability.

Poker Web Application

The goal is to create an interactive online application that puts the user at the centre and allows them to compete with the AI agent. Included in the scope are necessary gameplay features and the detailed hand review view.

AI Agent for Short Deck

The core focus and goal is to create an AI capable of understanding and engaging in Short Deck Poker. It will not cover other variations of poker.

Application Of CFR

The agent will be trained using Counterfactual Regret Minimisation (CFR) as its primary training strategy. This project will not explore alternative reinforcement learning algorithms outside of this scope.

1.5. Thesis Roadmap

This section will provide a brief overview of what the following chapters will be about.

2. *Literature Review*
   This section will explore the existing body of work surrounding AI applications in poker. It will examine the evolution of reinforcement learning techniques and their role in advancing computer poker. It will also highlight the key breakthroughs and existing projects related to my application, and the aspects of these implementations which aided the development of my project. I will also look at other popular poker applications to gain inspiration into their functionality and key features. Lastly, this section will outline and explain each technology that will be used in the project.

3. *System Design*
   This chapter outlines the architectural framework of the AI system. This includes the design of the AI agent, the choice of algorithm, the game state abstractions and the overall system workflow. It outlines and explains the chosen design methodology through the use of various UML diagrams.

4. *Software Development*
   This section will outline the software development process, it will describe every valuable aspect of the application, highlighting their purpose and techniques used to develop them.

5. *Testing and Evaluation*
   This section will outline the strategies for testing and evaluating the AI agent's performance and the web applications functionality.

6. *Issues and Future Work*
   The final chapter will discuss the issues faced when developing the prototype and will outline the future work to be carried out to achieve the final result.

2. Literature Review

2.1. Introduction

In this section I will be outlining and discussing the past research done involving artificial intelligence (AI) in poker, this research has been integral in the overall development of AI.

The exploration of Poker specific research began to grow in the late 1990s, especially by the Computer Poker Research Group in the University of Alberta. Research was also accelerated due to introduction of the Annual Computer Poker Competition in 2006, which led to many of the modern abstraction and algorithmic techniques used today which I have pointed out in the following sub sections.

I will also be exploring implementations of these research techniques which are available through online resources such as GitHub, I will address what aspects I like about their approach and aspects I will be utilising to help the development of my implementation.

Additionally, I'll be looking at various poker web applications which will have an impact on how I design aspects of my application.

In section 2.3. Technologies you've researched, I will be discussing the technologies I will be using and my thought process behind each decision, which aspects of each technology eventually led me to each decision.

In the conclusion of this section I will be outlining other areas of research which ultimately led me to this title and I will be exploring the similarities, successes and short comings of other final year projects.

2.2. Alternative Existing Solutions to Your Problem

2.2.1.Poki: Opponent Modelling: University of Alberta Computer Poker Research Group 1998

The University of Alberta led much of the early poker AI development and research, this was one of the earliest papers which focused on predicting the behaviour of other poker players. The approach of Poki was to estimate the effective hand strength of their opponent and their own hand with respect to the community cards, the agent's decision will then be adjusted based on the evaluation of the opponents strategy.

This approach worked effectively if the opponents strategy was straightforward, if the opponent was passive or overly aggressive in their play style, this AI agent would be perfectly suited to adjust their strategy to exploit the opponents predictable play style. Alternatively, Poki would be ineffective at facing a more modern and sophisticated play style which is commonly employed by the majority of players today. In today's game there's a huge focus on playing a balanced strategy which is used to become unpredictable and avoid this style of opponent modelling. If there isn't a clear pattern to how a player is playing it leaves the early method introduced by The University of Alberta Computer Poker Research Group to be ineffective in today's style of poker and ineffective for use in my project as I'm attempting a more modern approach to developing the AI agent.(6)

2.2.2 Approximating Game Theoretic Optimal Strategies for full scale poker 2003

This paper was the first to implement the use of abstraction to simplify the complexity of the game by reducing the number of information states. This is fundamental to all future implementations of a poker agent and is fundamental to my approach. The approach of this

agent is to iteratively play and their strategy is updated based on the previous plays and will eventually converge to the optimal strategy.
This method is effective when implemented in a game such as Heads up Limit Hold'em (HULHE), where the abstraction can make an effective approximation of the game while still maintaining the overall integrity of the original game. It also efficient at developing a strategy which is difficult for opponents to exploit the weaknesses in the strategy.
The primary drawback of this method is it may not preform optimally in real time search, where the opponent player has the ability to change their strategy as the agent will not be able to adjust based on the changing opponents strategy.(7)

2.2.3 Regret Minimization in Games with Incomplete Information. 2007

The introduction of the Annual Computer Poker Competition in 2006 was vital in the development and advancement in computer poker. This paper illustrates the widely used iterative Counterfactual Regret Minimisation (CFR) algorithm, which shows that through self-play the Nash equilibrium can be computed for the abstraction of Texas Hold'em with 10^12 states.
This paper was integral to the future development of Computer Poker and is the backbone of my project. The abstraction technique used with bucketing hands and expected hand strengths which was previously introduced, combined with the self-play algorithm converges to the Nash Equilibrium.
The only drawback of this technique for my implementation is the size of the game, 10^12 states is too large to be computed and iteratively played through to develop a strategy on my computer, I will have to decrease the size of the game using other techniques and a larger degree of abstraction. (8)

2.2.4 Heads-up Limit Hold'em Poker is Solved (Bowling et al) 2015

The authors of this paper used CFR to solve heads-up limit Hold'em poker, in their approach they didn't use any abstraction to limit the size of the game, due to the absence of abstraction they're able to get a more precise and realistic representation of the overall game. The absence of abstraction required significant computational power to calculate the strategy of the agent, the amount of computational power is the primary reason I would be unable to implement this approach, abstraction will be vital in my implementation as seen from the previous papers I've explored.
This solution works well if significant precision is required, considering the agent is trained in the full game of poker and not a simplified version, it will be accurate and precise and won't be approximating the strategy based on an abstracted game. It's also an effective solution if exte4nsive computational resources are available. (9)

2.2.5 GitHub: CFR Implementations

This is a considerably simplified poker variant. The game contains two players and 13 cards of one suit, which contains one betting round. Although it's a simpler poker game, being able to see how the simplified game operates helps when playing the full poker game. This implementation was done using C++ rather than Python, C++ has its clear advantages in speed and performance but I have chosen to use python due to the simplified syntax and readability which is vital when developing a complex agent. I will go into further detail about the pros and cons of python versus C++ in section 2.3. Technologies you've researched. This

program largely based on a paper released in 2013 which outlines the implementation of CFR in one-card poker called Kuhn poker. (10) (11)

The second implementation I took inspiration from for my project was a project called ExePokerWeb, which is a very simple web application which allows the user to play against an AI agent in No-Limit Texas Hold'em. The project takes a lot inspiration from the 2007 paper published by the University of Alberta (12). I took strong inspiration from this project, firstly in their use of the custom pickle module to serialise and save the data in a file for later retrieval, in my project I will use the Dill module which is a newer version of the pickle module. Additionally, this implementation showed me some of the logic requirements needed to implement a variant of poker using the CFR algorithm, it also guided me on the overall flow of the poker web application aspect and how it should work. This implementation of the CFR algorithm also influenced how I set my version up, even though they're the same algorithm there are some differences in the overall set up, but seeing a working version of it definitely helped in creating mine. Although, they used a different variant of poker there are a number of similarities between the different types of poker. In section 4.2.1.5.3 Hand Evaluation / Poker Logic, I highlighted the help I received from functions from this implementation.

2.2.6 Poker web application GG Poker

My project's user interface will take inspiration from the leading platform GG Poker. They use vibrant visuals and interactive elements to make their platform an immersive experience. The interface is very intuitive, it ensures easy navigation and clear gameplay instructions, it's also at the top when it comes to performance, the platform is fast and responsive which enables seamless gameplay. This research guides my design to balance the functionality and aesthetics.(13)

Figure 1: GGPoker UI Table

2.2.7 Poker Application Game statistics

The player statistics component of my application was influenced by the popular poker statistics website SmartHand, which provides players with a graphical and numeric breakdown of their online play. The service offers valuable insights into game patterns and player's strategies.

I found the website influential in designing the game statistics element of my project, it helped show how to effectively present statistical data in an accessible way ensuring the information provided was detailed but also easy for the user to understand the insight provided. (14)

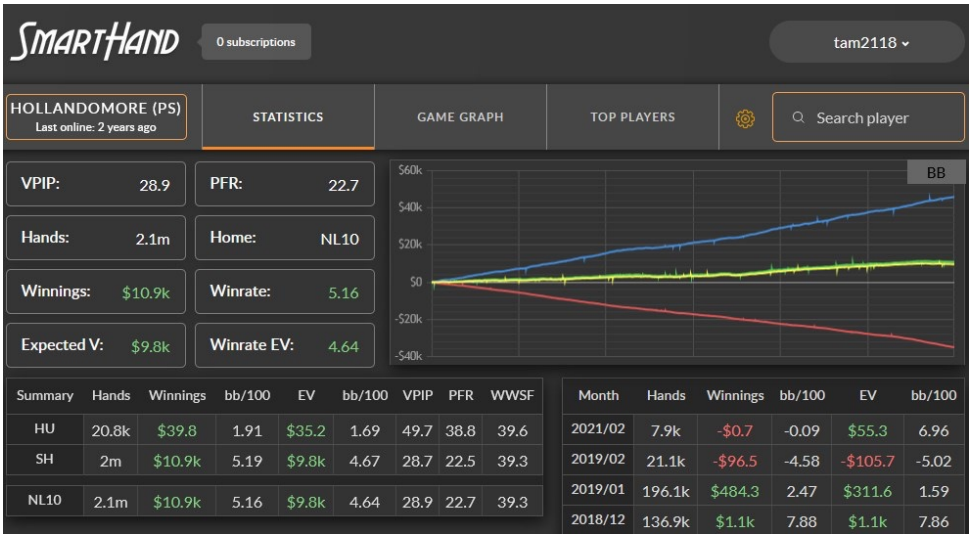


Figure 2: SmartHand Game statistics

2.2.8 Poker Application Hand Review

The hand review aspect of my application is an essential element for players to view their previously played hands and actions taken during the hands. This section not only provides great insight into the decisions the user made but also displays great detail into the decision making process of the AI.

The primary design of the section came from a couple of sources but primarily from the ICMIZER Replayer, which is a hand history replay tool "for replaying hands you've played before". Which shows a list of hands played, calculates the pot odds for each decision and many other features to provide the user with insight. The overall user interface and design greatly influenced the layout of my application, alternatively, the detail is different in my application and this tool, with mine being more focused on the AI's decision. (15)

Figure 3: ICMIZER Replayer - Hand Review

2.2.9.Poker Pre-flop Hand Strength

I did a lot of research into the strength of hands and pre-flop ranges, as it was vital for the hand bucketing abstraction, where hands of equal strength and value are treated as the same value. By categorising hands into buckets its greatly simplifies the game for the AI without removing the overall integrity of the decisions.

The CardMates website provided the greatest detail into the optimal heads – up strategy used by players, many other sources provided the hand strengths for multi-player poker which is a very different game. I used a series of calculations based upon the card attributes such as value and suit to create a close representation of the optimal pre flop strategy according to CardMates. (16)

Figure 4: CardMates - Pre-Flop Hand strength

2.3. Technologies you've researched

In this section, I inspect the key technologies and methodologies that form the backbone of my project. These choices are essential in software development, especially when they involve complex aspect such as artificial intelligence and reinforcement learning. The technologies explored have been carefully selected based on their suitability in overcoming the challenges of the project.

2.3.1 Programming languages:

Python

Python is a simple programming language which is very well suited for AI and machine learning projects. It has a vast ecosystem of libraries and frameworks that make complex tasks far easier to implement and understand, Python is also ideal for rapid prototyping once again due to its simplicity and readability.

Given high level nature of Python, the execution time can be quite slow in comparison to a language like C++. This is a drawback in my project when training the AI agent, the slower nature of python makes the task more computationally intense. Aside from the slower execution time, Python is a great programming language for my specific project, and will be the programming language I will use. (17)

C++

On the other hand, C++ offers better memory management and faster execution, when using C++ you also have more control over system recourses. These are key aspects when building a performance intensive game engine and training an AI agent. Although there are many positives tailored towards my project, I'm not very familiar with the C++ language and due to its steep learning curve and the complex nature of my project, it's not the ideal programming language to use for my project. (18)

2.3.2 Python Libraries:

Dill

The Dill module is used to serialise and deserialise python objects and supports the preservation of complex data types which Python may be unable to handle.

In my application, it's used to serialise the state of Node Manager objects, which encapsulate the training data for the AI. When serialised, the complete state of the objects are saved to a file, which are later deserialised for use by the AI in making decisions.(19)

NumPy

NumPy is a python library which is vital for numerical computing, its wide range of mathematical functions enable efficient data manipulation and analysis. Its application was essential in creating the grid for my pre-flop hand strengths.(20)

Itertools

This Python module, provides a range of tools for creating iterators to help with efficient looping, it's very useful in efficient algorithm design. The combinations tool was used in my project to enable efficient calculation of the hand value; the five best cards made up of the two cards and the five community cards combined.(21)

2.3.3 Framework:

Flask

Flask is a lightweight and minimalist framework which offers all the basic tools required for small-sized web application. Flask doesn't enforce any dependencies which promotes flexibility and control over the projects structure. However, this does imply greater responsibility in manually setting up and integrating the chosen database. Flask is ideal for single page applications that don't require any dynamic HTML, these were the main reason I decided against using Flask as the framework.(22)

Django

Django is a robust and scalable framework which is designed for more complex applications. It is known as the "Batteries included" framework due to the database schema migrations which allows developers to interact with the database using python rather than the SQL itself.

Django is tailored towards multi-page applications which is integral to the design of my project, even though my project will be running predominately on one page. It also allows for dynamic HTML, which is also essential, my page will have a number of other pages which will provide more functionality to the application and Flask would not be adequate to support this.(22)

2.3.5 Database

PostgreSQL

PostgreSQL is a robust relational database renowned for its reliability, ensuring transactional integrity operational security which ais paramount when dealing with user accounts and gameplay records and statistics. PostgreSQL adherence to SQL standards allows it to integrate seamlessly with the Django framework.(23)

2.3.6 Front End

HTML

HTML is "the standard markup language for web pages", it's designed to be displayed in a web browser. I decided HTML would be a good choice as a key component of the front-end of my application as its syntax is quite straightforward and I'm relatively comfortable using it.

Considering the other complex aspects of the web application, I felt it was necessary to have elements I found easy to use and it does the job accordingly, with all things considered there was no reason to look elsewhere.(24)

CSS

Cascading Style Sheets (CSS) is a "style sheet language used for specifying the presentation and styling of a document written in HTML". Considering this definition, it seemed the perfect language to accompany the markup language, a pair I have used together many time throughout my 4 years of studying. There are a couple of frameworks such as tailwind or bootstrap which I will likely utilise to speed up and enhance the styling of my front end.(25)

JavaScript / chart.js

JavaScript (JS) is a "core technology of the web", and so it was essential to include in my application to enhance the client side of the application. JS is nearly always used alongside HTML and CSS, "99% of website use JavaScript", with all things considered it was vital I included it in my application. It would be essential to get various features working, such as

the animations and the game statistics charts. For the development of the charts, I will be honing the powers of Chart.js, which makes these takes far easier and more aesthetically pleasing. (26)(27)

2.4. Other Research you've done

When beginning my understanding of reinforcement learning (RL), I was led to the knowledge within David Silver's series on RL. His lectures at UCL offered a deep dive into the sophisticated mechanics of RL algorithms. The insight I learnt from studying the lectures provided the tools required to create the Poker AI framework for my project. I began studying these lectures during my internship period in third year, these lectures were the initial inspiration to design a project around the idea of reinforcement learning. (4)

Equipped with the basic understanding of reinforcement learning, I began researching applications of reinforcement learning. One of the main ideas I looked at were the applications in the environmental sector. In particular in waste management, I explored the optimisation of domestic waste collection routes. This involves making the waste collection process as resource efficient as possible by optimising the routes the waste collection vehicles would take. However, I was unable to find any relevant data that would make this possible, it's also challenging for Dublin, as there are multiple individual domestic waste collection companies who are unlikely to provide the required data. There was also a huge amount of risk involved in this project idea due to the complexity of the task and the difficulty in presenting the project in action. (28)(29)

Although the research into these topics wasn't directly related to developing a poker AI agent, they greatly assisted in my understanding of reinforcement learning and its applications in the real world.

They showed the parallels between game specific agents and real world task specific agents, the differences being the environment they are working in and the actions they take. In waste collection, the agents environment is the area they collect waste in and the actions are, continue straight, turn left, turn right and stop. Their job is to optimise the route to make it as optimal as possible, in poker the goal is to optimise the strategy to make it as optimal as possible. The similarities between these problems greatly improved my clarity on reinforcement learning overall.

2.5. Existing Final Year Projects

Title: MindGame: A Machine Learning Framework for Electronic Games (30)

Student: Simon O'Neill

Description (brief): The student developed an AI framework to play multiple games themselves without human interaction.

What is complex in this project: The complexity in this project comes in the learning algorithms and the struggle with large form games where he stated that the program crashed when dealing with many game states

What technical architecture was used: He used a design matrix which later evolved after testing for the agent to make decisions The genetic algorithm was used to converge to the best response. He also has the game manager which was the game engine for each game.

Explain key strengths and weaknesses of this project, as you see it:

The strength of this project is in the complexity of each technical aspect and the success of the implementation and the analysis of each games performance.
I can't see any clear weaknesses, the only downside was that the sauce invaders agent didn't seem to play has he had hoped but the complexity of that game is far greater than the others.

Title: Conversional Agent in Virtual Reality (31)

Student: Andrew McCormack

Description (brief): The aim was to implement a system that will allow users to communicate with an AI agent from within a virtual environment.

What is complex in this project: The complexity in this project is in the conversional AI agent and getting the data required to train the agent to understand long form conversation.

What technical architecture was used: The system architecture was the Unity Game Engine and the Conversional Agent AI, there were smaller components relevant but these were the main two pieces of architecture.

Explain key strengths and weaknesses of this project, as you see it: The strength of this project was the overall complexity and the VR component, the weakness was the limitation on the AI agent itself, it could deal with small inputs but was unable to grasp context or longer form messages.

2.6. Conclusions

The literature reviewed in this section has been vital in shaping my understanding of AI development in poker, especially in terms of abstraction and the use of CFR. It also introduced other poker related web applications and tools widely used by avid poker players to enhance their game.

The development of AI in poker began with the early works of the University of Alberta's Computer Poker Research Group and then went to the great achievements of projects like Pluribus, which illustrates a significant evolution in AI capabilities and strategies.

The ability of SmartHand to display the player's game statistics in an informative but simple manner and how ICMIZER organised their website to show a large amount of valuable information about previously played hands will certainly help in deciding my overall design and layout.

Additionally, the GitHub repositories I researched have helped guide me on the development of my web application. Seeing another implementation of a similar project has enabled me to think of certain features differently, that I possibly wouldn't have thought of previously. This played an important role in my literature review.

In conclusion, this literature review has laid a solid foundation for the development of my AI agent. It has provided critical insights into the challenges and strategies of AI in poker and its appliance to a web application.

3. Experiment / Software Design

3.1 Introduction

In this chapter, the overall design of my project will be outlined. Firstly, the software methodology will be detailed and discussed, the reasoning behind the programming languages and frameworks used, visual representations of the game engine and the flow of the web application will be explained.

3.2. Software Methodology

3.2.1 Waterfall Model

The waterfall model is a structured approach, which is categorised by a sequence of events that flow downwards like a waterfall. Each step lays the groundwork for the following step. (32)

- At the first stage, specifications for the software are identified and recorded which lays the overall groundwork for the next phases.
- The analysis phase is where the system undergoes an examination to produce models, the structure and functions of the application are informed by these models.
- The third phase is where the technical architecture is mapped out. Decisions on the programming language and which technologies will be used are made in this phase.
- The coding phase involves turning the previously mapped out design into actual program code, while ensuring that the program logic is aligned with the design logic.
- The testing phase is the stage of the process where the software undergoes rigorous tests to identify any defects, faults or areas that require improvement in the software.
- The second last phase illustrates the steps taken to introduce the software to its environment and finally making the software accessible to the end users.
- The maintenance phase is the final stage in the waterfall model, this is the phase after development. It involves ensuring the software remains compatible with the evolving system or user requirements.

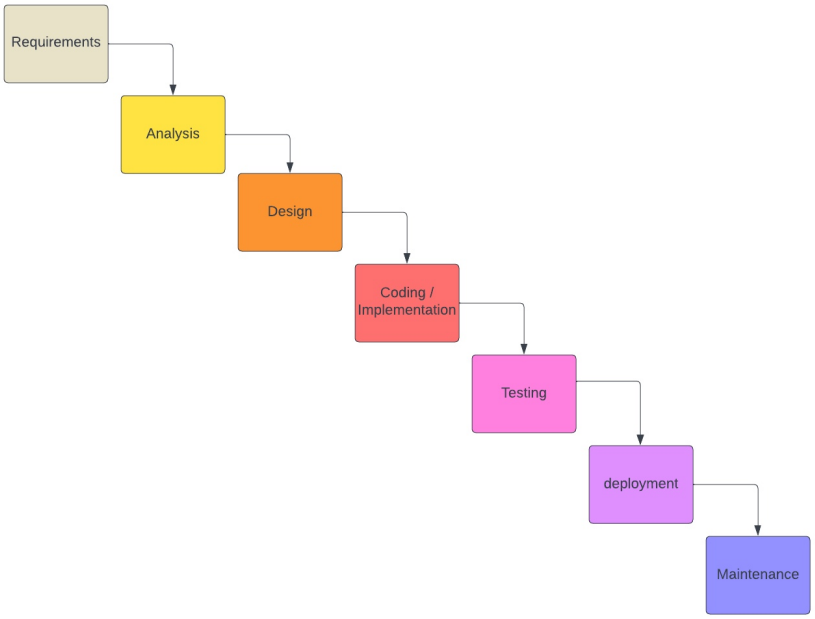


Figure 5: Waterfall Model

3.2.2 Scrum

The scrum methodology is an agile framework aiming to deliver new software features every 2-4 weeks. The time scale is predefined for the required work, this method enhances the focus and productivity of the team. Scrum reduces the challenges of underestimations of time, which is common in a large scale project.(33)

- The sprint is a defined period of usually 2-4 weeks, where specific tasks are completed for a set deadline.
- Sprint planning is a meeting which takes place to determine which aspects of the sprint backlog need to be addressed.
- Daily review is a short daily meeting where team members update on their progress, plans and potential issues they may have.
- Sprint review is where the team get to showcase the sprint's accomplishments, allowing the customer to provide feedback.

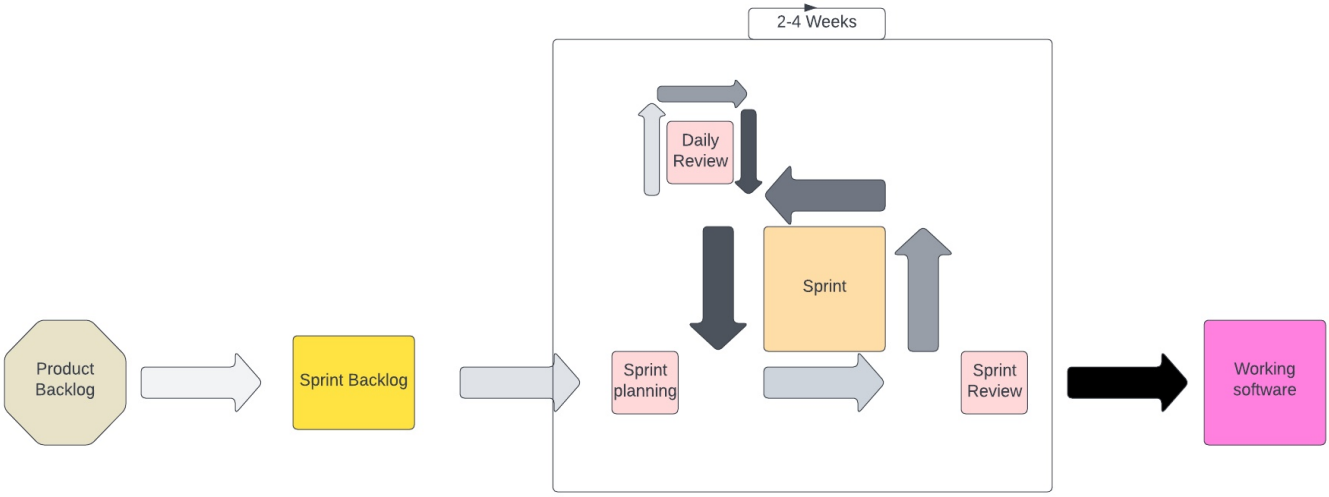


Figure 6: Scrum Methodology

3.2.3 Feature Driven Development

Feature driven development is a model-driven process that prioritises building features. The emphasis of this methodology is on efficiency through progress tracking.(34)

The structure is broken down into these key stages:

- Developing an overall model is created by understanding the domain and major functionalities that make up the software.
- Building a feature list is done by listing all the potential features which represent small functional units of the system are identified and categorised.
- Once the list has been completed, a detailed plan is crafted which schedules how and when each feature will be developed.
- Designing by feature is done by ensuring that the design of each feature is carefully constructed and ensures alignment with the systems overall architecture.
- Building by the feature ensures that actual coding takes place, and each feature is built and integrated into the evolving system. Consistent feedback ensures quality and alignment.

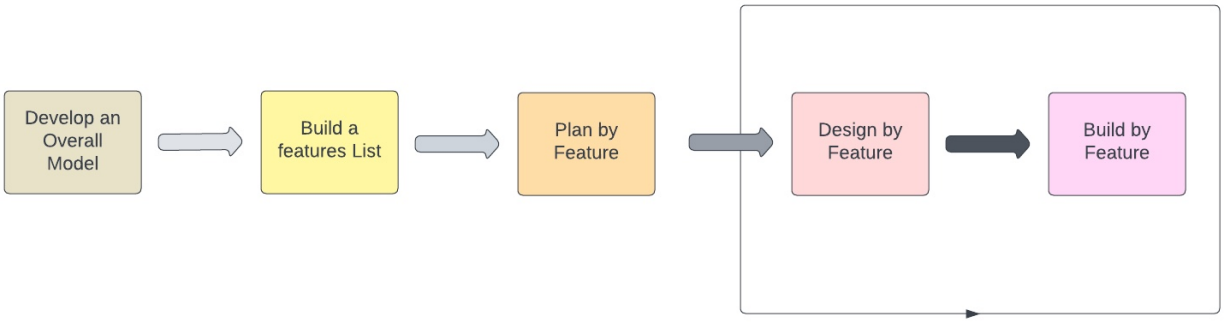


Figure 7: Feature Driven Methodology

3.2.4 Chosen Methodology

I initially considered a waterfall model due to its linear approach which would provide clear management and planning, and its strict requirements and clear direction at each phase which I thought would be beneficial for a large scale project. Subsequently, I found the waterfall model to be too rigid an approach for my particular project. Given the experimental nature of training an AI agent, the difficulty in going back and making frequent changes, ultimately I decided on an agile approach.

My project will use an agile based methodology, this methodology is well suited to this project due to the iterative nature of agile. This approach accommodates evolving requirements and encourages regular feedback, which is essential when training an artificial intelligence agent that must be consistently learning and improving over time through iteration.

The methodology I will be using in my project is a combination of feature driven development and scrum. There are two primary features that make up my project, the web application and the AI agent. Within these primary features, there are many sub features which make up these primary features. The first step in the process is to build an overall model of the project. The model will be outlined in the following sections in system overview, which will give a description of how the system will operate.

The next step will be outlining the list of features which make up the model. These features are split into two sections, the AI agent and the Web application, each section has many sub sections which can be seen in Figure 8: Feature List, each sub feature will be looked at in detail in a later section. The planning of the features is the next stage in the methodology, at this stage each listed feature is of equal importance but it makes sense to first focus on the user interface, followed by the game engine, and then the AI integration and training refinement.

The final two features involve the designing and building of the individual features, the process of the development of each will be shown in later sections.

At the beginning of each 1-2 week block, I will have a sprint planning session, where I will decide on a set of sub features to complete and detail the tasks required for each sub feature. To conclude the sprint block, I will review the completed features and the undelivered features to see how well the objective were achieved.

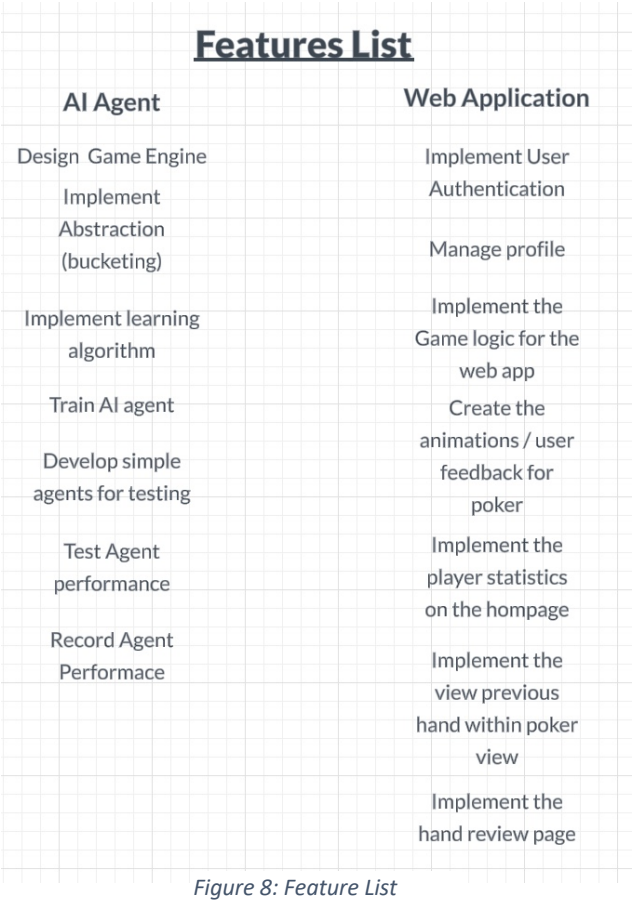


Figure 8: Feature List

3.3. Requirements Gathering

During the early stages of developing a web application, the development of a prototype is a vital step in the requirements gathering process. The prototype acts as an early stage representation of the application, highlighting the core functionality.
In my prototype, I utilised the robust Django framework, along with HTML and CSS. This prototype was primarily focused on getting the Django framework up and running as well as displaying the basic functionality such as the cards being dealt to the user and AI.

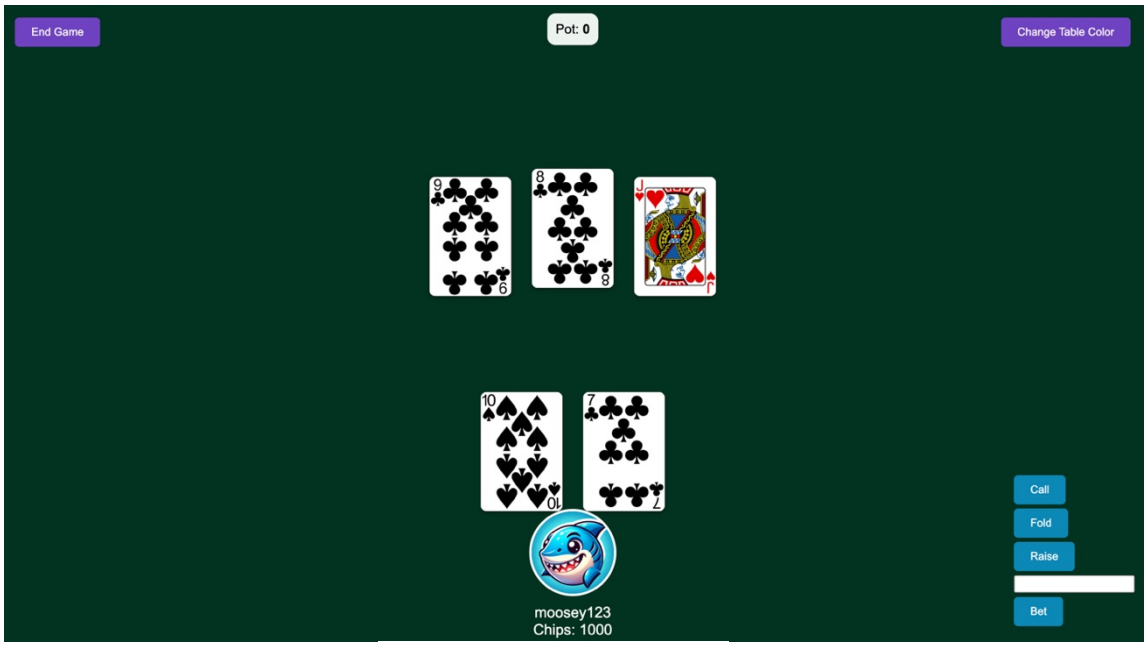


Figure 9: Play Poker Prototype

Additionally, the prototype allows the user to login and manage their profile which are essential features in the current iteration of the application. Furthermore, the PostgreSQL database sits within the Docker environment, this is also the setup used in the current iteration of the application. Developing the prototype of the application was an essential step in gathering knowledge and experience of the domain I would be working with. The prototype helped visualise what aspects of the application may look like and enabled me to test out different technologies and eventually decide on the technologies I'm currently using.

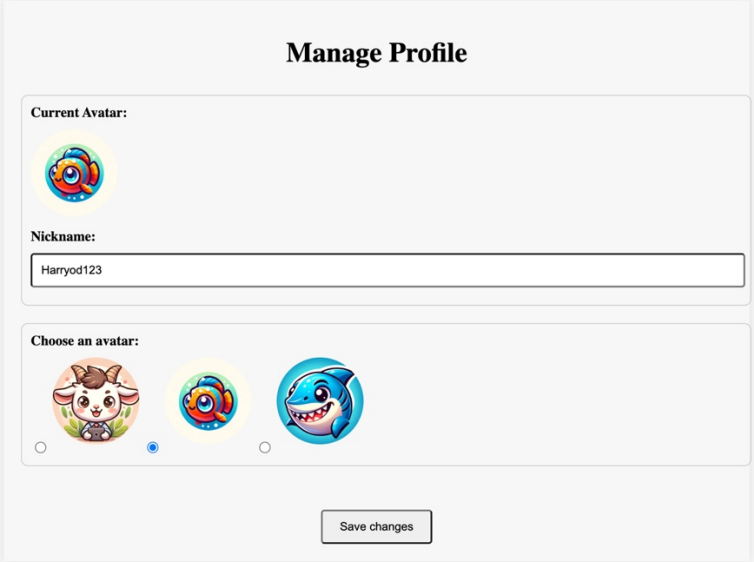


Figure 10: Manage Profile Prototype

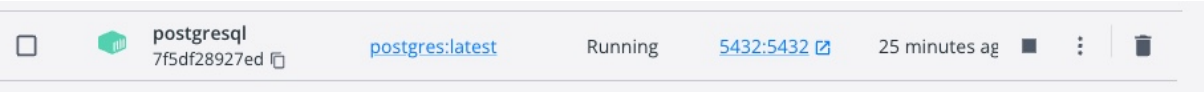


Figure 11: Docker Instance

The primary purpose for building a prototype was verifying the efficacy of the learning algorithm and ensuring I was able to implement a version of it. In this instance, creating a prototype that demonstrates an AI's learning process is both a foundational and a strategic endeavour. The AI poker agent prototype will use the same reinforcement learning algorithm that will be used in the final project, but I have used a much simpler game called Kuhn poker to convey the agent learning in action. The development was based off the research done on the Counterfactual Regret Minimisation (10), in their paper they provided pseudocode which I used to write the algorithm to train the agent in python. Kuhn poker is a two-player variant which uses a three card deck (1,2,3). Each player pays the Ante of one chip and then is dealt a card from the deck, players then have the choice to pass or bet a chip. The game reaches the terminal state if one player bets and the other passes or if both pass or both players bet, the highest card wins that round. The agents goal is to learn a strategy by deciding whether to bet or pass based on the player's cards and their opponent's strategy. This prototype serves as a foundation for the more complex final project, as the learning algorithm in foundationally the same for the final project.

```python
def cfr(cards, history, p0, p1, node_map):
    plays = len(history)
    player = plays % 2
    opponent = 1 - player

    # Terminal state payoff
    if plays > 1:
        terminal_pass = history[-1] == 'p'
        double_bet = history[-2:] == "bb"
        is_player_card_higher = cards[player] > cards[opponent]
        if terminal_pass:
            if history == "pp":
                return 1 if is_player_card_higher else -1
            else:
                return 1
        elif double_bet:
            return 2 if is_player_card_higher else -2

    #Construct the information set for the current player
    info_set = str(cards[player]) + history

    #Create a new node in the node_map if it doesn't exist
    if info_set not in node_map:
        node_map[info_set] = Node(info_set)

    #Get the node for the current information set and get the players strategy f
    node = node_map[info_set]
    strategy = node.get_strategy(p0 if player == 0 else p1)

    #Recursively call cfr with additional history and probability
    util = [0.0 for _ in range(NUM_ACTIONS)]
    node_util = 0
    for a in range(NUM_ACTIONS):
        next_history = history + ('p' if a == 0 else 'b')
        if player == 0:
            util[a] = -cfr(cards, next_history, p0 * strategy[a], p1, node_map)
        else:
            util[a] = -cfr(cards, next_history, p0, p1 * strategy[a], node_map)
        node_util += strategy[a] * util[a]

    # Update regrets for the curretn node
    for a in range(NUM_ACTIONS):
        regret = util[a] - node_util
        node.regret_sum[a] += (p1 if player == 0 else p0) * regret
    #retrun the utility
    return node_util
```

Figure 13: CFR Prototype Code

Algorithm 1 Counterfactual Regret Minimization (with chance sampling)

Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
Initialize initial profile: $\sigma^1(I,a) \leftarrow 1/|A(I)|$

function CFR(h, i, t, π_1, π_2):
if h is terminal **then**
  return $u_i(h)$
else if h is a chance node **then**
  Sample a single outcome $a \sim \sigma_c(h,a)$
  return CFR(ha, i, t, π_1, π_2)
end if
Let I be the information set containing h.
$v_\sigma \leftarrow 0$
$v_{\sigma_{I\to a}}[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ **do**
  if $P(h) = 1$ **then**
    $v_{\sigma_{I\to a}}[a] \leftarrow \text{CFR}(ha, i, t, \sigma^t(I,a) \cdot \pi_1, \pi_2)$
  else if $P(h) = 2$ **then**
    $v_{\sigma_{I\to a}}[a] \leftarrow \text{CFR}(ha, i, t, \pi_1, \sigma^t(I,a) \cdot \pi_2)$
  end if
  $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_{I\to a}}[a]$
end for
if $P(h) = i$ **then**
  for $a \in A(I)$ **do**
    $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I\to a}}[a] - v_\sigma)$
    $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
  end for
  $\sigma^{t+1}(I) \leftarrow$ regret-matching values computed using Equation 5 and regret
end if
return v_σ

function Solve():
for $t = \{1, 2, 3, \ldots, T\}$ **do**
  for $i \in \{1, 2\}$ **do**
    CFR($\emptyset$, i, t, 1, 1)
  end for
end for

Figure 12: CFR Pseudocode from research

The CFR function is responsible for traversing the game tree, calculating regrets and updating strategies. It begins with the function parameters, p0 and p1 are the probabilities of reaching the current information set for the players and the node_map, maps the information sets to their corresponding nodes, these nodes store the regrets and the strategies.

The next part of the function is about determining the states the end the game, which can happen if the last action is a pass or there's two successive passes or bets, in the instance where the game ends on a bet, the cards are evaluated by their strength and the pay-outs are given. In the other instances the pay0outs are given to the player who didn't pass.

The next section of the function is where the strategies are updated and future states are explored. **strategy = node.get_strategy(p0 if player == 0 else p1),** the current strategy is retrieved using get_strategy. Future states are then explored by recursively calling 'CFR' for every possible action (pass, bet) and then adjusting the probabilities (p0, p1) based on the current strategy. The final part of the function is where the regret of taking each action is updated. This is calculated by the difference between he the utility and the expected utility of that action,

This implementation of CFR used in the prototype will have a major influence on the final implementation I will use in my final project, the only differences will be the handling of the data such as the actions, defining the terminal node and storing the history, to name of a few minor differences.

3.4. Overview of System

This represents the system architecture of my application, a three-tiered structure designed for efficiency and scalability. At the forefront, the presentation layer, which provides the user interface, in this instance this will be the poker table where the players interact with the AI agent. The user interface is built to respond to user inputs, displaying the different game states and offering the player interactive features such as betting and folding.

At the centre of the system is the application layer, In this layer the system interprets the game's rules, manages the flow of the poker betting rounds and applies the AI's logic to the game. This layer is implemented in python using the Django framework.

The bottom layer, the data access layer, is assigned all the data interaction tasks. This layer manages the storage of game related data, such as user profiles, game history and AI's training decisions.

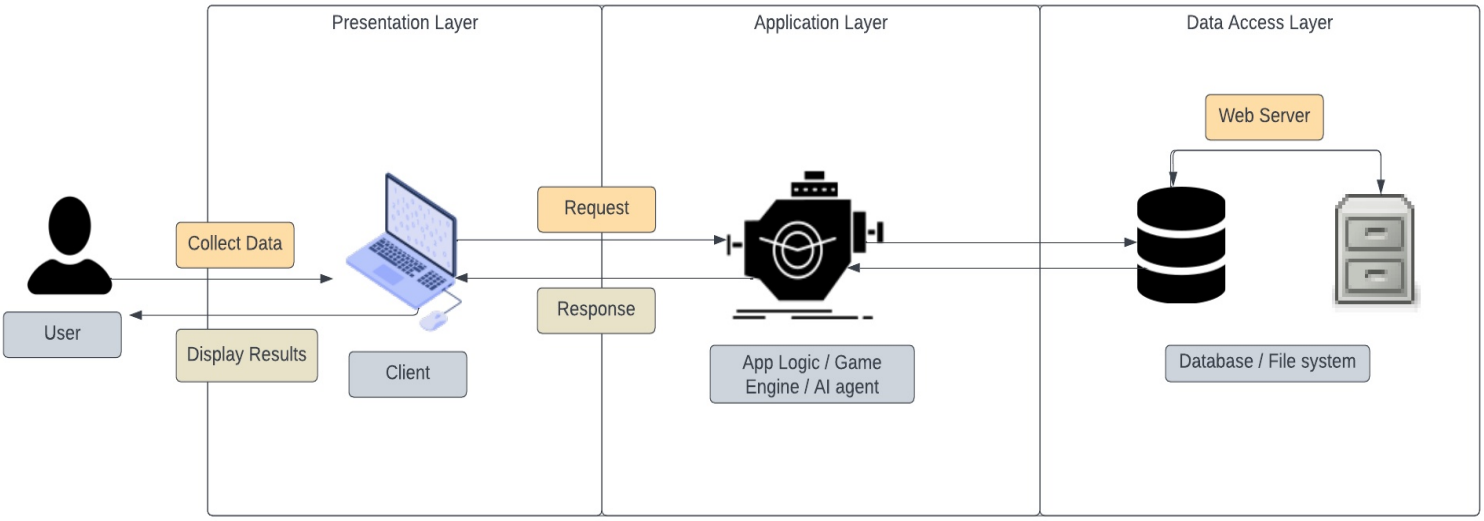


Figure 14: System Overview

3.4.1 Front End

3.4.1.1 Use Case Diagram (Basic)

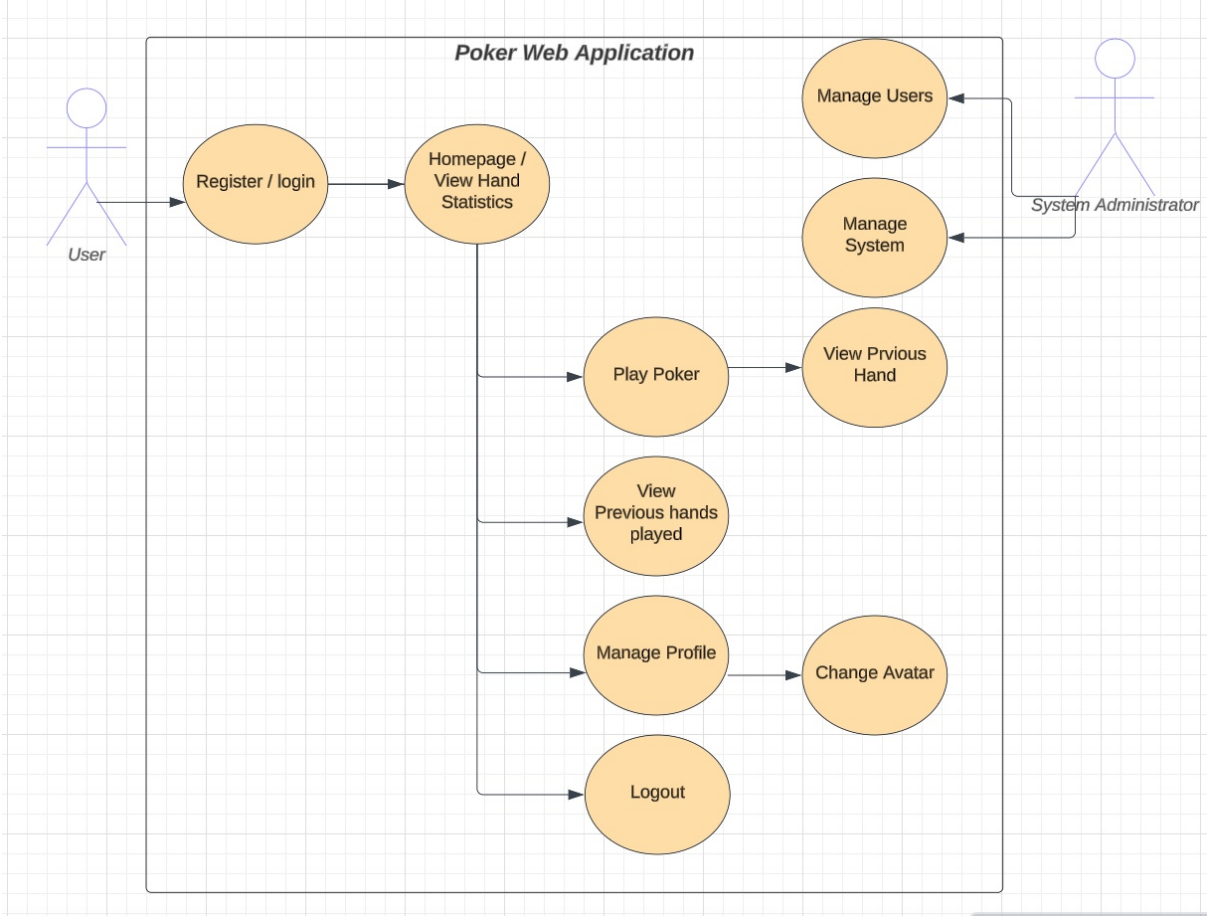


Figure 15: Basic application Use Case Diagram

The front-end design is displayed in the use-case diagram, it encapsulates the range of activities that the user can engage in within the web application.

It begins with user registration / login, this enables new players access to the application by prompting them for personal information and also allows returning players to enter their stored details to regain access. This access point is designed with simplicity and efficiency at the centre, enabling a frictionless account creation or re-entry.

Once logged in, the players will be faced with a homepage, which displays the players game dashboard. The dashboard allows players to reflect on their performance through looking at the various statistics and graphs about the games played that have been recorded, see *Figure 20: Game Dashboard wireframe*, for more information on the game dashboard.

Additionally, within the homepage the user will be faced with a series of menu options. The primary function of the menu is the ‘Play Poker’ use case, this is where the core experience takes place as players are immersed in their personal poker room. Here, they will play Short Deck poker against a trained AI agent. They will also have the option to view the previous hand they have played, to analyse and learn from the actions taken and the decisions the AI has made.

Furthermore, The view previous hands played use case facilitates player reflection and learning by allowing them to review past hands played. Users can analyse their actions taken,

as well as the decision making process and information which led to the AI's choices, which can aid in improving their future game strategies. This feature is critical for players who want to enhance their skills by identifying areas for improvement. It enriches the player's experience by adding an educational layer to the platform.

The Manage profile use case allows the user to adjust personal details such as their password. They will also have the ability to personalise their presence within the application, they will be able to create a nickname which will be displayed when playing a game and also attach an avatar to their profile from the varying range of avatars that are found on the change avatar view. This function provides users with the tools to express their individuality.

The system administrator's role encapsulates the management functions. These use cases are vital to maintaining the applications integrity, ensuring that user experiences are smooth and system operations remain efficient.

Finally, the Logout use case signifies the closure of the user's session. This process ensures that session information is safely secured and stored.

3.4.1.2 Activity Diagram (Playing a hand)

The activity diagram for playing a hand displays the flow of the game in the web application. It maps out the sequence of actions a player can take during a hand of poker.

The first stage to any hand of poker is the dealers deck shuffle and the cards being dealt. The players will then engage in a round of pre-flop betting (before any of the community card are shown). Here the player has a series of decisions based on their assessment of their hand strength, in this instance a 'call' represents calling the bet of the AI agent or checking (both players don't bet and they move onto the next round), if the player folds the hand terminates and the other player wins the chips.

As the diagram progresses, the 'deal flop' activity unveils the first three community cards. Once the cards are shown, the flop betting activity commences, this provided the players further opportunity to evaluate their hand strength and make a decision based on their odds of winning. The decision are the same as the previous round of betting, each decision branching off to another phase.

The 'deal turn' and 'deal river' bring the fourth and fifth community cards and conclude the cards being dealt for this round, each have their own separate betting rounds. Each action the player makes brings them closer to the eventual hand winner evaluation and showdown where the winner will be decided. Once the best hand is identified and the winner is determined they collect the chips from the pot.

This activity diagram outlines the blueprint that will inform the development of the game logic and user interaction. Ensuring integrity of the actual flow and rules of poker, from the initial fair and random dealing of cards to the outcome evaluation.

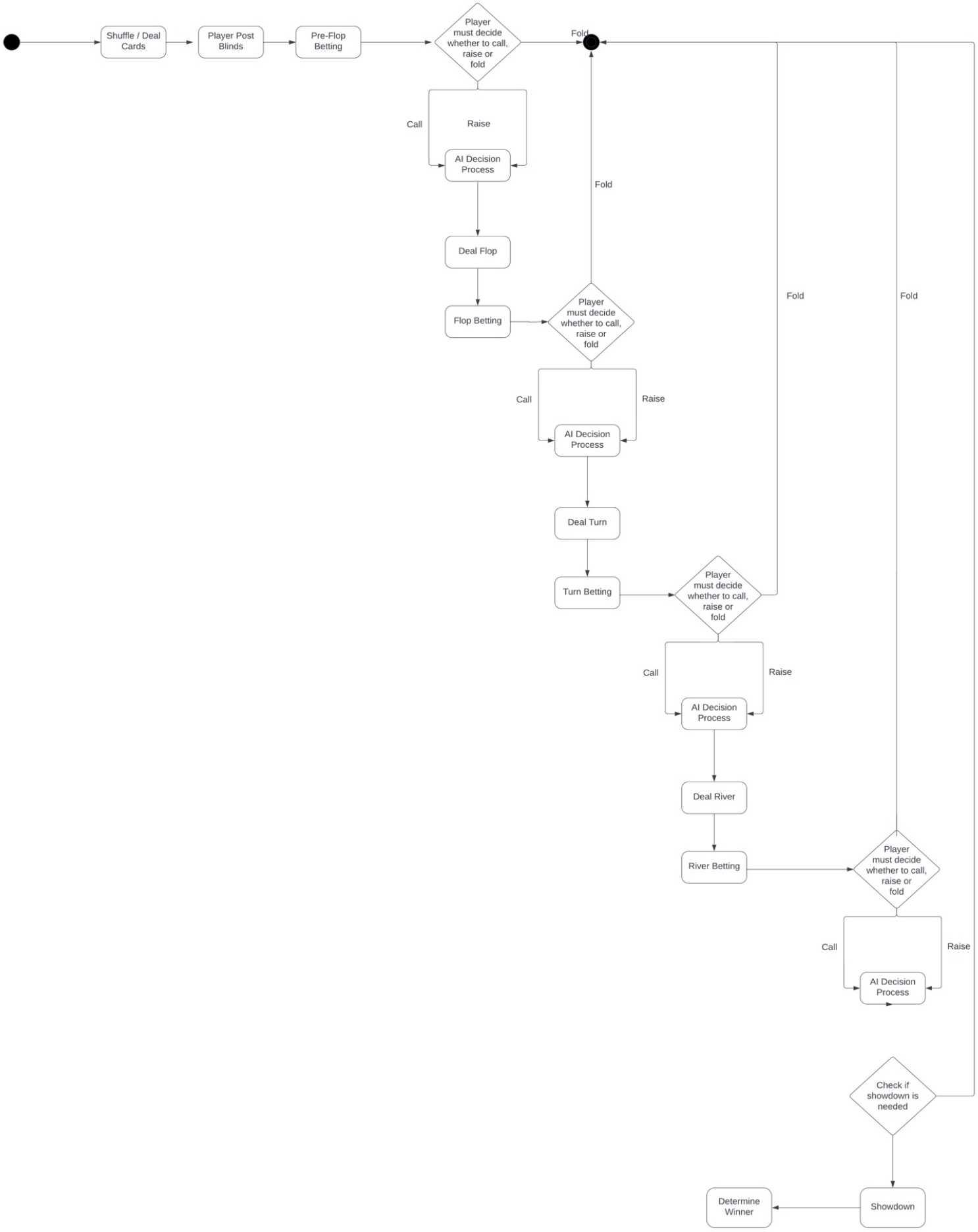


Figure 16: Playing a hand activity diagram

3.4.1.3 Game Statistics

In the player statistics section of the user interface, vital metrics are presented that offer an overview of the players performance. The player overview display the total number of hands played, the provides insight into the players experience. The win percentage is also tracked, it reflects the proportion of hands where the player was victorious. This metric reflects the overall success of the player at the table, it's the ratio of hands won to hands played.

The middle section delves deeper into the strategic habits of the player, this section provides the player with important detail that can be used to alter their strategy to improve their overall game.

- **Voluntarily put money in the pot (VPIP):** This metric reveals the player's tendency to contribute to the pot off their own accord during the pre-flop phase of the game, this excludes the obligatory blind posting. This statistic emphasises the player's tendency to actively engage with the game.

$$VPIP = \left(\frac{\text{Voluntary Contributions}}{\text{Hands Played}}\right) \times 100$$

Figure 17: VPIP calculation

- **Pre-Flop Raise (PFR):** This statistic tracks the frequency with which a player raises pre-flop. Similar to VPIP except only includes the raises and not the other actions.

$$PFR = \left(\frac{\text{Pre-Flop Raises}}{\text{Hands Played}}\right) \times 100$$

Figure 18: PFR calculation

- **Aggression factor (AF):** This metric is the frequency of aggressive actions, betting or raising, to passive actions, calling or checking. The higher the frequency, the more aggressive the player.

$$AF = \frac{\text{Number of Bets+Number of Raises}}{\text{Number of Calls}}$$

Figure 19: AF calculation

Each of these statistics provides insight into various aspects of a player's playing style ad strategy. They can assist players in identifying areas of improvement, such as playing too many hands (high VPIP) or being too passive and conservative (low PFR or AF).

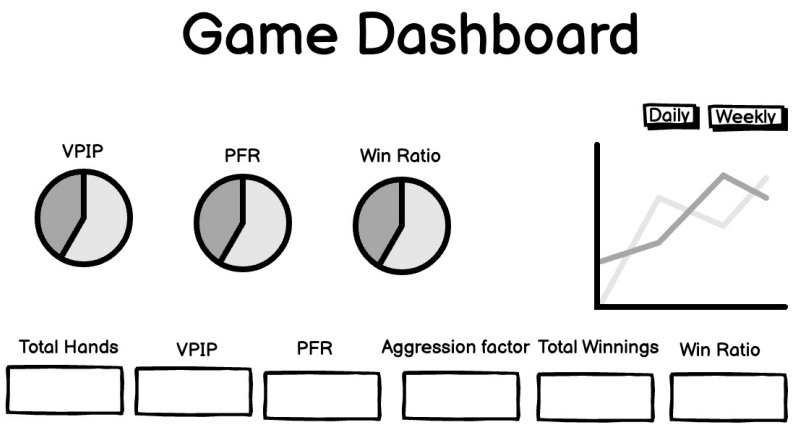


Figure 20: Game Dashboard wireframe

3.4.1.4 Manage Profile

The wireframe provided depicts the mange profile use case, users have the ability to customise and change their personal profile. This section empowers the users with the ability to control their in-game appearance.

The nickname and avatar can be changed at any time, these are displayed when playing poker within the game environment. The option to change the avatar, redirects the player to a new page where they have the ability to choose an avatar from displayed list, this gives the player a sense of ownership and provides an element of fun to the profile. The user also has the same license to change the nickname, the player may wish to link the nickname and avatar which is why this customisability option was vital.

The manage profile page, will display the username, first name, last name, email and location. They will be unable to change these account attributes, apart from the nickname. However, the user will have the option to change their password, they will be required to enter the current password and then their new password twice, this password management approach is a key component of account security.

Lastly, the players experience level will be shown in the top right corner of the screen, this is calculated based on how many hands they have played.

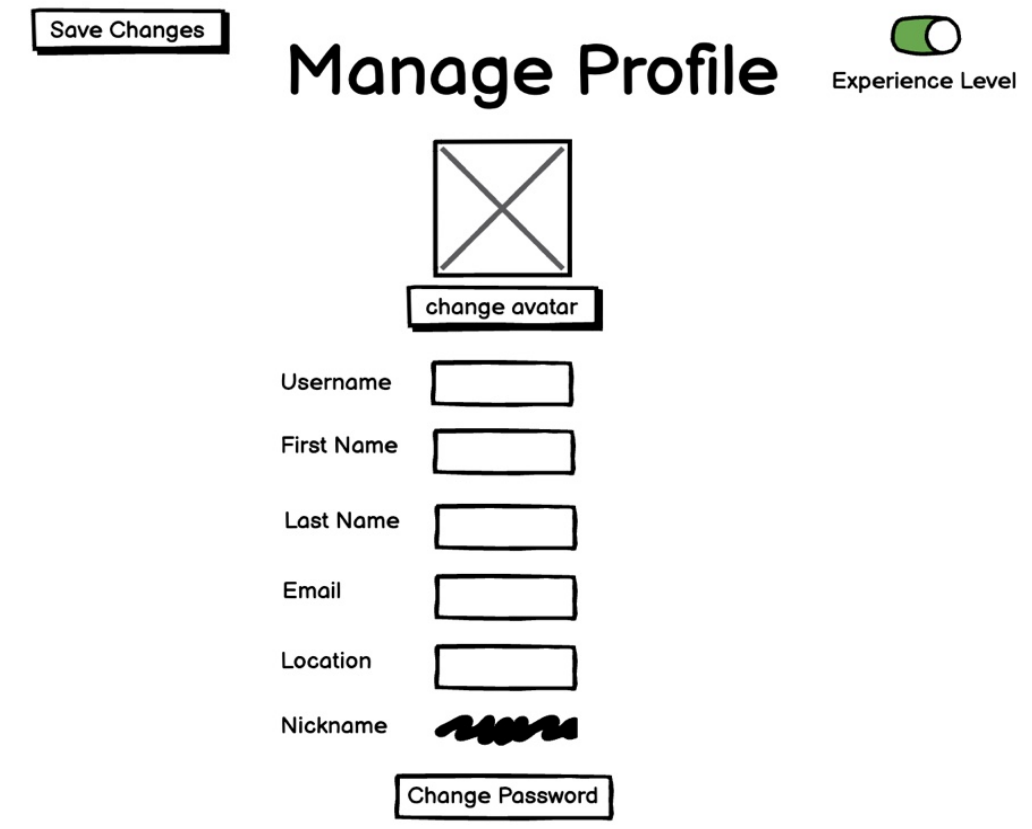


Figure 21: Manage Profile Wireframe

3.4.1.5 Hand Review

The hand review page was created to enhance user engagement and promote strategic learning for the user. It will offer users the opportunity to evaluate their gameplay and understand the AI's moves. This reflective practice will significantly improve the player's performance, which highlights that the platform isn't just for playing poker but also a tool for learning and improvement.

It will allow the user to view all the hands from the previously played game. The list of hands will be displayed in a container on the right side, the players cards and outcome of the hand will be used to identify each hand.

Once a hand is selected to view, the player's, AI's and community cards will be shown face up, along with all the actions taken by the user and the AI.

Additionally, the user can get a closer look at the decision making process of the AI. The "Why?" button provides insight into the AI's decision-making, revealing the logic behind each move, which will include probability distribution of the AI's choices, hand strength evaluation, and the action taken.

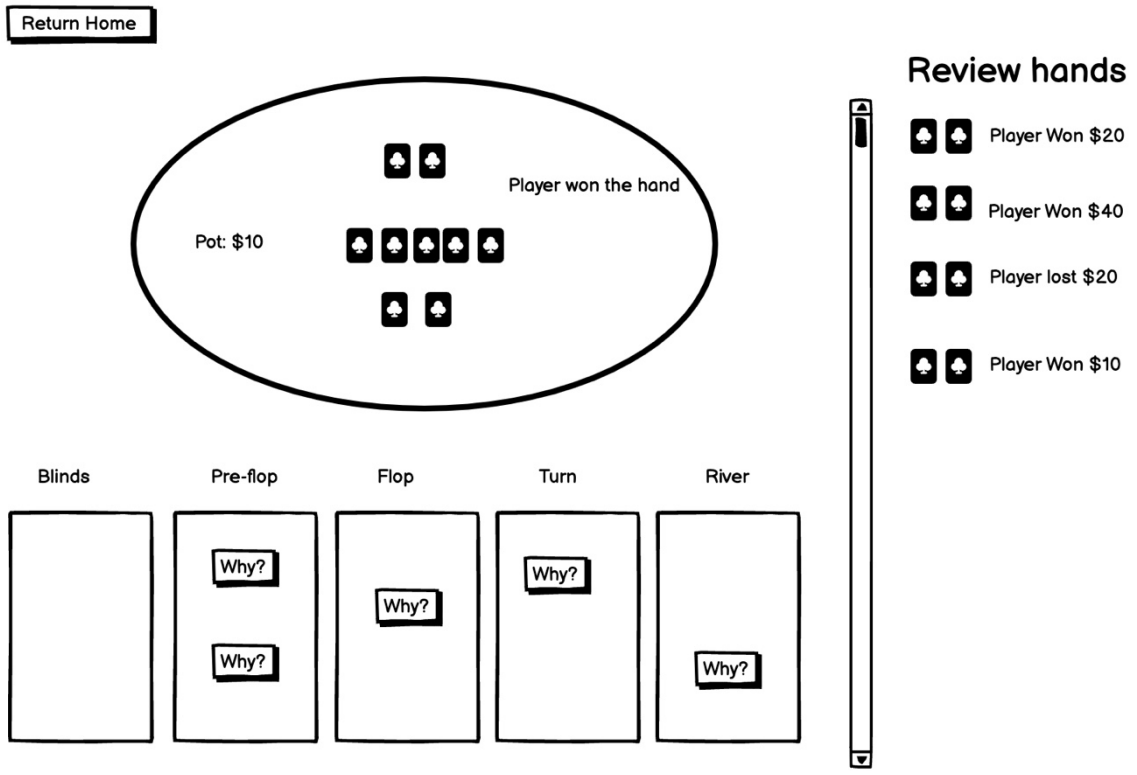


Figure 22: Hand Review Wireframe

3.4.2 Back-End

3.4.2.1 Database ERD

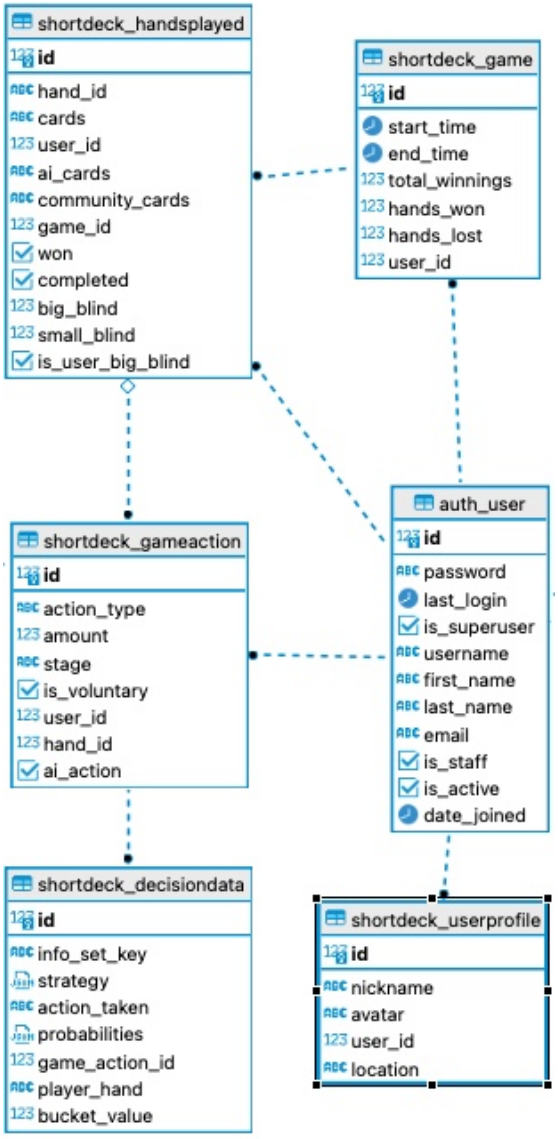


Figure 23: Database ERD

```python
#User profile mdoel stores the user prfile informatioon in the database, one-one relationship with user model.
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    nickname = models.CharField(max_length=100, blank=True)
    avatar = models.CharField(max_length=100, choices=AVATAR_CHOICES, default='avatar1.webp')
    location = models.CharField(max_length=100, blank=True)

    #method to get the avatar image url
    def get_avatar_url(self):
        return f'/static/images/avatars/{self.avatar}'

    # method ot get the experience of the player
    def get_experience_level(self):
        hands_played_count = HandsPlayed.objects.filter(user=self.user).count()
        if hands_played_count < 100:
            return 'Novice'
        elif hands_played_count < 500:
            return 'Advanced Beginner'
        elif hands_played_count < 1500:
            return 'Intermediate'
        elif hands_played_count < 3000:
            return 'Advanced'
        else:
            return 'Expert'
    def get_experience_percentage(self):
        hands_played_count = HandsPlayed.objects.filter(user=self.user).count()
        max_hands_for_expert = 1000
        return min(hands_played_count / max_hands_for_expert * 100, 100)

# The game model tracks the general game information
class Game(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    start_time = models.DateTimeField(auto_now_add=True)
    end_time = models.DateTimeField(null=True, blank=True)
    total_winnings = models.IntegerField(default=0)
    hands_won = models.IntegerField(default=0)
    hands_lost = models.IntegerField(default=0)

# The handsplayed model tracks all the hand information which is linked ot a game
class HandsPlayed(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    game = models.ForeignKey(Game, on_delete=models.CASCADE, related_name='hands_played')
    hand_id = models.CharField(max_length=255)
    cards = models.CharField(max_length=255)
    game_actions = models.ManyToManyField('GameAction', related_name='hands_played')
    ai_cards = models.CharField(max_length=255)
    community_cards = models.CharField(max_length=1000)
    won = models.BooleanField(null=True)
    completed = models.BooleanField(default=False)
    small_blind = models.IntegerField(default=0)
    big_blind = models.IntegerField(default=0)
    is_user_big_blind = models.BooleanField(default=False)

# The game actions model tracks the actions taken within each hand
class GameAction(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='actions')
    action_type = models.CharField(max_length=255)
    amount = models.IntegerField()
    stage = models.CharField(max_length=255)
    is_voluntary = models.BooleanField(default=True)
    ai_action = models.BooleanField(default=False)
    hand = models.ForeignKey(HandsPlayed, on_delete=models.CASCADE, null=True, blank=True)

# The decsion data model tracks the AI's decison data that calcualted by the CFR alogrithm
class DecisionData(models.Model):
    game_action = models.ForeignKey(GameAction, on_delete=models.CASCADE, related_name='decision_data')
    info_set_key = models.TextField()
    strategy = models.JSONField()
    action_taken = models.CharField(max_length=10)
    probabilities = models.JSONField()
    bucket_value = models.IntegerField(null=True, default=None)
    player_hand = models.CharField(max_length=255, null=True)

# The hand strength model holds all the hand combinations used ot mkae the hand strength grid
class HandStrength(models.Model):
    cards = models.CharField(max_length=255, unique=True)
    bucket_value = models.IntegerField()
    hand_strength = models.IntegerField()
```

The database for my application is designed to store the users personal data and all the required data that's collected when playing the poker game. This data is used to calculate the various user game statistics and analyse the previous hands players, as well as the actions taken by both the user and the AI. In order for that amount of data to be retrieved and organised in a way that's simple for the user to understand, the database is organised in an efficient manner.

Furthermore, rather than writing the SQL code for each query and creating the database manually, I have implemented ORM (Object-Relational Mapping) which serves as an intermediary layer that bridges the Python application with the relational database. It simplifies the operations by provided a simple Python interface for interacting with the database, by creating models and query the database directly with the Python. This has enabled me to put more focus into the overall functionality and using my limited time more efficiently.

The User table stores user related data and acts as the applications focal point for user related interactions.

The User Profile Table enables the user to personalise their identity with additional attributes, maintaining a direct one-to-one relationship with the User table.

The Game table captures the progress and the outcome of each game.

The HandsPlayed table records each hand within a game, while ensuring a detailed trace of the play, and is connected to the Game table with a many-to-one relationship. It allows each individual hand to be tracked within the game using the hand_id.

The GameAction table is vital to the overall functionality of the hand review page. It stores all the actions and the stages that the actions took place, such as pre-flop. This is important This table has a many-to-one relationship with the user and hands played tables.

The Decision Data table is an element within the database design that captures the decision-making process of the AI agent. It associates each action taken during a game with a set of data it uses to make its decisions. Additionally, it has a relationship to the GameAction table, which adds context to the decision within a specific hand.

3.4.2.2 Sequence Diagram: AI Decision Making in Poker Game

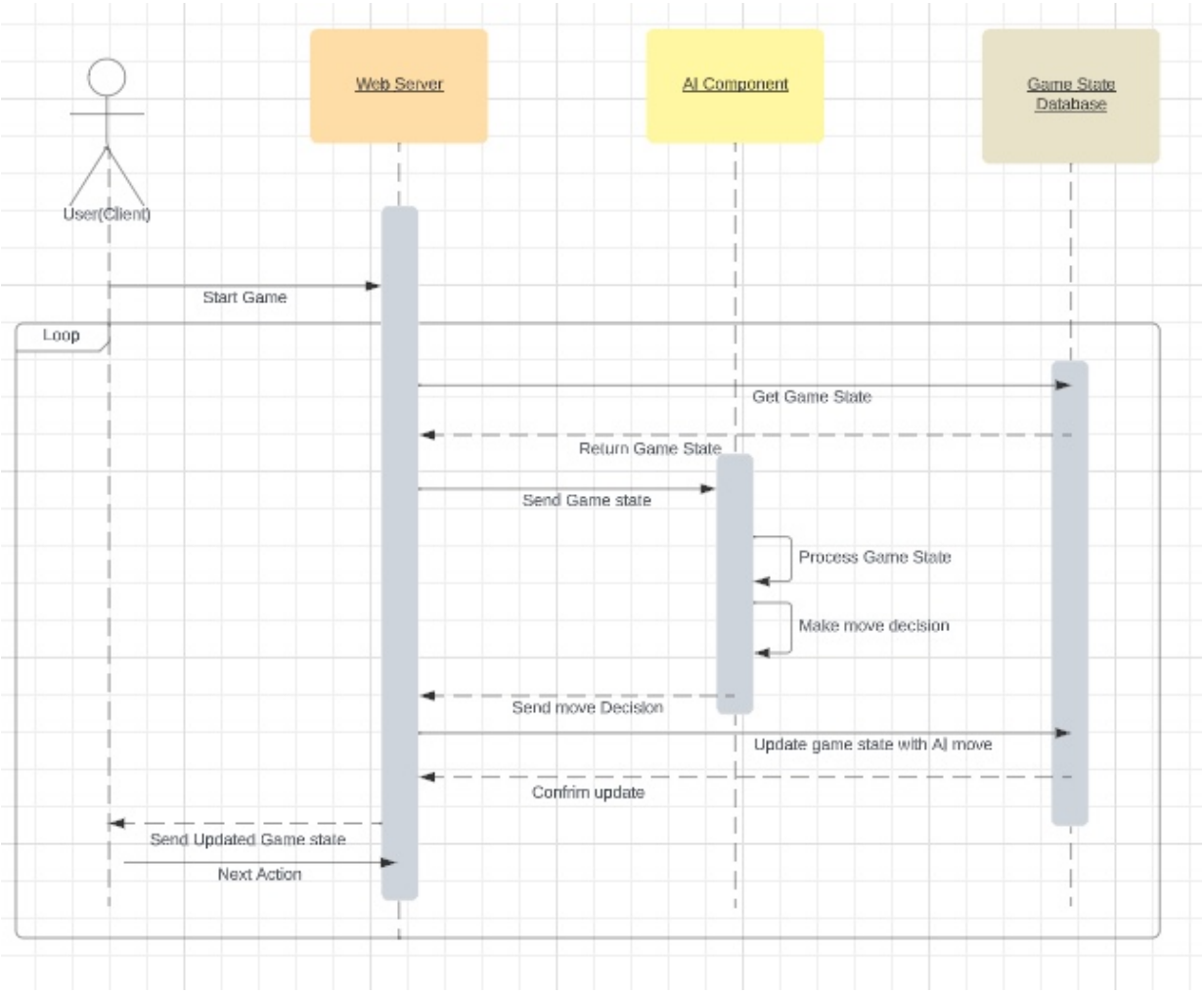


Figure 24: Decision Making Process Sequence Diagram

The AI decision making process is an integral component of the web application and overall project. The user initiates a round, this causes the game state to be retrieved by the web server and sent to the AI component. The AI component receives the game state and processes their decision, which is then sent back to the web server. The AI component decision making process is described in detail in 3.4.2.3 AI Component. The web server then updates the game state and notifies the user of the new state for their next move. This workflow is repeated throughout the game, it ensures an effortless integration of the AI component into the web application.

3.4.2.3 AI Component

The AI component of the Short Deck poker web application is the foundation of an interactive and challenging poker experience. The development is done using Counterfactual Regret Minimisation to train the AI, ensuring that the strategies are intelligent and practical for real-time gameplay.

To make the learning process computable, the game is simplified so that the agent can compute the decisions effectively without losing the strategic essence of the game. This simplification is done using two abstraction techniques.

The first technique that will be used is a method called action abstraction. Rather than considering every possible action the agent can take, I will limit the amount of actions the AI agent will have to consider. Instead of calculating the outcomes for every possible bet size, I will limit the agent to a standard raise amount of 20, if the player of AI is facing a raise, and they wish to reraise, the raise amount will be 40. The AI will also have the option of calling the raise amount, checking or folding. This vastly reduces the amount actions the AI has to consider and the complexity of the decision-making process. Each betting stage, such as pre-flop, flop, turn or river, will be limited to 4 consecutive raises by the player and the AI. If the betting limit has been reached they will be forced to call or fold.

Furthermore, the other abstraction technique I will be employing is information set abstraction such as bucketing similar hands together. Rather than treating each hand differently and computing a unique strategy for every hand, I will treat similar hands as the same, due to them having similar playing styles and pre-flop odds. For instance, 9-10 suited will be treated as same state as 9-8 suited, as both sets are cards are suited connecting cards and will have the same strategy pre-flop. This drastically reduces the numbers of scenarios the AI has to learn, but if there are too few buckets are created the strategy will be over simplified and easy to play against. The buckets will be created using a calculation based on their value and whether they are suited, this will be shown in detail in *4.2.2.1 Abstraction*

Once the abstraction is complete, the AI can begin training, it will engage in offline self-play, the agent will play repeatedly against itself to improve its strategy. It explores a wide range of game scenarios, making decisions and learning from their outcomes. First, the concept of information sets is crucial, an information set in the context of Short Deck is defined using a combination of abstraction methods. These sets form the decision points for the AI, where

each set makes up a combination of the player's private cards and the sequence of game actions.

The CFR algorithm is used to evaluate the effectiveness of the decision by calculating the regret values, which are a measure of what the AI could have won had it made a different decision. Based on the regret the AI adjusts its strategy accordingly which improves its ability to make decisions.

Once the AI is trained, the strategies are broken down into probabilities for each action across various game states and stored for quick retrieval during real time gameplay. I will use python's dill module to store the information in a file, it converts the object into a byte stream before being written to the file. When it's time to making a decision in a real time game, it reads the file and restores the AI decision making model into active memory. The AI can assess the current game state and compute the probabilities for best action based on the its current hand.

3.5 Conclusions

This section describes the various design methodologies considered and outlines the chosen methodology and the reasoning behind choosing this methodology. The methodology I have chosen to follow is an agile based methodology which follows a hybrid of Feature Driven Development and Scrum. It was chosen due to the project's needs for flexibility and iterative improvement, which is especially important for AI development.

Additionally, I illustrated the plan for the front end of the web application through the use of various designed UML diagrams and wireframes showcasing the ideal appearance of the user interface.

Furthermore, I described how the previously created prototype has helped in my design and will help the implementation of the final product. The set-up of the Django framework and testing and evaluation of the simplified AI agent using Counterfactual Regret Minimisation have been pivotal in providing me with the belief in the achievability of the final product.

4. Experiment / Software Development

4.1. Introduction

This section looks at the critical stags of developing the web application, by outlining the important aspects of the application. The process begins with the user's first interaction through registration, once the user is signed in. The development delves into various features of the web application and finally with the development of the AI agent. I will describe the reasoning behind certain aspect and the many intricate details throughout the application.

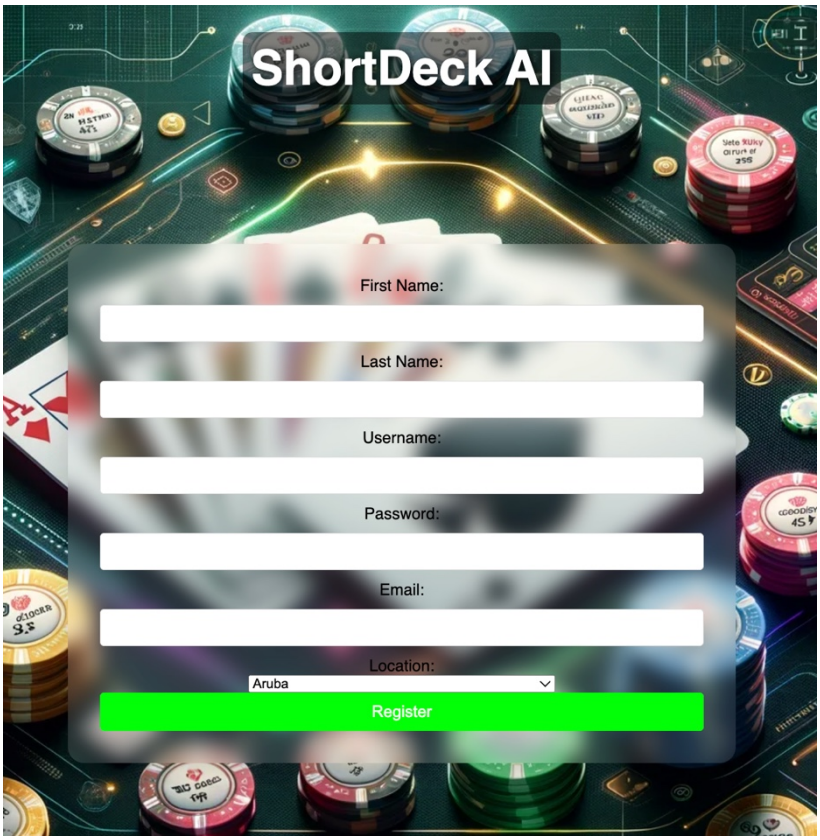
4.2. Software Development

4.2.1 Web Application

I will begin by outlining the functionality and development process of the web application. I will highlight the challenges, key aspects of each page and the role it plays in the overall flow of the application.

4.2.1.1 Registration / Login

The registration view is a process where users can create an account by providing the information which can be seen in *Figure 25: Application - registration.* The system checks if the username and email are unique to prevent duplicate entries. When successfully registered, the user will be redirected to the homepage, where they will be prompted to enter the login details of their new account. The user accounts are handled using Django's built in user authentication system. The User model from django.contrib.auth.models is the core of this system.



```python
#Allows the users to register
def register(request):
    countries = [{'code': country.alpha_2, 'name': country.name} for country in pycountry.countries]

    if request.method == 'POST':
        username = request.POST['username']
        password = make_password(request.POST['password'])
        email = request.POST['email']
        first_name = request.POST.get('first_name', '')
        last_name = request.POST.get('last_name', '')
        location = request.POST.get('location', '')

        if User.objects.filter(username=username).exists():
            messages.error(request, 'Username already exists.')
            return render(request, 'register.html', {'countries': countries})
        elif User.objects.filter(email=email).exists():
            messages.error(request, 'Email already registered.')
            return render(request, 'register.html', {'countries': countries})
        else:
            user = User.objects.create_user(username=username, password=password, email=email, first_name=first_name, last_name=last_name)
            UserProfile.objects.create(user=user, location=location)
            messages.success(request, 'Registration successful.')
            return redirect('login')
    else:
        # This return is for handling GET requests or other non-POST methods
        return render(request, 'register.html', {'countries': countries})
```

Figure 25: Application - registration

The user_login function is an interface for existing users to enter their username and password, and the system will validate the information against the stored data. If the user is successful they will gain access to the application and to their historical playing data displayed on the game dashboard. If incorrect details are entered, an error message will be displayed on the screen and the user will have to try again.

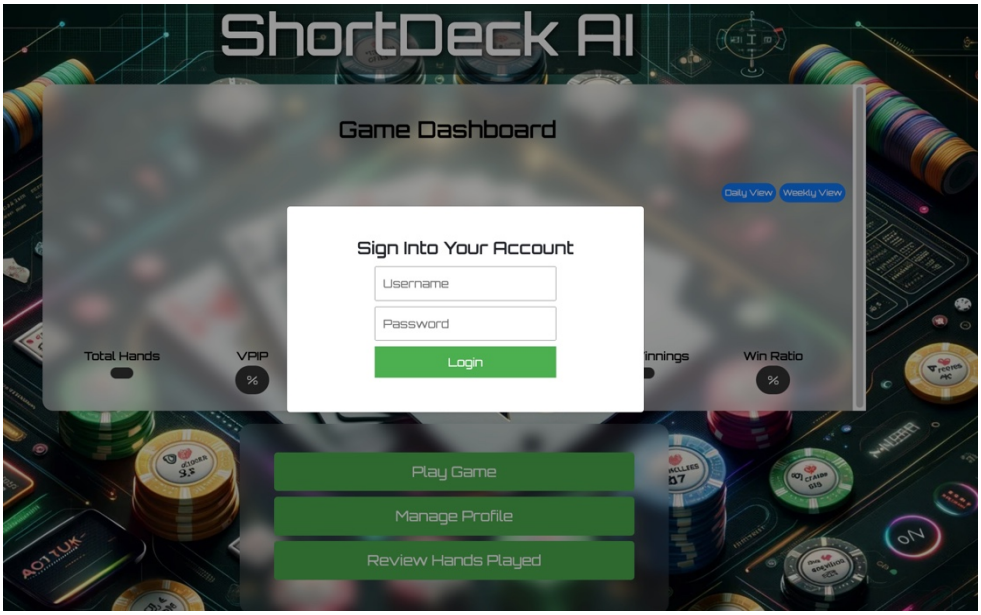


Figure 26: Application - user login

```python
#Methos handles the user login, checks if the user is valid and directs to homepage
def user_login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
        else:
            messages.error(request, 'Invalid username or password. Please try again.')
            return redirect('home')
    else:
        return redirect('home')
```

4.2.1.2 Homepage / Game Dashboard

The homepage and game dashboard serve as the primary interface for user interaction after they've logged in. Once authenticated, which is confirmed by checking request.user.is_authenticated, the user is presented with a dashboard that is personalised and informative. It is designed not just as a navigation hub but also as a statistical overview of the user's playing history.

The statistical information displayed within the game dashboard provides a visual representation of the users performance against the AI agent. All the data for this dashboard is stored in the 'HandsPlayed' model and the visualisation were created using chart.js.

The statistical information begins with a count of the hands played which gives an immediate insight into the amount the user has played. Additionally, key performance metrics such as Voluntarily Put Money In Pot (VPIP), Pre-Flop Raise (PFR), and Aggression Factor are calculated and displayed to help players understand and improve their playing style.

When the metrics on the game dashboard are hovered over, a pop up will provide context for the selected metric, which is essential for less experienced players that don't know the relevance of the displayed information.

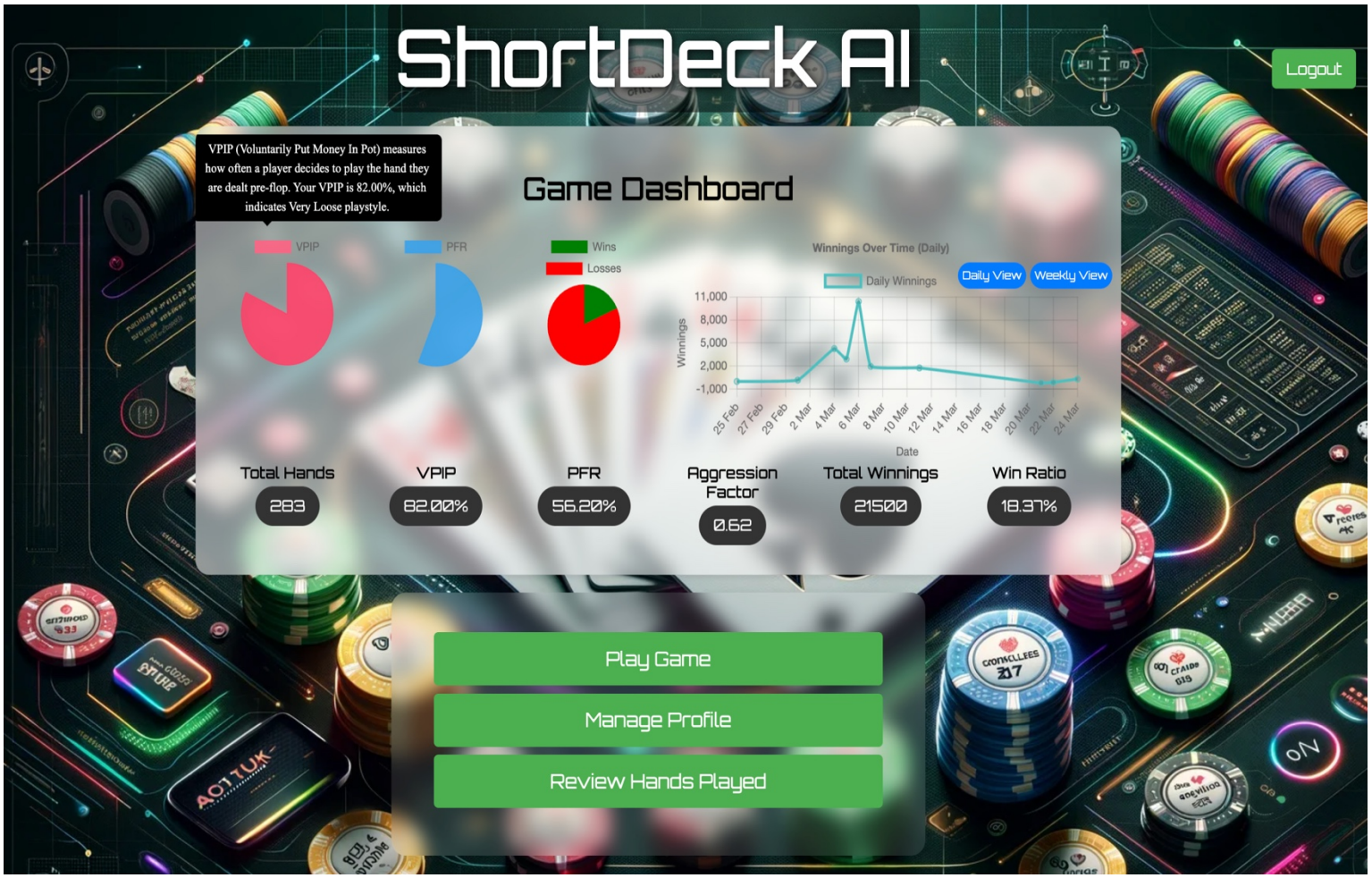


Figure 27: Application homepage / Game Dashboard

```python
def calculate_vpip(user):
    # Retrieve all HandsPlayed instances for the user
    hands_played_count = HandsPlayed.objects.filter(user=user).count()
    hands_played_ids = HandsPlayed.objects.filter(user=user).values_list('id', flat=True)

    # Count the number of unique hands where the user has voluntarily put money into the pot pre-flop
    voluntary_actions = GameAction.objects.filter(
        hand_id__in=hands_played_ids,
        user=user,
        is_voluntary=True,
        stage='Pre-flop'
    ).exclude(action_type__in=['post_blind', 'Check', 'Fold']).values_list('hand_id', flat=True).distinct().count()

    # Calculate VPIP
    vpip = (voluntary_actions / hands_played_count) * 100 if hands_played_count > 0 else 0
    vpip_2dp = round(vpip, 1)
    return vpip_2dp
```

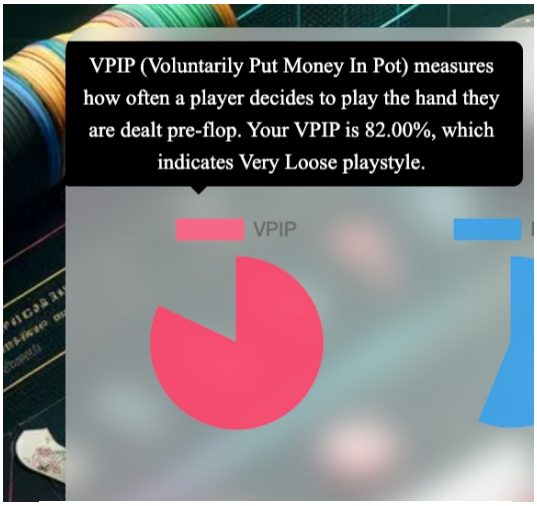


Figure 28: VPIP Pie Chart w/ Hover Information

VPIP records how actively a player participates in hands pre-flop. It represents the percentage of hands in which a player makes a voluntary contribution to the pot pre-flop, it doesn't include posting blinds.

It's calculated by the total number of voluntary actions divided by the total number of hands played.

The VPIP can be used to show how a player is playing. For example, a high VPIP would suggest a loose playstyle, where a player will play more hands and enter pots more often. Alternatively, a low VPIP would suggest a tight approach, where a player is more selective about the hands they play.

```python
def calculate_pfr(user):
    # Retrieve the count of all hands played by the user
    hands_played_count = HandsPlayed.objects.filter(user=user).count()
    hands_played_ids = HandsPlayed.objects.filter(user=user).values_list('id', flat=True)
    # Count the number of unique hands where the user has raised pre-flop
    pre_flop_raises = GameAction.objects.filter(
        hand_id__in=hands_played_ids,
        user=user,
        action_type='Raise',
        stage='Pre-flop'
    ).values_list('hand_id', flat=True).distinct().count()

    # Calculate PFR
    pfr = (pre_flop_raises / hands_played_count) * 100 if hands_played_count > 0 else 0
    pfr_2dp = round(pfr,1)
    return pfr_2dp
```

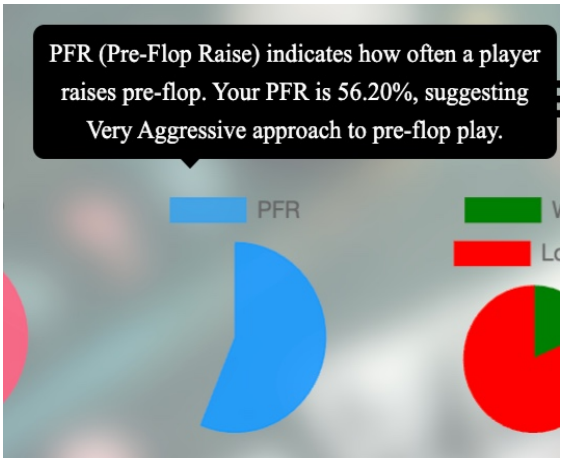


Figure 29: PFR Pie Chart w/ Hover Information

Pre-flop Raise tracks the frequency with which a player raises before the flop. This metric is a direct indicator of a player's aggression and willingness to take control early in the hand.

A higher PFR typically indicates a more aggressive player who enters the pot with a raise rather than a call or check.

It's calculated by dividing the total number of pre flop raises by the total number of hands played before the flop.

The code snippets below illustrate how the pie charts were created using the Chart.js library. The createPieChart function dynamically creates the three charts based on the various attributes that are passed into the function. The function initialises the new chart with the dataset variable, which includes the chartElementId, labels, data and other chart specific information, this is the information that creates each specific pie chart.

```
// Win/Loss Pie Chart
createPieChart('winRatioChart', ['Wins', 'Losses'], [winRatioData, 100 - winRatioData], ['green','red'], 'Win vs Loss Percentage');

// VPIP Pie Chart - showing only VPIP
createPieChart('vpipPieChart', ['VPIP'], [vpipData, 100 - vpipData], ['rgba(255, 99, 132, 0.9)', 'rgba(0, 0, 0, 0)'], 'VPIP Percentage');

// PFR Pie Chart - showing only PFR
createPieChart('pfrPieChart', ['PFR'], [pfrData, 100 - pfrData], ['rgba(54, 162, 235, 0.9)', 'rgba(0, 0, 0, 0)'], 'PFR Percentage');
```

```
// fucntion for the pie charts, created using the data for each chart once create pie chart is called
function createPieChart(chartElementId, labels, data, backgroundColors, chartLabel, includeOthers = true) {
    var ctx = document.getElementById(chartElementId).getContext('2d');

    var dataSet = {
        label: chartLabel,
        data: includeOthers ? data : [data[0]],
        backgroundColor: includeOthers ? backgroundColors : [backgroundColors[0]],
        borderColor: includeOthers ? backgroundColors.map(color => color.replace('0.2', '1')) : [backgroundColors[0].replace('0.2', '1')],
        borderWidth: 1
    };

    new Chart(ctx, {
        type: 'pie',
        data: {
            // Use only the first label if includeOthers is false
            labels: includeOthers ? labels : [labels[0]],
            datasets: [dataSet]
        },
        options: {
            responsive: true,
            plugins: {
                legend: {
                    display: includeOthers,
                    position: 'top',
                },
                title: {
                    display: false,
                },
                tooltip: {
                    enabled: false
                }
            },
            animation: {
                animateScale: true,
                animateRotate: true
            }
        }
    });
}
```

Aggression Factor (AF) compares the number of aggressive actions a player takes (raise) to the number of passive actions (check or call) for the players all round play not just pre-flop.
A higher AF indicates that a player is more aggressive, often raising instead of calling. Alternatively, a lower AF suggests that the player is more passive and prefers calling rather than take the initiative in betting.
An AF less than 1 indicates that the player is very passive, 1-2 indicates a standard aggression factor and greater than 2.5 would be considered aggressive.

```
def calculate_aggression_factor(user):
    # Retrieve all HandsPlayed instances for the user
    hands_played_ids = HandsPlayed.objects.filter(user=user).values_list('id', flat=True)

    # Initialize counts
    aggressive_actions_count = 0
    passive_actions_count = 0

    # Iterate through each unique hand played by the user
    for hand_id in hands_played_ids:
        aggressive_actions = GameAction.objects.filter(
            hand_id=hand_id, user=user, action_type='Raise').count()
        aggressive_actions_count += aggressive_actions

        passive_actions = GameAction.objects.filter(
            hand_id=hand_id, user=user, action_type__in=['Call', 'Check']).count()
        passive_actions_count += passive_actions

    # Calculate Aggression Factor
    aggression_factor = aggressive_actions_count / passive_actions_count if passive_actions_count > 0 else float('inf')
    aggression_factor_2dp = round(aggression_factor, 2)
    return aggression_factor_2dp
```

The Winnings Over time line graph provides a visual representation of the users performance. It allows the user to adjust the view, to display their daily winnings or weekly winnings using the buttons above the graph.

Figure 30: Winnings over time Line graph

The functionality of this graph was created using chart.js, the code snippet below shows the function used to create the time trend graph. The view type is set to daily by default, but by clicking the button above the graph they can toggle between the daily and weekly view of the graph.

```
// fucntion of rthe time trend line graph
function updateTimeTrendView(viewType) {
    var ctxTimeTrend = document.getElementById('timeTrendChart').getContext('2d');

    // befreo creating a new chart, the old must be deleted.
    if (chart) {
        chart.destroy();
    }

    // Using chart.js to create it, it can toggle between the viewtype which is daily and weekly data, defualtis set to daily
    chart = new Chart(ctxTimeTrend, {
        type: 'line',
        data: viewType === 'daily' ? winningsData.daily : winningsData.weekly,
        options: {
            scales: {
                x: {
                    type: 'time',
                    time: {
                        parser: 'yyyy-MM-dd',
                        tooltipFormat: 'DD MMM, yyyy',
                        unit: viewType === 'daily' ? 'day' : 'week'
                    },
                    title: {
                        display: true,
                        text: 'Date'
                    }
                },
                y: {
                    beginAtZero: false,
                    title: {
                        display: true,
                        text: 'Winnings'
                    },
                    ticks: {
                        stepSize: 1000,
                        callback: function(value) {
                            return value.toLocaleString();
                        }
                    }
                }
            },
            responsive: true,
            plugins: {
                legend: {
                    display: true,
                    position: 'top'
                },
                title: {
                    display: true,
                    text: 'Winnings Over Time (' + (viewType === 'daily' ? 'Daily' : 'Weekly') + ')'
                }
            }
        }
    });

}
```

```
// daily is called by defualt
updateTimeTrendView('daily')

// the button for the daily view, if clicked it will hsow the daily data
document.getElementById('dailyView').addEventListener('click', function() {
    updateTimeTrendView('daily');
});

// the button for the weekly view, if clicked it will hsow the weekly data
document.getElementById('weeklyView').addEventListener('click', function() {
    updateTimeTrendView('weekly');
});
```

4.2.1.3 Manage Profile

The manage profile section focuses on the user managing and customising their profile. The user will have the ability to view all their stored personal data, change the password protecting their account as well as change their avatar and nickname which is displayed in the poker game.

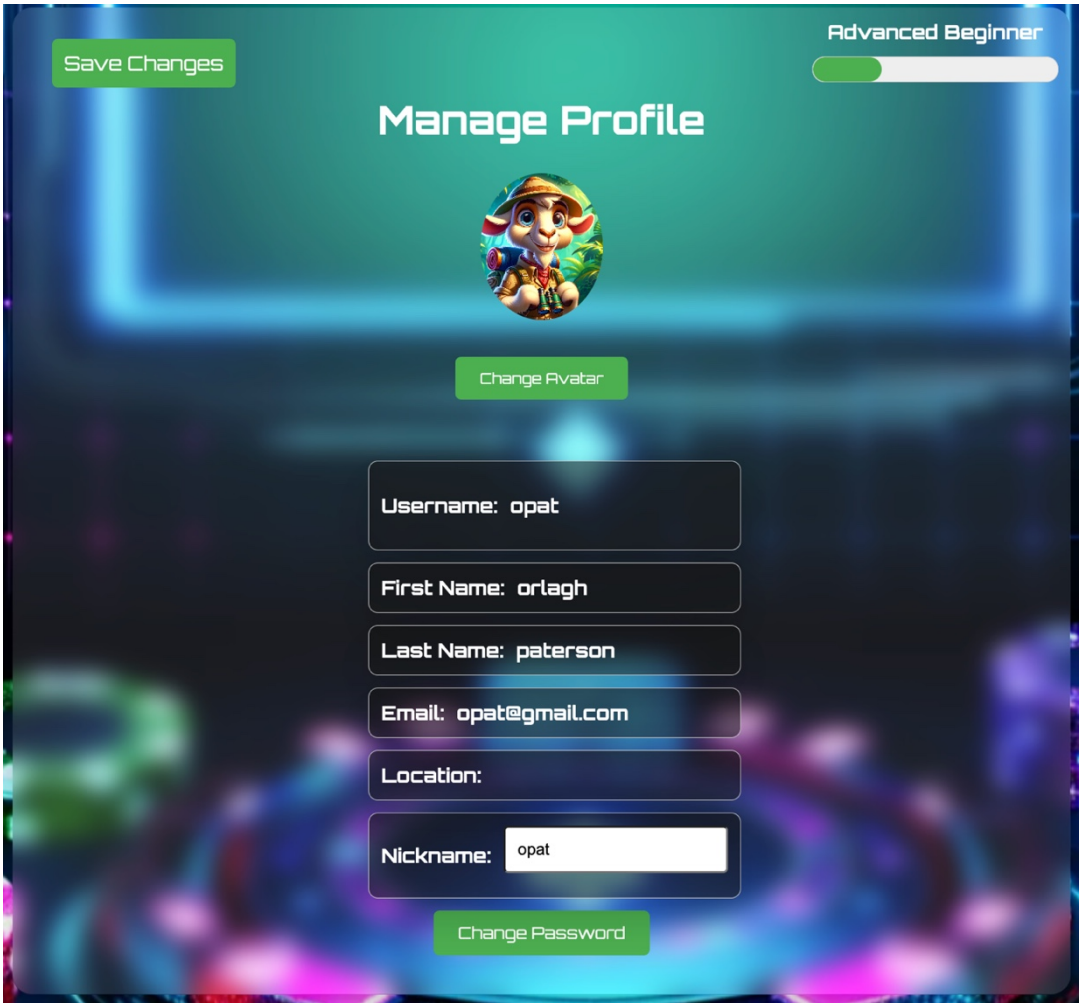


Figure 31: Application Manage Profile

```python
# Manage profile method
@login_required
def manage_profile(request):
    user_profile = UserProfile.objects.get(user=request.user)
    if request.method == 'POST':
        profile_form = UserProfileForm(request.POST, request.FILES, instance=user_profile)

        if profile_form.is_valid():
            profile_form.save()
            # Redirect to homepage if successful
            return redirect('home')
        else:
            messages.error(request, 'Please try again')
    else:
        profile_form = UserProfileForm(instance=user_profile)

    context = {'profile_form': profile_form}
    return render(request, 'manage_profile.html', context)
```

Once the ‘Change Password’ button is selected they are directed to the change password view, which requires them to enter the current password and verify with their new selected password.

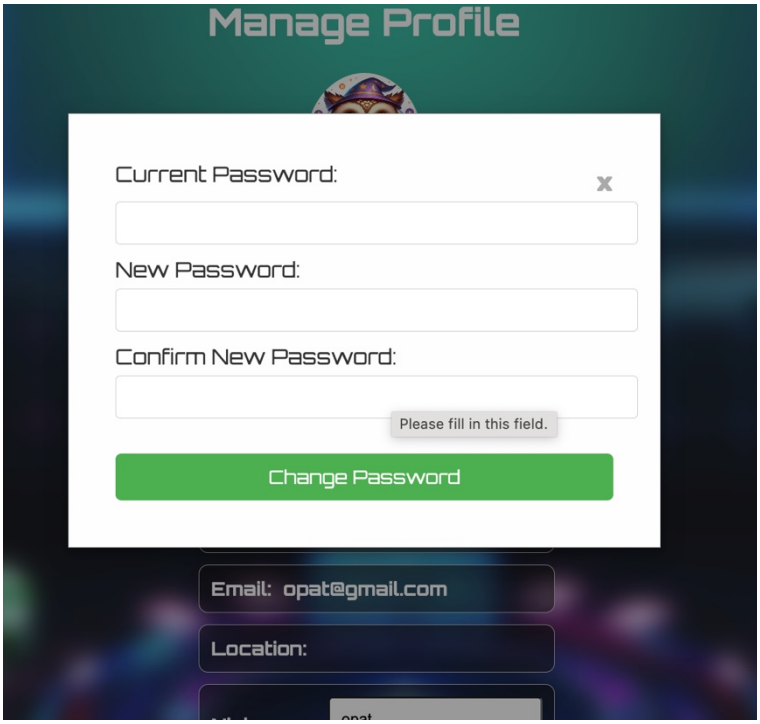


Figure 32: Application Change Password

The Manage profile view also shows the experience level of the user, as they play more hands, the experience bar will increase until they progress to the next level. This feature provides the user with a sense of achievement attached to their profile and rewards them with a new level the more they play. The ‘get_experience_level’ method shows the different levels depending on the number of hands played.

```
def get_experience_level(self):
    hands_played_count = HandsPlayed.objects.filter(user=self.user).count()
    if hands_played_count < 100:
        return 'Novice'
    elif hands_played_count < 500:
        return 'Advanced Beginner'
    elif hands_played_count < 1500:
        return 'Intermediate'
    elif hands_played_count < 3000:
        return 'Advanced'
    else:
        return 'Expert'
```

Furthermore, when the change avatar button is selected, the user is redirected to a new view which displays all the possible avatars to choose from to represent their profile. These avatars were created using Dall-E (35), which is a free image generator, this enabled me to create 10 images in the exact same size and style that are completely free to use and display. The images are saved to an avatars folder and saved under avatar choices. I’ve decided to use the .webp image extension rather than .png or .jpg as they provide a 30% reduction in file size and it led to an increase in the speed page load which will be explained in section

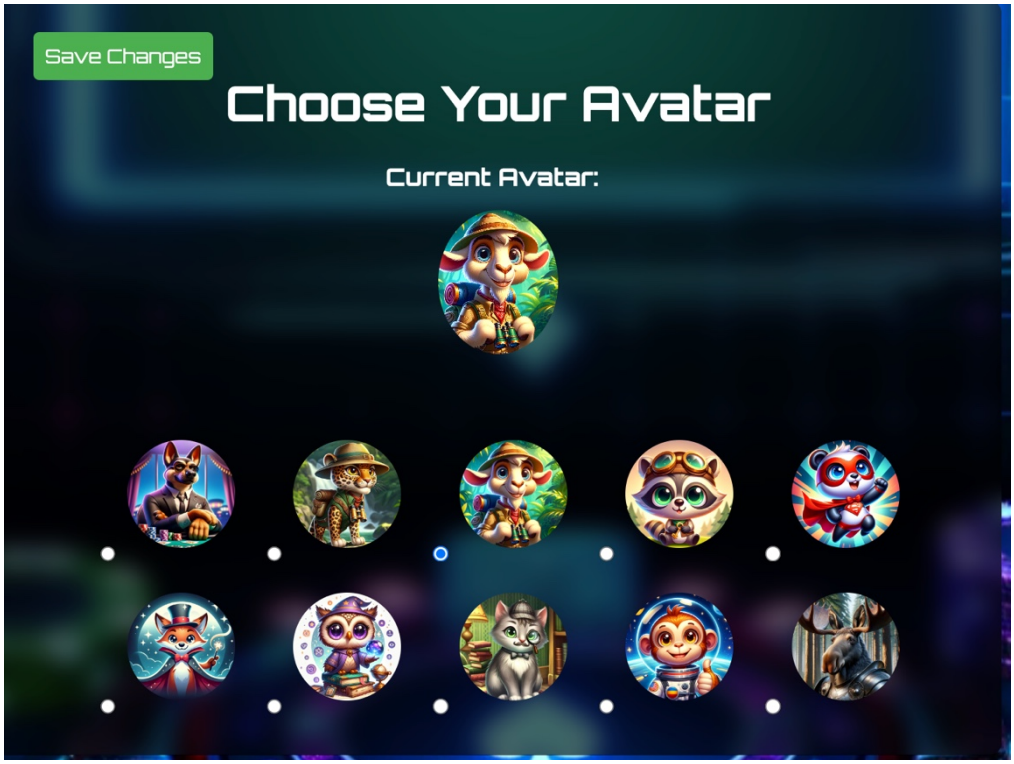


Figure 33: Application Change Avatar

Additionally, when the user is content with their selected avatar and selected to save the changes, they will be reverted to the manage profile view, which can be seen in the code. The save changes button also saves the changes to the database, the user will now see their new saved avatar in the manage profile view.

```python
@login_required
def change_avatar(request):
    user_profile = UserProfile.objects.get(user=request.user)

    if request.method == 'POST':
        form = UserProfileForm(request.POST, request.FILES)
        if form.is_valid():
            new_avatar = form.cleaned_data['avatar']
            user_profile.avatar = new_avatar
            user_profile.save()
            return redirect('manage_profile')
    else:
        form = UserProfileForm(initial={'avatar': user_profile.avatar})

    context = {
        'form': form,
        'userProfile': user_profile,
    }
    return render(request, 'change_avatar.html', context)
```

```python
# Define your avatar choices
AVATAR_CHOICES = (
    ('avatar1.webp', 'Dog'),
    ('avatar2.webp', 'cheetah'),
    ('avatar3.webp', 'Goat'),
    ('avatar4.webp', 'raccoon'),
    ('avatar5.webp', 'panda'),
    ('avatar6.webp', 'fox'),
    ('avatar7.webp', 'owl'),
    ('avatar8.webp', 'kitten'),
    ('avatar9.webp', 'monkey'),
    ('avatar10.webp', 'moose'),
)
#User profile mdoel stores the user prfile informatioon in the database, one-one relationship with user model.
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    nickname = models.CharField(max_length=100, blank=True)
    avatar = models.CharField(max_length=100, choices=AVATAR_CHOICES, default='avatar1.webp')
    location = models.CharField(max_length=100, blank=True)
    #method to get the image url
    def get_avatar_url(self):
        return f'/static/images/avatars/{self.avatar}'
```

4.2.1.4 Hand Review

This section explains the development process to the create one of the most complex aspect of my application, the hand review page. This page displays all the hands played in the players previous game, the hands can be viewed on the right side of the screen, if the user wishes to see more information about the hand, they can select ‘view hand’.
The ‘view hand’ button illustrates the cards of each player, the community cards, the pot, the hand outcome and all the actions taken by the AI and player.
Storing the hand data required a careful design of the database to ensure that all necessary details of a poker hand were recorded accurately. This included the actions taken by players and the decision-making process of the AI.
The hand review page had to be mapped closely to the poker game logic to ensure that the displayed hand history correctly matched the actual gameplay.

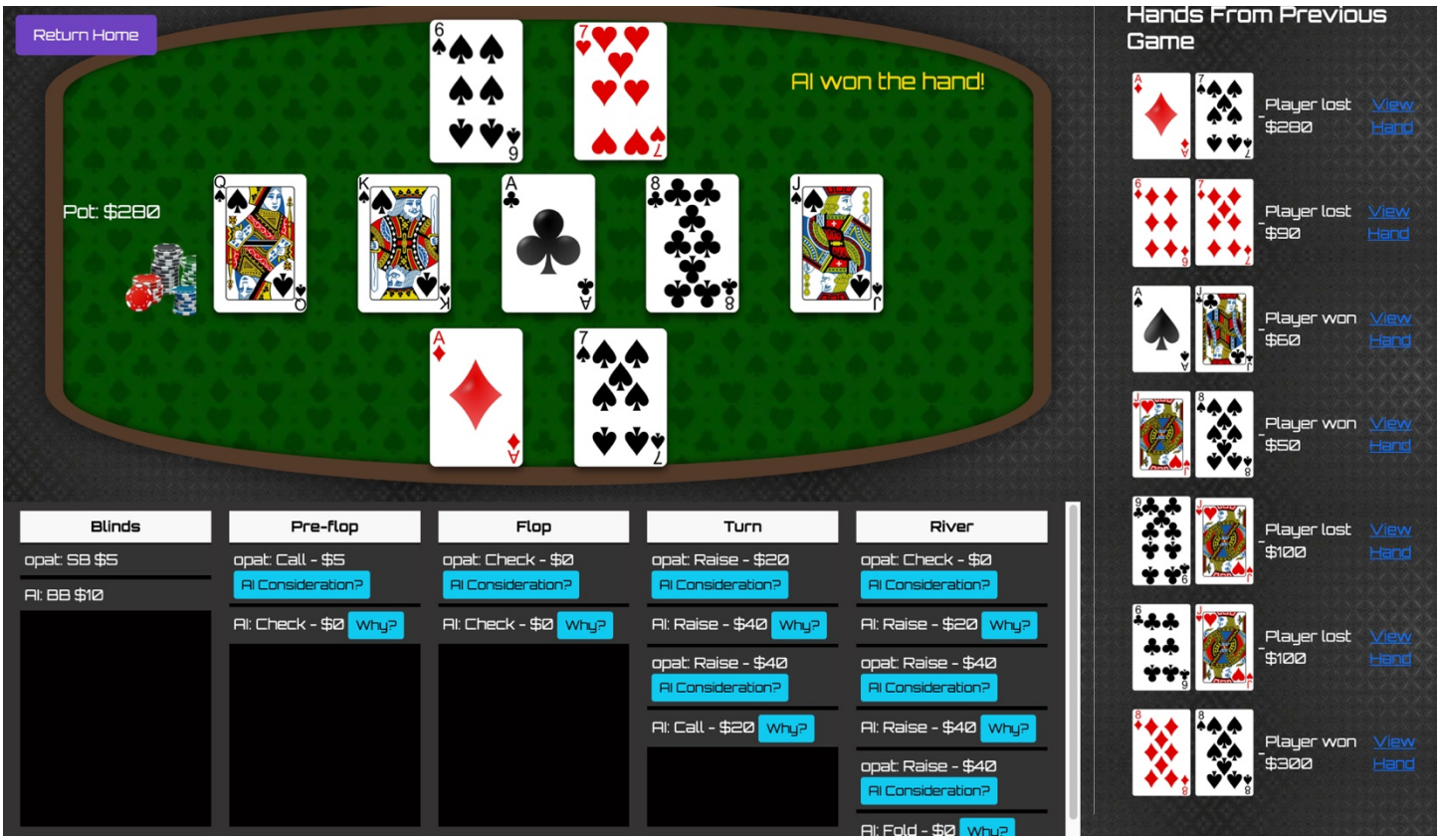


Figure 34: Application Hand Review Page

The hand review page improves the user experience by providing the players with insights into their past games and the AI's logic. The users can consider the decisions they’ve made during the previous hands. It starts by retrieving the last game and all the hands related to that game, firstly it checks if there were any hands played in the previous game, if so, it begins by retrieving each hand and displaying them in the right sidebar. It uses the image of the players hand along with whether they won or lost the hand and the size of the pot. Once a hand is selected, all the card data is deserialised and displayed, as well as all the actions broken down by the stage the actions took place. The code snippet below illustrates the implementation of the hand review page.

```python
# thia is the method for the hand reivew page which enables the user to view all their hands from the previous game, its a menu option.
@login_required
def hand_review_page(request, hand_id=None):
    # retrieve the last game
    last_game = Game.objects.filter(user=request.user).order_by('start_time').last()
    # get all the hands from the last game from the handsplayed model
    hands_query = HandsPlayed.objects.filter(game=last_game, user=request.user, completed=True).order_by('-id')
    hands = hands_query.all()
    has_hands = bool(hands)
    context = {
        "hands": hands,
        "has_hands": has_hands,
    }
    # Check whether or not a hand was played inthe last game
    if has_hands:
        if hand_id:
            current_hand = get_object_or_404(hands_query, id=hand_id)
            request.session["hand_id"] = current_hand.id
            hand_id = current_hand.id
        else:
            current_hand = hands_query.first()
            hand_id = current_hand.id


        # Prepare the cards for the side cards so they can be displayed down the right hand side
        for hand in hands:
            hand.player_cards_urls = [
                converting.cardImageURL(card) for card in converting.deserialise_cards(hand.cards)
            ] if hand.cards else []

            # get the total of the game aactions plus blinds
            game_actions_total = GameAction.objects.filter(hand=hand).aggregate(total=Sum('amount'))['total'] or 0
            pot = game_actions_total + hand.big_blind + hand.small_blind

            if hand.won:
                # Player won: Show positive amount as win
                hand.result_amount = f"Player won ${abs(pot)}"
            else:
                # Player lost or hand tied: Show positive amount as loss
                hand.result_amount = f"Player lost ${abs(pot)}"

        context["hands"] = hands
        # get the total of the game aactions plus blinds
        game_actions_total = GameAction.objects.filter(hand=current_hand).aggregate(total=Sum('amount'))['total'] or 0
        pot = game_actions_total + current_hand.big_blind + current_hand.small_blind
        context["pot"] = pot

        # deserialise all the cards data
        player_cards = converting.deserialise_cards(current_hand.cards) if current_hand.cards else []
        ai_cards = converting.deserialise_cards(current_hand.ai_cards) if current_hand.ai_cards else []
        community_cards = converting.deserialise_cards(current_hand.community_cards) if current_hand.community_cards else []
```

```python
#displays all the cards required for the view
for i in range(2):
    context["playerCard"+str(i+1)] = converting.cardImageURL(player_cards[i])

for k in range(5):
    if k < len(community_cards):
        context["tableCard"+str(k+1)] = converting.cardImageURL(community_cards[k])
    else:
        context["tableCard"+str(k+1)] = converting.back_card_URL()

for i in range(2):
        context["AI_cards"+str(i+1)] = converting.cardImageURL(ai_cards[i])

# Organise the actions by stage
actions_by_stage = defaultdict(list)
for action in GameAction.objects.filter(hand=current_hand).order_by('id'):
    actions_by_stage[action.stage].append(action)

context['actions_by_stage'] = dict(actions_by_stage)

# Determine who the winner was so it can be displayed in the view
if hand.won is not None:
    context["winner"] = "Player" if current_hand.won else "AI"
else:
    context["winner"] = "Tied"

# Determine who is the small blind and who is the big blind
if current_hand.is_user_big_blind:
    context['big_blind_user'] = request.user.username
    context['small_blind_user'] = 'AI'
else:
    context['small_blind_user'] = request.user.username
    context['big_blind_user'] = 'AI'

    context['small_blind_amount'] = current_hand.small_blind
    context['big_blind_amount'] = current_hand.big_blind

    request.session["hand_id"]=hand_id

turn render(request, 'hand_review_page.html', context)
```

The decision data pop-up reveals the underlying logic of the AI's decisions. Players can see the probability distribution of the actions taken by the AI at different stages of the game. For instance, the pop-up shows how the AI decided to Check with a particular hand, this is supported by the calculated strategy of the AI, which will give the users a deeper understanding of the decision making process. Showing the AI's decision-making process, enables the players to improve their own strategies.

In summary, this tool not only enhances the gaming experience but also serves as an educational platform that explains complex AI strategies for players.

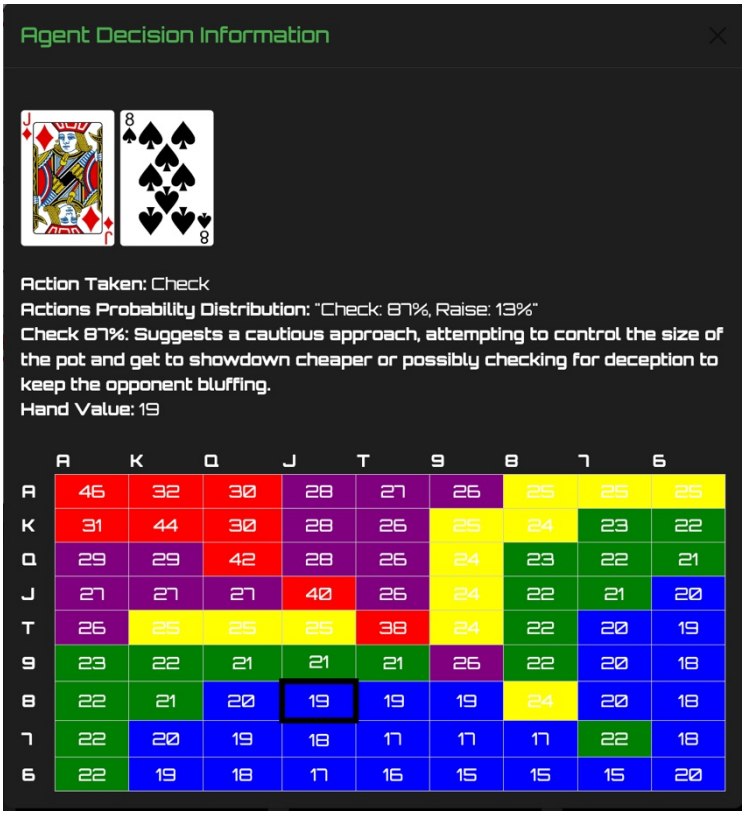


Figure 35: Application – Pre-Flop Hand Review Decision Data

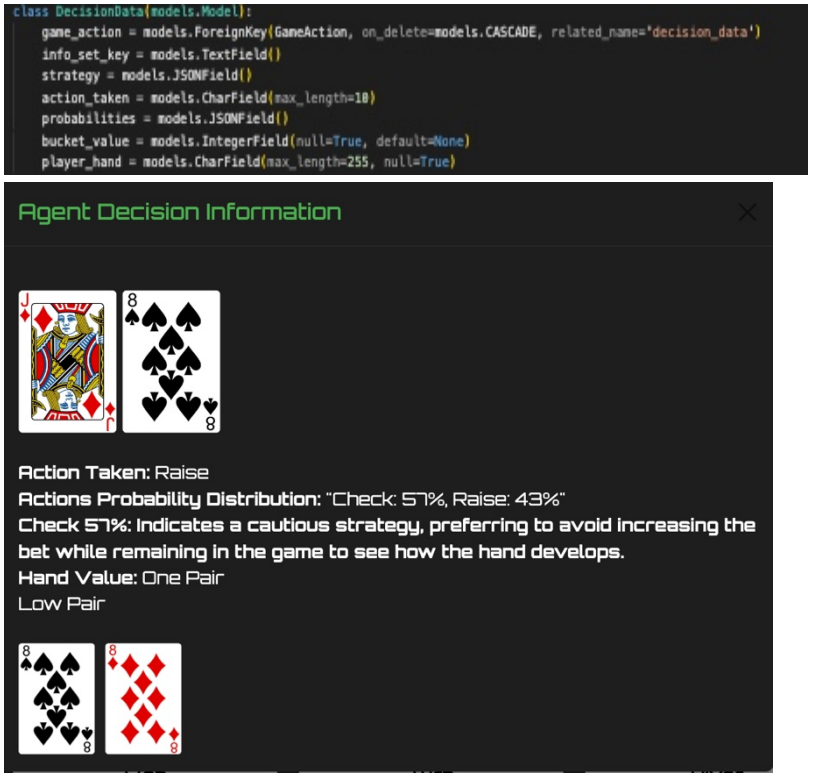


Figure 36: Application – Post-Flop Hand Review Decision Data

If the game is in the pre-flop stage the strength of the hand itself is assessed and this is displayed to the user in the hand strength grid, where each hand is assigned to a color-coded bucket representing different levels of strength.

The code snippet below illustrates how this feature was created, it starts with the retrieval of all the game action information related to the selected hand. Then there's a check to determine if the stage is pre-flop, if so the grid is created using the generate_hand_stregnths_grid function and the players hand is then highlighted in a black box, which can be seen in Figure 35: Application – Pre-Flop Hand Review Decision Data. If the stage isn't pre-flop, then evaluate the hand and determine what the strength of the hand is at the given stage. For example, If the current_stage is Turn, it only considers the first 4 community cards, combined with the AI's cards to get the strengths of the hand with regards to the community cards. Figure 36: Application – Post-Flop Hand Review Decision Data shows this in action where the strength of the hand at the river, which is considering all community cards and the AI's hand, is just a low pair. This illustration is the exact method the AI uses when considering the strength of their hand in the context of the game.

```python
# This method is where the decision data of each of the AI's actions will be displayed in the hand review view.
def decision_data_view(request, action_id):
    # get the hand its associated with
    hand_id = request.session.get("hand_id")
    decision_data = get_object_or_404(DecisionData.objects.select_related('game_action__hand'), game_action_id=action_id)
    hand = get_object_or_404(HandsPlayed, id=hand_id, user=request.user)
    game_action = get_object_or_404(GameAction, id=decision_data.game_action_id)
    current_stage = game_action.stage

    # Check if the current stage is the Pre-flop
    is_preflop = current_stage == 'Pre-flop'

    # Determine the actions which beloing ot the AI
    if game_action.ai_action:
        # For AI's action, use AI's cards
        cards_to_use = hand.ai_cards if hand.ai_cards else []

    # Deserialise the selected cards and generate image URLs
    cards = converting.deserialise_cards(cards_to_use)
    card_images = [converting.cardImageURL(card) for card in cards]
    community_cards = converting.deserialise_cards(hand.community_cards)

    # If its post-flop
    if not is_preflop:
        # find out the number of community cards to include based on the current stage
        if current_stage == 'Flop':
            community_cards = community_cards[:3]
        elif current_stage == 'Turn':
            community_cards = community_cards[:4]
        elif current_stage == 'River':
            community_cards = community_cards[:5]

        # Used to find the value of the hand post flop
        combined_cards = cards + community_cards
        hand_type, involved_cards, pair_category = evaluate_hand_for_decision_data(combined_cards)
        involved_cards_image = [converting.cardImageURL(card) for card in involved_cards]

    else:
        # Default values for pre-flop or other stages
        hand_type, involved_cards_images = None, []
```

```python
# Generate the grid for the hand strengths
grid = CFR.generate_hand_strengths_grid()

# Correctly determine positions for highlighting
player_positions = [rank_to_index[map_value_to_rank(card['value'])] for card in player_hand]
is_suited = player_hand[0]['suit'] == player_hand[1]['suit']

i, j = player_positions
if is_suited:
    grid[min(i, j)][max(i, j)]['highlight'] = True
else:
    grid[max(i, j)][min(i, j)]['highlight'] = True


# displays the grid pre flop
grid_html = ''
if is_preflop:
    grid_html = render_to_string('grid_template.html', {
        'grid_data': grid,
        'ranks': ranks,
        'row_headers': row_headers,
    })
player_hand_string = ' '.join([f"{card['value']}{card['suit']}" for card in player_hand])

data = {
    'info_set_key': decision_data.info_set_key,
    'strategy': decision_data.strategy,
    'action_taken': decision_data.action_taken,
    'probabilities': probabilities.strip('"'),
    'is_preflop': is_preflop,
    'card_images': card_images,
    'player_hand': player_hand_string,
    'explanatory_message': explanatory_message,
}

# only show if pre-flop
if is_preflop:
    data.update({
        'bucket_value': decision_data.bucket_value,
        'grid_html': grid_html,
        'color_legend': color_legend,
    })
# show if post flop
else:
    data.update({
        'hand_type': hand_type,
        'involved_cards_image': involved_cards_image,
        'pair_category': pair_category,
    })
return JsonResponse(data)
```

4.2.1.5 Play Game

Figure 37: Application - Play Poker

The poker game view is the core of the application, it enables the user to test their ability against the AI. This was certainly the most complex aspect of the application, it required many different elements to get the flow of the game working as desired. Additionally, I utilised JavaScript to create animations of standard poker movements, such as throwing chips into the pot when betting and throwing the cards into the centre when folding. It was also used to portray the AI robot thinking when the AI is making a decision and provide the user with constant feedback throughout the flow of the hand.
Additionally, the logic for the poker game itself, such as evaluating the winner of the game and getting the flow of the game working as desired was quite complicated.

Figure 38: Player Action Bubble pop-up

Figure 39: Application - Winner Pop-up

4.2.1.5.1 Cards

In the poker web application, the Django framework manages the session variables and handles the card images for the user interface. The card dealing mechanism begins with the dealdeck function, which deals the cards from the session deck to the player, AI agent, and community which are stored in a serialised format within the user's session.

```
def dealdeck(num, deck):
    # Selects random cards from the deck
    hand = random.sample(deck, num)
    # Removes the selected cards from the deck
    for card in hand:
        deck.remove(card)
    return hand
```

```
# Deal the players cards and the community cards and serialsie / save to the session
playerCards = cards.dealdeck(2, deck)
playerList[0].hand = playerCards
request.session["playerCards"] = converting.serialise_cards(playerCards)
request.session["deck"] = converting.serialise_cards(deck)
```

The 'cardImageUrl' function is used in this process, by changing the cards values into a string that links to the image file for each card. This naming convention is vital for linking each card object to its visual representation on the front end. The 'back_card_URL' provides the path to the image of a card back, which is used when the AI and community cards are to be displayed face down.

The session data, once deserialised using the 'deserialise_cards' function, is used to display either the front or back of the cards, depending on the game's progress. Community cards are revealed depending on what stage of the betting round has been reached, it reflects the real flow of a poker game. The opponent's cards, which are controlled by the AI agent and are not visible to the player until the end of the round, when the cards are finally shown to determine the winner.

```
# Used so the corresponding card can be assigned an image which is then displayed
def cardImageURL(card):
    # Extract suit and name directly from the Cards instance
    suit_name = card.suit.lower()

    # Convert value to corresponding name
    if card.value == 14:
        card_name = "A"
    elif card.value == 13:
        card_name = "K"
    elif card.value == 12:
        card_name = "Q"
    elif card.value == 11:
        card_name = "J"
    else:
        card_name = str(card.value)

    return f"/static/images/ShortDeckCards/{card_name}{suit_name}.webp"

# Used to show the back of the card when players not meant ot see them
def back_card_URL():
    return f"/static/images/ShortDeckCards/BackCard.webp"
```

```
# Used to turn the card objects into a JSON string so they can be stored for the session
def serialise_cards(cards):
    card_dicts = [card.to_json() for card in cards]
    # Convert list of card dictionaries to a JSON string
    return json.dumps(card_dicts)

# Used to turn the JSON string back into the card  object once retirved from the session
def deserialise_cards(serialised_cards_str):
    if isinstance(serialised_cards_str, str):
        card_dicts = json.loads(serialised_cards_str)
        return [Cards.from_json(card_dict) for card_dict in card_dicts]
    else:
        # Handle the case where the data is already in the expected format
        return [Cards.from_dict(card_dict) for card_dict in serialised_cards_str]
```

4.2.1.5.2. Actions

When a player is presented with their action options in the poker game, they interact with the frontend form by clicking one of the available action buttons. These form choices can be seen in the code snippet below on the left. The actions that are available to the player or AI are determined by the action history. For instance, if the last action was a raise by the AI (history[-1] == ‘Raise’), the player has the option to call, raise again or fold. The available actions depending on the history can be seen in the right hand code snippet below

For example, if the player clicks "Raise," the backend updates the player's chip and adds the chips to the pot. This triggers the game to move to the next action, which will be made by the AI agent. The player's session data keeps track of all these changes to ensure that when the page is refreshed or a new request is made, the game state is consistent with the player's last action.

Additionally, as the player or AI makes an action, the action is logged using the log_game_action function, this data is later retrieved for hand review and analyses.

```
<!-- action form -->
<div class="player-actions">
    <form id="actionForm" method="post">
        {% csrf_token %}
        {% if formChoices|length <= 0 %}
            <button type="submit" name="choice" value="Next Round" class="actionButton">Next Hand</button>
        {% endif %}
        {% if "Call" in formChoices %}
            <button type="submit" name="choice" value="Call" class="actionButton">Call</button>
        {% elif "Check" in formChoices %}
            <button type="submit" name="choice" value="Check" class="actionButton">Check</button>
        {% endif %}
        {% if "Fold" in formChoices %}
            <button type="submit" name="choice" value="Fold" class="actionButton">Fold</button>
        {% endif %}
        {% if "Raise" in formChoices %}
            <button type="submit" name="choice" value="Raise" class="actionButton">Raise</button>
        {% endif %}
    </form>
</div>
```

```
# Raise action by the user
if choice == "Raise":
    # The amount to raise on top of matching the current highest bet
    raise_amount = 20
    request.session.modified = True
    # Calculate the amount needed to match the opponent's bet, then add the raise amount
    difference = playerList[1].bet - playerList[0].bet
    playerList[0].bet += difference + raise_amount
    # Update the pot with the match amount plus the raise
    raise_total = difference + raise_amount
    pot += raise_total
    # save the pot to the session
    request.session["pot"] = pot
    request.session.modified = True

    balance -= raise_total
    chips -= raise_total
    request.session["balance"] = balance
    request.session["chips"] = chips

    stage = request.session['stage']

    # Log the game action to the database
    game_action = log_game_action(
        request,
        user=request.user,
        action_type='Raise',
        amount=raise_total,
        stage=stage,
        is_voluntary=True,
        ai_action=False
    )
    # The action that the user can choice from is determined by the history
    actions = []
    if endRound:
        actions = []
    elif history[len(history) - limit : len(history)] == ["Raise"]*limit:
        actions = ["Call","Fold"]
    #prevents index error
    elif len(history) == 0:
        actions = ["Call","Fold","Raise"]
    #necessary response actions, fold removed when check is possible
    elif history[-1] == "Raise":
        actions = ["Call","Fold","Raise"]

    elif history[-1] == "Check" or history[-1] == "Call" or history[-1]=="Round":
        actions = ["Check","Raise"]
    request.session["formChoices"] = actions
```

4.2.1.5.3 Hand Evaluation / Poker Logic

This section explains how the hands are evaluated and how the poker logic will work, it will also explain how the winner is determined.

The ‘calculate_hand_value’ function finds the best 5 card combination based on the combination of the community cards and the player’s cards. It loops through all the 21 possible 5 card combinations that can be made from the 7 cards and checks which is the best one using the ‘evaluate_hand’ and ‘getWinningHand’ functions.

The Texas hold’em implementation mentioned earlier in the literature review section (36), was used to help with this part of the project, I gained some insight into how to calculate the value of the hand and also determining the winning hand.

The hand of the player or AI , as well as the community cards are passed into the ‘calculate_hand_value’ function, it then loops through each hand from the 21 possible hands, determines which is the best and returns that hand.

```
# Evaluates the best hand from a player's cards and community cards.
@staticmethod
def calculate_hand_value(hand, community_cards):
    # Initialise a placeholder for the best hand found, Used to store hand ranking and tie-breaking information.
    best_hand = [0,0,0,0,0,0,0]
    # Generate all possible 5-card combinations from the 7 available cards (2 hole cards + 5 community cards) = 21 combinations
    for hand in combinations(hand+community_cards,5):
        # For each combination, evaluate the hand to get its ranking
        current_hand_ranking = PokerPlayer.evaluate_hand(list(hand))[0]

        # Prepare a list to compare the current combination's ranking against the best found so far
        compare = [best_hand, current_hand_ranking]

        # Determine which hand is better based on the game's hand ranking logic
        bestIndex = PokerGame.getWinningHands(compare)[0]

        # Update the best hand if the current hand is better
        best_hand = compare[bestIndex]

    return best_hand
```

In the ‘getWinningHands’ function the hands are compared to see which has the highest rank, if there is only 1 winner, then there has been a winner found, otherwise it’s still a draw. It is capable of handling draw and uses recursion to break draw if necessary, when the function is recursively called the compare_num parameter is incremented. The compare_num parameter iterates over the hand ranking to ensure that the best hand is found in a draw situation.

```
@staticmethod
def getWinningHands(rank_list, compare_num=0, to_compare=None):
    if to_compare is None:
        to_compare = list(range(len(rank_list)))

    # Initialise the highest rank and the list of potential winners.
    highest_rank = -1
    potential_winners = []

    # Iterate over each hand that is still in contention.
    for hand_index in to_compare[:]:
        hand_rank = rank_list[hand_index][compare_num]

        # Check if the current hand has the highest rank found so far.
        if hand_rank > highest_rank:
            # If a new highest rank is found, update the list of winners.
            highest_rank = hand_rank
            potential_winners = [hand_index]
        elif hand_rank == highest_rank:
            # If the current hand has a rank equal to the highest, add it to the list of winners.
            potential_winners.append(hand_index)

    # only include the potential winners.
    to_compare[:] = potential_winners

    # Determine if a set of winners has been found.
    if len(potential_winners) == 1 or highest_rank == 0 or compare_num == 5:
        return potential_winners
    else:
        # if there's still a tie call again
        return PokerGame.getWinningHands(rank_list, compare_num + 1, to_compare)
```

The evaluate_hand function is used to determine the strength of a hand, the rankings variable is an array which is used to store the hand rankings. The first element in the array is used to categorise the hand, for example, 7 would be a flush. The next elements in the array are used to further categorise the hand, in the case of a full house, the value of the three of a kind card is the next value in the rankings array and followed by the pair value. The pairs are found using the value_count variables which counts the number of occurrence of a card to determine if it's a pair or three of a kind.

In this variant of poker, Short Deck, there are a couple of unique rules that it follows. In Short Deck, an ace can act as the lowest, which would be a 5 or the highest card. It enables straights to be made from 'A-6-7-8-9' or '10-J-Q-K-A', the 'check_straight' method describes the steps which enables this scenario.

```python
@staticmethod
def evaluate_hand(hand):
    # Sort the hand by card value in descending order for easier evaluation
    hand = sorted(hand, key=lambda card: card.value, reverse=True)
    is_flush = all(card.suit == hand[0].suit for card in hand)

    # Check for straight, including Ace as low
    is_straight, straight_high_card = PokerPlayer.check_straight(hand)

    # Count occurrences of each card value
    value_counts = {card.value: sum(card.value == x.value for x in hand) for card in hand}
    value_count_pairs = sorted(value_counts.items(), key=lambda x: (-x[1], -x[0]))

    # Initialize rankings and msg
    rankings = [0, 0, 0, 0, 0, 0]
    msg = ""

    # Straight FLush
    if is_flush and is_straight:
        rankings[0] = 9
        msg = "Straight Flush"

    # Four of a kind
    elif value_count_pairs[0][1] == 4:
        rankings[0] = 8
        rankings[1] = value_count_pairs[0][0]
        msg = "Four of a Kind"

    # Flush
    elif is_flush:
        rankings[0] = 7
        # High card values for flush
        for i, card in enumerate(hand):
            rankings[i + 1] = card.value
        msg = "Flush"

    # Full House
    elif value_count_pairs[0][1] == 3 and value_count_pairs[1][1] == 2:
        rankings[0] = 6
        rankings[1] = value_count_pairs[0][0]  # Three of a kind value
        rankings[2] = value_count_pairs[1][0]  # Pair value
        msg = "Full House"

    # Striaght
    elif is_straight:
        rankings[0] = 5
        msg = "Straight"

    # Three of a kind
    elif value_count_pairs[0][1] == 3:
        rankings[0] = 4
        threeOkind = value_count_pairs[0][0]
        rankings[1] = threeOkind
        extraCard = [i for i in hand if i.value != threeOkind]
        rankings[2] = extraCard[-1].value
        rankings[3] = extraCard[-2].value
        msg = "Three of a Kind"

    # Two pair
    elif value_count_pairs[0][1] == 2 and value_count_pairs[1][1] == 2:
        rankings[0] = 3
        rankings[1] = value_count_pairs[0][0]  # Higher pair value
        rankings[2] = value_count_pairs[1][0]  # Lower pair value
        rankings[3] = [i for i in hand if not i.value in value_count_pairs][0].value
        msg = "Two Pair"

    # One pair
    elif value_count_pairs[0][1] == 2:
        rankings[0] = 2
        rankings[1] = value_count_pairs[0][0]
        extraCard = [i for i in hand if not i.value in value_count_pairs]
        rankings[2] = extraCard[-1].value
        rankings[3] = extraCard[-2].value
        rankings[4] = extraCard[-3].value
        msg = "One Pair"

    # High card
    else:
        rankings[0] = 1
        for i, card in enumerate(hand):
            rankings[i + 1] = card.value
        msg = "High Card"
    return rankings, msg
```

```python
@staticmethod
def check_straight(hand):
    """Check for a straight in the hand, considering Ace as both high and low."""
    # Extract the values from the hand and sort them, removing duplicates
    values = sorted(set(card.value for card in hand), reverse=True)

    # The Ace can be high (14) or low (5), so check both conditions
    straight = False
    high_card = None

    # Check for the special case of a low Ace straight first (A-6-7-8-9)
    if set([14, 6, 7, 8, 9]).issubset(set(values)):
        return True, 5  # Return True with '5' as the highest card in the A-6-7-8-9 straight

    # Check for normal straights
    for i in range(len(values) - 4):
        if values[i] - values[i + 4] == 4:
            straight = True
            high_card = values[i]
            break
        # Check sequences of 5 for a straight, allowing for internal sequence breaks
        sequential = True
        for j in range(4):
            if values[i + j] - 1 != values[i + j + 1]:
                sequential = False
                break
        if sequential:
            straight = True
            high_card = values[i]
            break

    return straight, high_card
```

4.2.1.5.4 Animations

Animations play a crucial role in enhancing the poker web application, it improves the users experience through constant visual feedback from the application.
The most evident animation which I created was the chips animation, this shows the player placing a specific amount of chips into the pot, depending on if they have raised, called or re-raised. Figure 40: Application - Chips and Bet amount Animation, shows the player re-raising and placing $40 worth of chips into the pot, a re-raise will occur if the player is faced with a raise and rather than calling and matching the bet, they double the raise amount to $40. The code shows the conditions for the different chips images being triggered, for the player to bet $40, the last action by the AI will be raise, handled by the aiRaised flag. However, if the player selects call or raise and the last action by the AI isn't raise it will be the standard $20 raise. Alternatively, the AI's actions are handled using the last_action element, which stores the AI's last action.
The starting position of the animation is the player or AI's cards, depending on who's moving the chips, and the end position is towards the pot, which is shown in the bottom of the left code snippet . The code for the other chip animations is very similar, the only differences are the chip images used depending on which raise is taken and value used to represent the bets, and also the starting and end position is slightly different for the player and AI.

```
document.addEventListener('DOMContentLoaded', function() {
    // animation for regular raise or call by the player
    function animatePlayerChipsToCenter(betAmount) {
        const playerChips = document.getElementById('movingChipsPlayer');

        // Create and style the bet amount
        const betAmountText = document.createElement('div');
        betAmountText.textContent = `+$20`;
        betAmountText.classList.add('bet-amount-text');

        // Position the bet amount
        betAmountText.style.position = 'absolute';
        betAmountText.style.left = '27%';
        betAmountText.style.top = '45%';
        betAmountText.style.transform = 'translate(-50%, -50%)';
        betAmountText.style.zIndex = 100;
        document.body.appendChild(betAmountText);

        // Fade in and flash the bet amount
        gsap.fromTo(betAmountText,
            { autoAlpha: 0 },
            {
                duration: 0.5,
                autoAlpha: 1,
                repeat: 3,
                yoyo: true,
                onComplete: () => {
                    gsap.to(betAmountText, {
                        duration: 1,
                        autoAlpha: 0,
                        delay: 1,
                        onComplete: () => betAmountText.remove()
                    });
                }
            }
        );
        // Animation of chips towards the pot
        gsap.set(playerChips, { display: 'block' });
        gsap.to(playerChips, {
            duration: 2,
            left: '15%',
            top: '70%',
            x: '-50%',
            y: '-50%',
            onComplete: () => {
                playerChips.style.display = 'none';
            }
        });
    }
```

Figure 40: Application - Chips and Bet amount Animation

```
// Used to trigger the chip animations depending on what the previous action was.
document.getElementById('actionForm').addEventListener('submit', function(event) {
    const actionValue = event.submitter.value;

    // Check the last action made by the AI
    const lastActionText = document.getElementById('last_action').textContent.trim();
    const aiRaised = lastActionText.includes('AI raises');

    // If the AI raised last and now the player is raising, raise value is 40
    if (actionValue === 'Raise' && aiRaised) {
        sessionStorage.setItem('playerRaised', 'true');
        updatePotValue();
        animatePlayerChipsToCenter40();
    }
    // use the default animation, whne it's a regualr call or raise
    else if (actionValue === 'Call' || actionValue === 'Raise') {
        if (actionValue === 'Raise'){
            sessionStorage.setItem('playerRaised', 'true');
        }
        animatePlayerChipsToCenter();
        updatePotValue();
    }
    // If action is not Raise, playerRaised is cleared
    if (actionValue !== 'Raise') {
        sessionStorage.setItem('playerRaised', 'false');
    }

});
    // Use last action in log to animate AI's chips
    const lastAction = document.getElementById('last_action').textContent.trim();
    const playerRaised = sessionStorage.getItem('playerRaised') === 'true';
    if (playerRaised && lastAction.includes('AI raises')) {
        animateAIChipsToCenter40();
        updatePotValue();
    }
    else if (lastAction.includes('AI calls') || lastAction.includes('AI raises')) {
        animateAIChipsToCenter();
        updatePotValue();
    }
    });
```

Additionally, when the player or AI folds, the cards will be animated to the centre of the table to resemble what would happen if a player fold during a poker hand in real life. This feature keeps the pace of the game lively and visually engaging, mirroring the tempo of real gameplay. The AI fold is handled using the last_action variable, similarly to how the actions were handled when animating the chips. The throwAnimation in the CSS file is used to handle the throw element to the cards, which pushes the cards towards the centre, causing them to be faded out and removed from the view. The players fold is dealt with the same way using the styling, except it's triggered when the user clicks the Fold button, the thrown element is then triggered and the animation takes place for 3 seconds, until a new hand is prompted.

```
// Used to show the AI folding by throwing the cards into the centre of the tbale
document.addEventListener('DOMContentLoaded', (event) => {
  const lastActionText = document.querySelector('#last_action').textContent.trim();

  // Check if last action indicates AI fold
  if (lastActionText === 'AI folds') {
      animateAIFold();
  }
  // aniimation for the cards being thrown
  function animateAIFold() {
      const aiCards = document.querySelectorAll('#botCards .cards');

      aiCards.forEach(card => {
          card.classList.add('thrownAI');

          // Listen for the end of the animation and then remove the cards
          card.addEventListener('animationend', () => {
              card.remove();
          });
      });
  }
});
```

```
// Used to show the PLayer folding by throwing the cards into the centre of the tbale, triggered by clicking the fold button
document.addEventListener('DOMContentLoaded', () => {
    const foldButton = document.querySelector('button[value="Fold"]');
    if (foldButton) {
        foldButton.addEventListener('click', (e) => {

            // Trigger the animation for folding cards
            const playerCards = document.querySelectorAll('#playerCards .cards');
            playerCards.forEach(card => {
                card.classList.add('thrown');
            });
            // 3 second animation
            setTimeout(() => {
            }, 3000);
        });
    }
});
```

```
@keyframes throwAnimation {
  to {
    transform: translateX(10vw) translateY(-30vh);
    opacity: 0;
  }
}

.thrown {
  animation: throwAnimation 3s forwards;
}
```

Figure 41: Application - Cards Folded Animation

After the AI makes an action, an action bubble is displayed. This action bubble states what action the AI has taken, displays the message for 2 seconds, it then fades out and disappears. When making the styling for this pop up, I wanted it to resemble a speech bubble, while being informative of what action the AI has taken but still remaining sleek and subtle. The background and backdrop-filter are used to create that soft blur which helps the bubble blend in but remain clear, and the transition tag helps the bubble pop up and disappear seamlessly.

```
// Logs the AI's last action and displays it beside the AI for 2 seconds
document.addEventListener('DOMContentLoaded', (event) => {
  const actionLogItem = document.querySelector('.actionLogItem');
  if (actionLogItem) {
      setTimeout(() => {
          actionLogItem.style.opacity = '0';
          setTimeout(() => {
              actionLogItem.remove();
          }, 0);
      }, 2000);
  }
});
```

```
#actionLog {
  font-family: 'Orbitron', sans-serif;
  position: absolute;
  right: 60%;
  top: 7%;
  transform: translateY(-50%);
  background: rgba(255, 255, 255, 0.2);
  backdrop-filter: blur(10px);
  color: #000;
  padding: 10px;
  border-radius: 5px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.2);
  z-index: 10;
  max-height: 200px;
  overflow-y: auto;
}

.actionLogItem {
  opacity: 1;
  transition: opacity 0.5s ease-in-out;
}
```

Figure 42: Application - player action bubble

Lastly, the AI's decision-making process is given a visual form. While the AI thinks about its next move, a subtle yet clear processing icon signals that it’s considering its next move. This informs the player that the game is actively processing and also replicates the natural thinking time a human opponent would take when deciding on their next move in a real poker game. The function which handles the AI processing Gif is triggered by the user making an action which requires a response (Call , Check , Raise). This causes the Gif to pop up and be displayed for 4.5 seconds, it then disappears with a response and the action is back on the user to make another action.

```
// This is used to trigger and display the AI processing robot Gif
document.addEventListener('DOMContentLoaded', function() {
    const aiGif = document.getElementById('aiGif');
    const aiGifImage = aiGif.querySelector('img');
    const aiGifText = aiGif.querySelector('p');

    const actionForm = document.getElementById('actionForm');

    actionForm.addEventListener('submit', function(event) {
        const actionValue = event.submitter.value;

        // Check if the action requires AI to make a decision, triggered by the players actions using the action form
        if (actionValue === 'Call' || actionValue === 'Check' || actionValue === 'Raise') {
            // Show the AI GIF and text indicating AI is deciding
            aiGif.style.display = 'block';
            aiGifImage.style.display = 'block';
            aiGifText.style.display = 'block';

            // Hide the AI GIF after 4.5 seconds
            setTimeout(() => {
                aiGif.style.display = 'none';
                aiGifImage.style.display = 'none';
                aiGifText.style.display = 'none';
            }, 4500);
        }
    });
});
```

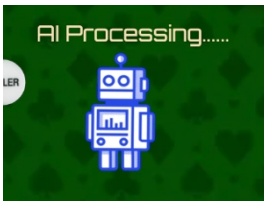


Figure 43: Application - AI Processing Robot

4.2.2 AI Agent

4.2.2.1 Abstraction

```
def bucketing(hand, communityCards=[]):
    # Define the card values, Ace is represented as 14 and is the highest vlaue
    card_values = {
        6: 6, 7: 7, 8: 8,
        9: 9, 10: 10, 11: 11, 12: 12, 13: 13, 14: 14
    }
    # Assign the cards from the hand into two variables
    card1, card2 = hand

    # Map the card values using the assgined values to get their ranks.
    rank1, rank2 = card_values[card1.value], card_values[card2.value]

    # Pre flop = ( < 3 cards)
    if len(communityCards) < 3:
        # Initial strength is the sum of the ranks of the two cards.
        strength = rank1 + rank2

        # Pairs: If both cards have the same value (E.G: 9 spades & 9 hearts) = +8 points
        if rank1 == rank2:
            strength += 8

        # Top Pairs: Jacks to Ace pairs = +10 points
        if rank1 == rank2 and rank1 >= 10:
            strength += 10

        # Suited: Cards of the same suit = +3 points
        if card1.suit == card2.suit:
            strength += 3

        # Connectors: cards which are adjacent / close in value = adjacent (E.G: 8 & 9) +3 points / 1 gap (E.G:  7 & 9) +1 point
        if abs(rank1 - rank2) == 1:
            strength += 2
        elif abs(rank1 - rank2) == 2:
            strength += 1

        # Additional condition for different, unsuited cards both 10 or above
        if card1.suit != card2.suit and rank1 != rank2 and rank1 >= 10 and rank2 >= 10:
            strength += 2

        # Additional Bonus for Ace-6 and Ace-7 as they are bottom end connectors
        if (rank1 == 14 and rank2 == 6) or (rank1 == 6 and rank2 == 14):
            strength += 2  # Bonus for Ace-6

        elif (rank1 == 14 and rank2 == 7) or (rank1 == 7 and rank2 == 14):
            strength += 1  # Bonus for Ace-7

        return strength
```

The bucketing function is the hand abstraction that's used for the AI, it's used to simplify the decision space of the AI by categorising the strength of hands into buckets. This simplification allows the AI to make decisions based on hand strength categories rather than individual hand values. Pairs, suited cards, connectors, and high card combinations are assigned additional strength points, which will be used to determine the hand's rank.
Once the bucket values are calculated, the hand_rankings function assigns a specific rank to the hand based on its bucket value. This rank is crucial for comparing different hands and making decisions.
If the community cards haven't been dealt, it computes the hand's strength based only using the players cards. The strength is initially the sum of the card values and the bonuses are calculated based on a number factors which increase the strength of the hand. This point system was created to achieve a similar pre-flop hand chart as Figure 4: CardMates - Pre-Flop Hand strength

- Pairs: If the two hole cards are the same, 8 points are added to the hand strength. A 10 point bonus is awarded for pairs that are Jacks or better (Jacks, Queens, Kings, Aces).
- Suited Cards: If the two cards are the same suit, 3 extra points are given, as the hand could potentially leading to a flush.
- Connectors: points are given for consecutive card values or cards with one gap between them, as there's potential for a straight. There are 2 points given to adjacent valued cards and 1 point to cards with a one card gap.
- High-Value Cards: If the cards are high-valued (10 or above) but not suited or paired, a small 2 point bonus is added.
- Special Cases for Ace-6 and Ace-7: These combinations receive a bonus as they have potential for a straight with the bottom end connected to the Ace. This is a unique rule to Short Deck poker where ace can be the bottom of the straight.

```
def hand_rankings(bucket_value):
    # Used to find the strength of the hand using the bucketing function.
    strength = bucket_value
    # Based on strength, they're hand is assgined a rank
    if strength > 29: return 35
    elif strength > 25: return 28
    elif strength > 23: return 25
    elif strength > 20: return 22
    else: return 17
```

This point system in the bucketing function is used to create the pre-flop hand strength buckets. The hand_rankings function is used to map the preliminary strength score to a final hand ranking, which is critical in comparing hands to make decisions. This ranking simplifies the variety of possible hand strengths into a manageable number of categories that the AI can use for strategic decisions.

```
# post-flop mor ethan 3 cards dealt, calculate the hand value considering the community cards.
values = PokerPlayer.calculate_hand_value(hand, communityCards)
# Round values into three groups, low, med, high
roundedValues = [values[0]]
# Categorise the value into buckets based on values
for val in values:
    if val >= 6 and val <= 9:
        roundedValues.append(1)

    elif val >= 10 and val <= 12:
        roundedValues.append(2)

    elif val >= 13:
        roundedValues.append(3)

return tuple(roundedValues)
```

If community cards are present (post-flop), it calls another function to calculate hand value, then categorises the value into buckets: low (1), medium (2), or high strength (3). It does this for every card in the players best possible hand from the combined player and community cards. The first number tuple describes the value of the hand, see Figure 44: Post-Flop Hand Strength which present a value of 1 which is the lowest value (high card). The remaining numbers represent each other card in the hand and which category they belong to.

```
(1, 2, 2, 2, 1, 1)
```

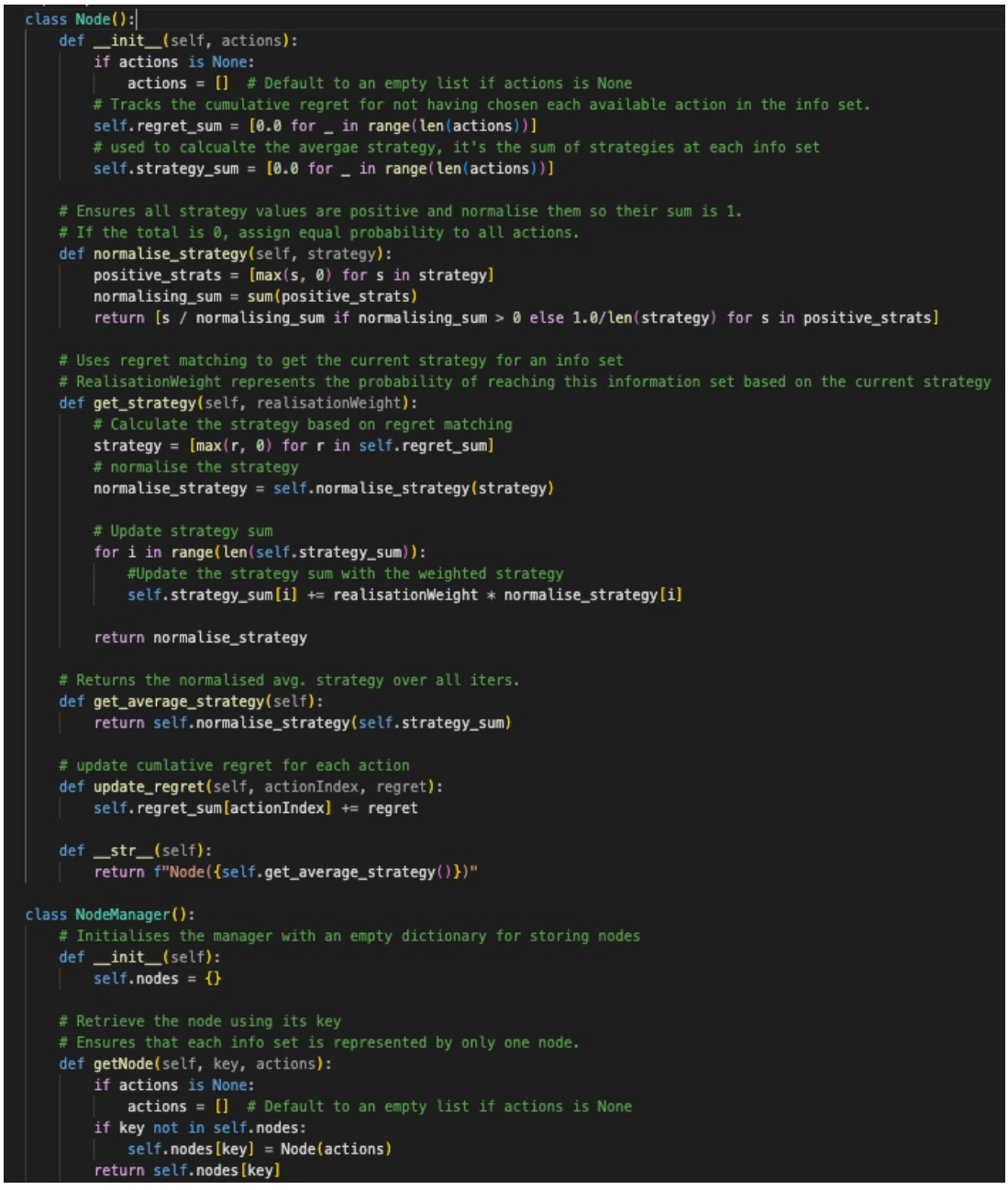
Figure 44: Post-Flop Hand Strength

4.2.2.2. Decision Points Storage Handling

The Node and NodeManager classes handle the storage of decision points in the game, also known as information sets and the strategies associated with those decision points, they act as the command centre of the operation. The strategies evolve as the game progresses and the AI learns from the outcomes of its decisions. The node represents a decision point in the game, each node corresponds to a point where the AI must decide on an action.

- The regret_sum array holds the cumulative regret for not having taken each possible action from the information set in past games. Regret is a measure of how much better the AI could have done if it had taken a different action, given what it knows now after the game.
- The strategy_sum array tracks the sum of strategies used over time at this node. A strategy is a probability distribution over the possible actions.
- The normalise_strategy method ensures that the probabilities in a strategy sum to 1 and are not negative. This is important because the strategy represents the likelihood of choosing each action.
- The get_strategy method uses regret matching to determine the current strategy for the node, weighted by the likelihood of reaching that information set (realisationWeight).

- The get_average_strategy method computes the average strategy over all iterations, which converges to the Nash equilibrium strategy as the AI trains over many games.
- The NodeManager serves as a storage system for all the nodes in the game. Since the number of decision points can be quite large in complex games like poker.
- The nodes dictionary acts as a database where the keys are the unique identifiers for each information set, and the values are Node objects.
- The getNode method retrieves a node given an information set key. If a node for a particular key doesn't exist, it creates a new one with the possible actions for that information set.



```python
class Node():
    def __init__(self, actions):
        if actions is None:
            actions = []  # Default to an empty list if actions is None
        # Tracks the cumulative regret for not having chosen each available action in the info set.
        self.regret_sum = [0.0 for _ in range(len(actions))]
        # used to calcualte the avergae strategy, it's the sum of strategies at each info set
        self.strategy_sum = [0.0 for _ in range(len(actions))]

    # Ensures all strategy values are positive and normalise them so their sum is 1.
    # If the total is 0, assign equal probability to all actions.
    def normalise_strategy(self, strategy):
        positive_strats = [max(s, 0) for s in strategy]
        normalising_sum = sum(positive_strats)
        return [s / normalising_sum if normalising_sum > 0 else 1.0/len(strategy) for s in positive_strats]

    # Uses regret matching to get the current strategy for an info set
    # RealisationWeight represents the probability of reaching this information set based on the current strategy
    def get_strategy(self, realisationWeight):
        # Calculate the strategy based on regret matching
        strategy = [max(r, 0) for r in self.regret_sum]
        # normalise the strategy
        normalise_strategy = self.normalise_strategy(strategy)

        # Update strategy sum
        for i in range(len(self.strategy_sum)):
            #Update the strategy sum with the weighted strategy
            self.strategy_sum[i] += realisationWeight * normalise_strategy[i]

        return normalise_strategy

    # Returns the normalised avg. strategy over all iters.
    def get_average_strategy(self):
        return self.normalise_strategy(self.strategy_sum)

    # update cumlative regret for each action
    def update_regret(self, actionIndex, regret):
        self.regret_sum[actionIndex] += regret

    def __str__(self):
        return f"Node({self.get_average_strategy()})"

class NodeManager():
    # Initialises the manager with an empty dictionary for storing nodes
    def __init__(self):
        self.nodes = {}

    # Retrieve the node using its key
    # Ensures that each info set is represented by only one node.
    def getNode(self, key, actions):
        if actions is None:
            actions = []  # Default to an empty list if actions is None
        if key not in self.nodes:
            self.nodes[key] = Node(actions)
        return self.nodes[key]
```

4.2.2.3. CFR Algorithm

The CFR algorithm is the training algorithm which was used to train the AI agent through self-play. It starts by creating a copy of the game's history, this prevents the original history from being changed. Folding results in a quick assessment of the winner, on the other hand, the winner will be discovered using the functions mentioned earlier in section 4.2.1.5.3 Hand Evaluation / Poker Logic, and finally the pay-out will be calculated and returned using the calculate_payout function.

```
# Calcuate the total playout for all the players
def calculate_payoff(players):
    return sum([p.bet for p in players])
```

```
def CFR(deck,history,players,reachProbs,currentPlayer,sets,limit):
    # Create a deep copy of the game history to prevent changing the original history
    history = deepcopy(history)
    # Check whether the termial state has been reahced
    if is_terminal(history,players):
        # If the last action was fold then calcualte the payoff
        if history[-1] == "Fold":
            return calculate_payoff(players)
        else:
            communityCards = players[0].communityCards
            # Calculate the hand strength for each player using the community cards.
            hand_strength_player = [PokerPlayer.calculate_hand_value(player.hand, communityCards) for player in players]
            # find the winner based on the calculated hand value
            winners = PokerGame.getWinningHands(hand_strength_player)
            # If the game is a draw then return 0 / If the current player wins then return their payoff / otherwise return the negative
            if len(winners) == 2:
                return 0
            elif winners[0] == currentPlayer:
                return calculate_payoff(players)
            else:
                return -calculate_payoff(players)

    # Check if the previous betting round is over
    if round_completed(history):
        history += ["Round"]
        # Deal the flop if there are less than 3 community cards.
        if len(players[0].communityCards) < 3:
            newCards = cards.dealdeck(3,deck)
        # DFeal the turn / river
        else:
            newCards = cards.dealdeck(1,deck)
        # Update community cards for all players
        for player in players:
            player.communityCards += newCards


    #if bet limit reached, ban raising
    if history[len(history) - limit : len(history)] == ["Raise"]*limit:
        actions = ["Call","Fold"]
    #prevents index error
    elif len(history) == 0:
        actions = ["Call","Fold","Raise"]
    #necessary response actions, fold removed when check is possible
    elif history[-1] == "Raise":
        actions = ["Call","Fold","Raise"]
    elif history[-1] == "Check" or history[-1] == "Call" or history[-1]=="Round":
        actions = ["Check","Raise"]
```

The game will be progressed through the different stages using the dealdeck function and the actions which can be taken by the AI's are determined by the history of the game. It is vital that the set up mirrors the actual game that the user will play against the AI, because the AI must have encountered the exact scenarios before in training to be able to act on them in the real game.

```
# Converts the actions into a history string.
def history_str(history):
    symbols = {"Check": "X", "Fold": "F", "Raise": "R", "Call": "C", "Round": "_"}
    return ''.join(map(lambda action: symbols[action], history))
```

The AI uses the current game state, the history and bucket value, to retrieve the relevant information set. The history is converted into a string represented by a character, in Figure 45: info set key, the previous history is call, check and then the next stage (flop). The numbers represent the bucket value, before the flop is dealt its represented by the bucket value (28), post flop its represented by the hand value, which was explained in 4.2.1.5.3 Hand Evaluation / Poker Logic.

```
infosetkey:  ('C', 28)
infosetkey:  ('CX_', (2, 1, 1, 1, 1))
```

Figure 45: info set key

The information set includes the possible actions and the associated strategies developed so far. CFR uses the strategies from the information sets to determine the next action. It then updates regrets based on the outcomes of these actions. Regrets tell the AI which actions could have been more profitalbe in hindsight, which guides how the strategy will be adjusted in future games.
CFR calls itself recursively to determine new game states resulting from taken actions. This recursive nature allows the AI to learn strategies over the entire game tree.
Finally, the CFR function computes and returns the utility of the node (game state). This utility is used to update the regrets and strategies in the information sets.

```python
# Before entering the strategy computation in CFR, determine if it's pre-flop or post-flop
if len(players[0].communityCards) < 3:
    # Pre-flop: Use bucketing to get the preliminary bucket value based on hand strength
    pre_flop_bucket_value = bucketing(players[currentPlayer].hand, players[0].communityCards)
    # Then use the hand_rankings function to adjust the bucket value based on the hand's rank
    bucket_value = hand_rankings(pre_flop_bucket_value)
else:
    # Post-flop: Use the original bucketing function, which already considers community cards
    bucket_value = bucketing(players[currentPlayer].hand, players[0].communityCards)

# Calculate the opponent's index , their posotion
opponent = (currentPlayer + 1) % 2

## Retrieve the information set for the current game state and available actions.
info_set_key = (history_str(history), bucket_value)

info_set = sets.getNode(info_set_key, actions)
# Get the current strategy from the information set.
strategy = info_set.get_strategy(reachProbs[currentPlayer])

# Initialise regrets for each action.
newRegrets = [0 for i in range(len(actions))]

for i in range(len(actions)):
    #gets each action and its probability of being chosen
    actionProb = strategy[i]
    action = actions[i]
    #modifies current player's reach probability
    newReachProbs = reachProbs.copy()
    newReachProbs[currentPlayer] *= actionProb

    # Simulate the game state after taking the action.
    player = deepcopy(players)
    if action == "Raise":
        player[currentPlayer].bet = player[opponent].bet + 20
    elif action == "Call":
        player[currentPlayer].bet = player[opponent].bet

    d = deepcopy(deck)

    # Recursively call CFR for the new game state.
    newRegrets[i] = -CFR(d, history + [action], player, newReachProbs, opponent, sets, limit)

#value is regrets weighted by action probability
nodeValue = 0
# Calculate the expected utility of the node.
for i in range(len(strategy)):
    nodeValue += strategy[i] * newRegrets[i]

# Update cumulative regrets for each action in the information set.
for i in range(len(strategy)):
    regret = reachProbs[opponent] * (newRegrets[i] - nodeValue)
    info_set.update_regret(i, regret)

return nodeValue
```

In the context of the agent, the CFR functions role is to simulate countless game scenarios, each time improve the AI's strategy through regret minimisation. Over many iterations, this process enables the AI to go towards a better strategy, resulting in an improvement in the AI's overall performance in the game.

4.2.2.4 Training The AI

Training the AI is a crucial step in the development of the agent. The training functions runs depending on the amount of time chosen to run it. The AI plays games against itself to learn the game and the decisions to make without any huma interaction.

The CFR function is called on each iteration, updating the strategy based on the game's outcome. The value returned by the CFR function reflects the utility from the round, considering the actions taken. This value is used to adjust the AI's strategy after each round.

The average training time varied depending on whether I was using my laptop or not, if I left it dormant for any amount of time it would vastly increase the average iteration time to upwards of 30 seconds per iteration. However, If I was actively using my laptop, the training time would be around 9 seconds.

This training data once completed is saved to a file which will be described in the next section.

```
# Function used to train the agent, the betting limit is 4
def train(sets,num_iterations, limit=4):
    for i in range(num_iterations):
        #creates fresh player list and deck
        playerList = cards.PokerPlayer.getPlayerList(2,300)
        deck = cards.Cards.shortDeck()
        history = []
        #gives both players their cards
        for player in playerList:
            player.hand = cards.dealdeck(2,deck)
        #sets first player as small blind, second as big
        playerList[0].bet = 10
        playerList[1].bet = 20
        #performs 1 iteration of training
        value = CFR(deck,history,playerList,[1,1],0,sets,limit)
    return value
```

4.2.2.5 Saving The Training Data

After completing the training process, it is integral that the training data is saved. This data includes the strategies that the AI has developed throughout the training process. The data is saved in a file called sets, followed by the number of training iterations, Figure 46: Training files. The process involves serialisation, which is converting the AI's state, into a format that can be saved to a file. This is done using the dill module in Python, where a custom unpickler class is implemented. This class has methods to serialise and deserialise the AI's intricate data structures.

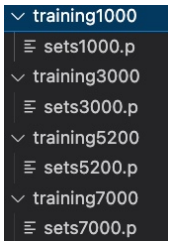


Figure 46: Training files

```python
# Custom unpickler to handle loading of specific classes
class Dill(dill.Unpickler):
    # Override to specify how to find and load the NodeManager and Node classes during unpickling.
    def find_class(self, module, name):
        if name == 'NodeManager':
            return NodeManager
        if name == "Node":
            return Node
        return super().find_class(module, name)

# Function used to save the objects to the file
def save_sets(obj, filename):
    try:
        with open(filename, "wb") as file:
            dill.dump(obj, file)
    except IOError as e:
        print(f"Failed to save object")

# function to set the training details (how long)
def train_time(sets, mins, startItr, limit=4, saveDir="Saves", saveInterval=100):
    # Initialise the training information and start timer.
    info = sets
    start = time.time()
    iterations = startItr

    # Train for a specified number of minutes.
    while (time.time() - start)/60 <= mins:
        iterations += 1
        # calls the train function and does a single iteration of CFR training.
        train(info, 1, limit)

        # save the current state
        if iterations % saveInterval == 0:
            save_sets(info, os.path.join(saveDir, "sets" + str(iterations) + ".p"))
    end = time.time()

    # print what the training achieved and save the chnages
    save_sets(info, os.path.join(saveDir, "sets" + str(iterations) + ".p"))
    print("total iterations completed:", iterations)
    print((end - start) / (iterations - startItr), "seconds per iteration on AVG")
```

Essentially, this step is about capturing the AI's strategies at certain points in the game. This file can then be loaded into the application where at every decision point (node), the AI will look up the node it’s currently located at in the file and see what is the calculated strategy for that certain game state. This process will continue throughout the game, the saved file will act as the brain of the operation.

4.3 Conclusions

This section describes the complete development process of the poker web application. Starting with the components that make the application function as desired, followed by the aspects that led to the creation of the poker game and lastly, the process that led to the development of the AI agent.

5. Testing and Evaluation

5.1. Introduction

The testing and evaluation phase is where all elements of the web application are carefully tested. White Box Testing begins the testing process, looking at the code to make sure every input leads to the right output, which is crucial for a smooth user experience.

I then moved onto Usability Testing, this involves receiving feedback from real users and find out what they had to say about their time using the app, from signing up to playing poker against the AI agent.

Speed Performance Testing was all about getting the page load times down, making sure players aren't left waiting and unsure of what's happening when they make a request.

The AI performance testing is where the most important testing was done to see how effective the learning process actually was and if the agent actually learnt anything. This section walks you through the various versions of the agent and runs various tests against random bots , each other and human players. It also takes you through the differences in decision making using two example hands to show the evolution in the training. Each test shaped the web application into a more polished platform.

5.2. System Testing

5.2.1 White Box Testing

White box testing is a form of testing that makes sure that the flow of inputs through the code prodcues the correct outputs. In this section, I will present a series of white box tests conducted on my poker web application. These tests are carefully designed to verify that all components of the system work as expected in varoius scenarios, ranging from user login to gameplay mechanics.

Furthermore, each test outlined is aimed at evaluating specific functional aspects of the application, such as login processes or game dashboard display accuracy. The expected outcomes are defined clearly to establish a standard for success. The results of these tests serve as a test of reliability and functionality of my system. By going through these tests, it ensures a seamless and intuitive experience for the users. (37)

Test	Expected Result	Result
Users not logged in try to play Poker	User should be redirected to the homepage where they will be forced to sign in.	Pass
Login / Registration	Users can register an account / Login successfully	pass
Logging Out	When users log out they are redirected to the homepage with the login pop up	pass
Do all the metrics on the homepage display the correct data?	The metrics on the homepage query all the users data and display the pie charts and line graph correctly, the daily / weekly buttons work as expected	pass
Change Avatar	When the user selects a new avatar and selects the save button, they are reverted to the manage profile page with their new avatar displayed.	pass
Can the user change their Password?	When the user selects change password and they enter current password they will be able to select a new password.	pass
Save profile details	When the user enters a new nickname and select 'save changes', it saves all the changes made to the database and reverts to the homepage.	Pass
Does the hand Review page show the correct content?	When the user attempts to review a hand, all the necessary content is displayed	Pass
Does the system warn the user when trying to view a game with no hands played?	When the user attempts to review hands from a game where no hands were played, they are redirected to the page with a warning message	Pass
Does the 'View Hand' button show all the content for the selected hand?	The view hand button dynamically changes all the displayed content and decision data related to that hand.	Pass
Does the decision data pop up show the correct information?	The user can see all the correct data related to the AI's decisions for each action taken.	Pass
Do all the users actions in the poker game do what's intended?	The action buttons do what's intended of them (raise, call, check, fold)	Pass
Does the view previous hand button show the previous hand and the related decision data?	The review previous hand firstly gets if a hand has been played yet, if not it will display a warning message, if a hand has been played it will display all the required information	Pass
Does the user receive constant and informative from the system when playing poker?	When the user is playing against the AI agent, they receive constant feedback about what's happening, such as (AI processing or Loading next hand)	pass

Table 1: White Box Testing

5.2.2 Usability Testing

Usability testing is an essential phase in the development of an application, offering insights into real users interactions with the system. This section focuses on the experiences of users as they navigate through the poker application. It aims to grade the intuitiveness of the user interface, the ease of registration, the applications aesthetic, the clarity of the game's statistics, and the overall user experience during gameplay against the AI agent.

Question	Answer (User 1)	Answer (User 2)
Was it easy to create register and login to your account?	Easy	Easy
How did you find navigating the application?	Very Easy	Easy
Did you find the application aesthetically pleasing?	Yes, Very pleasing	Kind of
Were you able to easily change your avatar?	Yes, very straightforward	Yes, I worked it out
Did the find the game statistics easy to understand?	Yes, I understand what most of it means	Yes, I understand what most of it means
Did the AI's decision making process in the hand review page provide useful information?	Yes	Kind of
Was playing poker against the AI agent intuitive, did you receive the necessary feedback from the system?	Yes, it was very intuitive to play against	No, I was left guessing at time

Table 2: User Usability Test

I found the results to be relatively positive, in terms of the basic operations and user interface the users seemed to be content with the web application. This is vitally important, as the basic navigation and uses cases are at the core of the application and shouldn't require any technical ability and knowledge of the domain.

However, there wasn't complete satisfaction with regards to the understanding of the game statistics and the AI's decision making process information. The users I had test the application have basic prior knowledge of poker and they weren't completely satisfied with these aspects. I'm overall content with these sections as I wanted to find the balance between a detailed explanation and not over explaining what's going on. I would probably add some more detail in the decision making process section in future iterations of the web application.

Finally, There was slightly mix responses on the system feedback when playing the poker game. Overall, there may have been more animations and pop ups I could have added to inform the user. Given the mixed reaction, I'm satisfied with the functionality and system feedback within the poker game.

In conclusion, I found the usability testing very effective and informative. I'm satisfied with the outcome of the application given the users feedback, even though there are some slight improvements that could be made, which is to be expected.

5.2.3 Speed Performance Testing

Speed performance testing is an invaluable aspect of the quality assurance process, utilised to test the responsiveness and speed of the web application. This section displays the various metrics used by google lighthouses speed testing process, found in built into the chrome browser, which is used to assess the efficiency of the application. (38)

Page	Speed Index	Total blocking Time	First Contentful Paint	Largest Contentful Paint	Cumulative Layout Shift	Overall Score
Homepage	0.7s	0ms	0.7s	1s	0.093	96
Manage Profile	0.5s	0ms	0.4s	0.7s	0.011	100
Hand review	0.5s	0ms	0.5s	0.9s	0.014	99
Play Poker	2.3s	0ms	0.5s	0.6s	0.003	93

Table 3: Speed Performance Testing

I tested each page individually to get the detailed information on the performance and flaws of each page. The score and overall performance of the first three pages is very high, the speed index, which is used to determine the overall loading speed of the page, is of a very high standard.

Additionally, I made a number of changes to improve the score of each page. One of the changes I. made was changing all the images used from PNG to Webp format and ensuring all the images are the correct size that will be displayed. Changing the format resulted in a 30% improvement in the storage size required for these images.

Furthermore, I moved some of the important styling for the poker page into the HTML file rather than the CSS file to improve the initial loading speed. The styling file was quite large and moving the essential CSS to the HTML file resulted in a 0.4 second decrease in the initial loading speed.

However, the only issue in the performance of the application is the poker application aspect. I ran a number of tests on the database and the response time wasn't an issue and due to the fact the other pages aren't having any speed issue which are far more database intensive would suggest it can't be to do with the database. I suspect the issue it to do with the AI's response time, as it's required to check the whole file for the information set and then make a decision based of the strategy and that will take a second or two.

Overall I'm very satisfied with the speed performance results and the improvements I've made to increase the overall score.

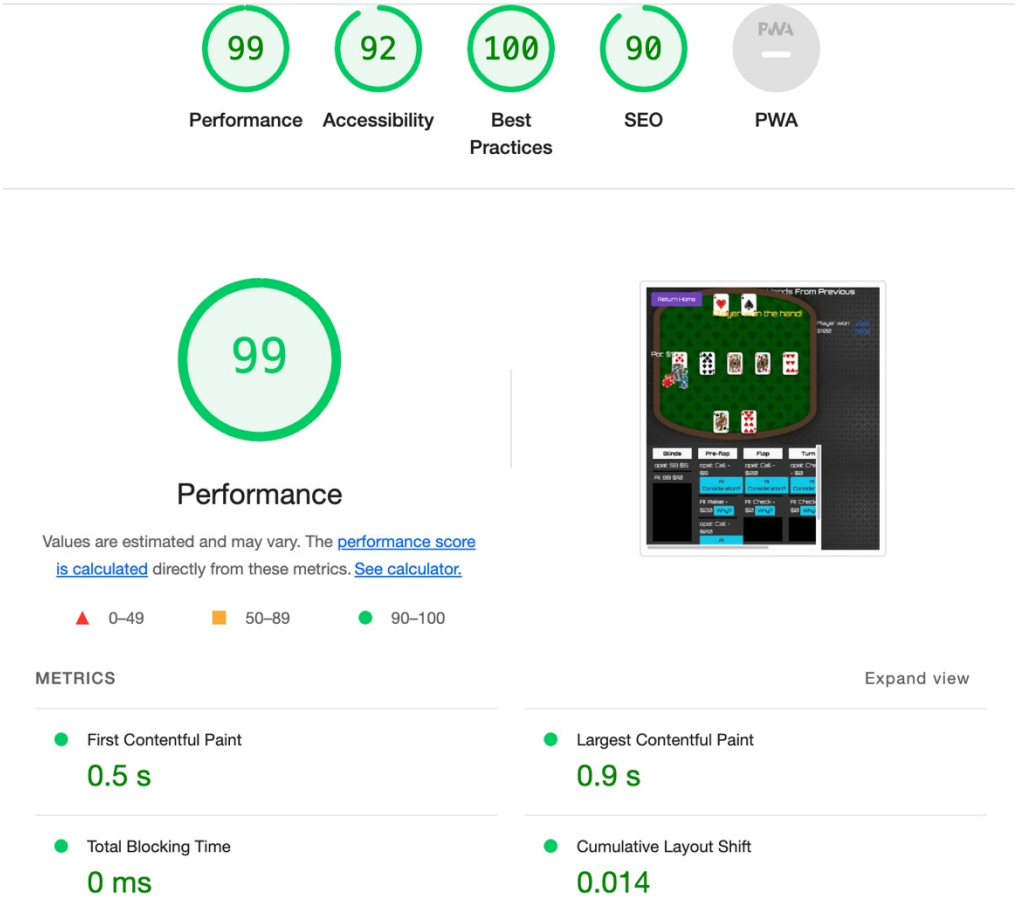


Figure 47: Speed Performance Test

5.2.4 AI Performance Testing

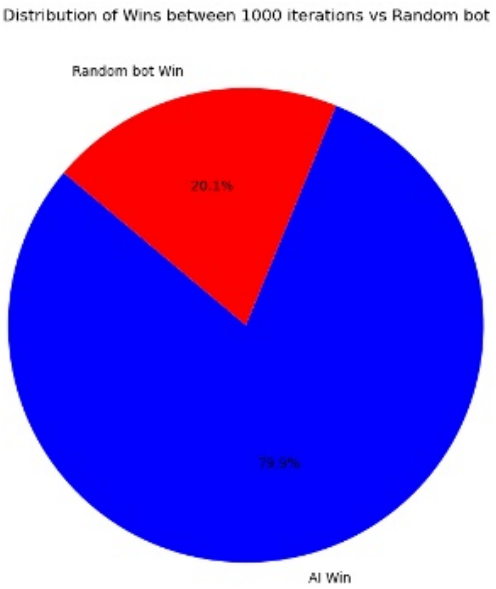
5.2.4.1 AI's vs Random Bot

This section outlines performance of the AI agent during various stages of development and training. I ran a series of simulations to investigate the efficacy of the different AI's against a bot that acts completely randomly. For this tests I decided to run the simulations for 300,000 iterations, which I thought was an adequate amount of iterations to get a true reflection of the results and the AI's incremental improvements over time.

The visualisations I chose to create track the win rate of both players and the monetary gain or loss of the players. I created a pie chart to illustrate the distribution of wins and a line graph to show the amount won by the AI against the random bot over the total number of iterations. Additionally, the line graph includes the average amount won or lost by the AI per 100 hands, which is considered to be a good way to identify performance of a player.

AI (Iterations)	Win %	Avg. win per 100 hands	Big Blinds won per 100 hands
1000 Iterations	79.9	2036.6	101.83
3000 Iterations	83.1	2195.1	109.76
5000 Iterations	84	2206.4	110.32
7000 Iterations	84.5	2224.8	111.24

Table 4: AI vs Random Bot performance



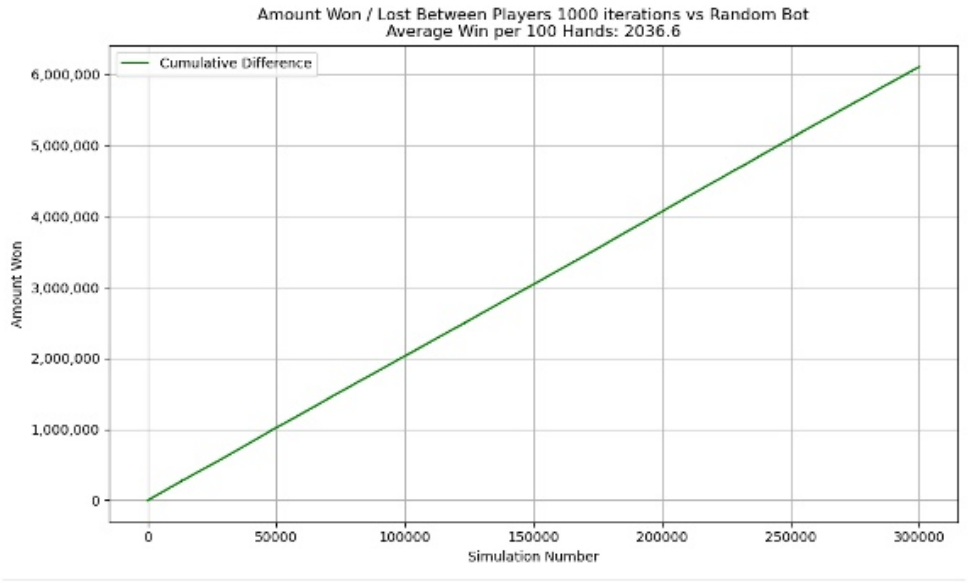
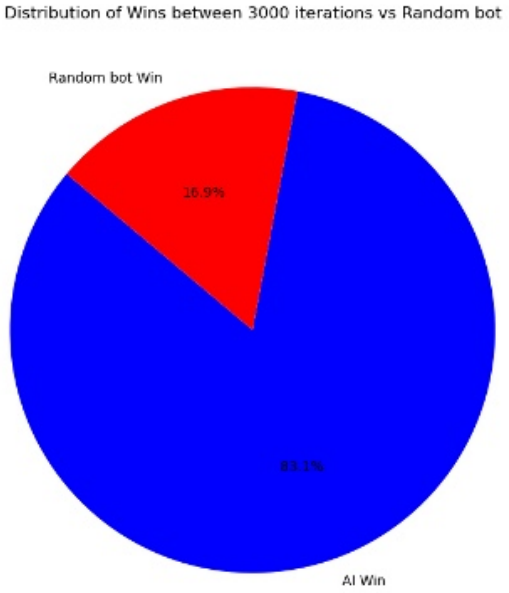


Figure 48: AI vs Random Bot 1000 itrs



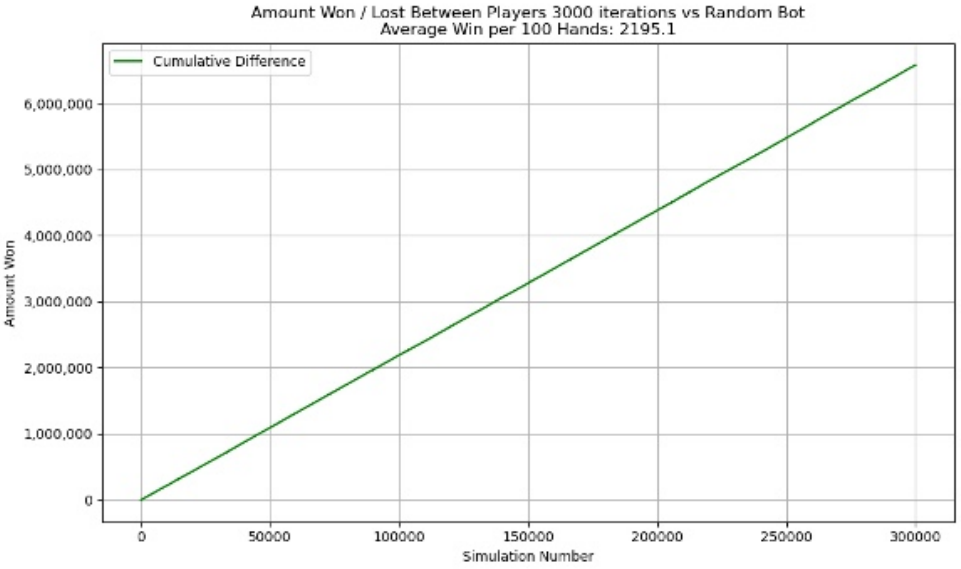
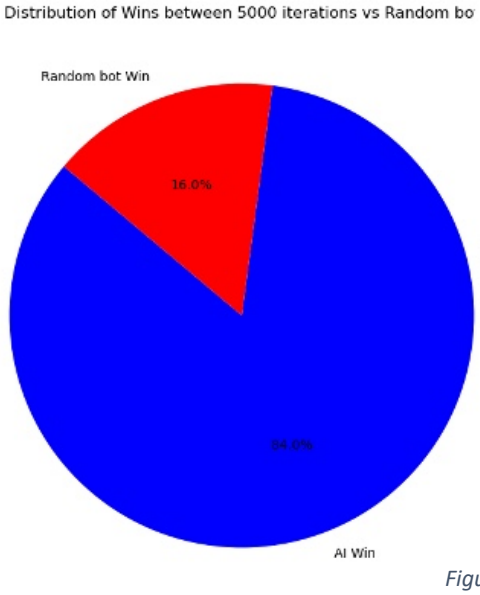


Figure 49: AI vs Random Bot 3000 itrs



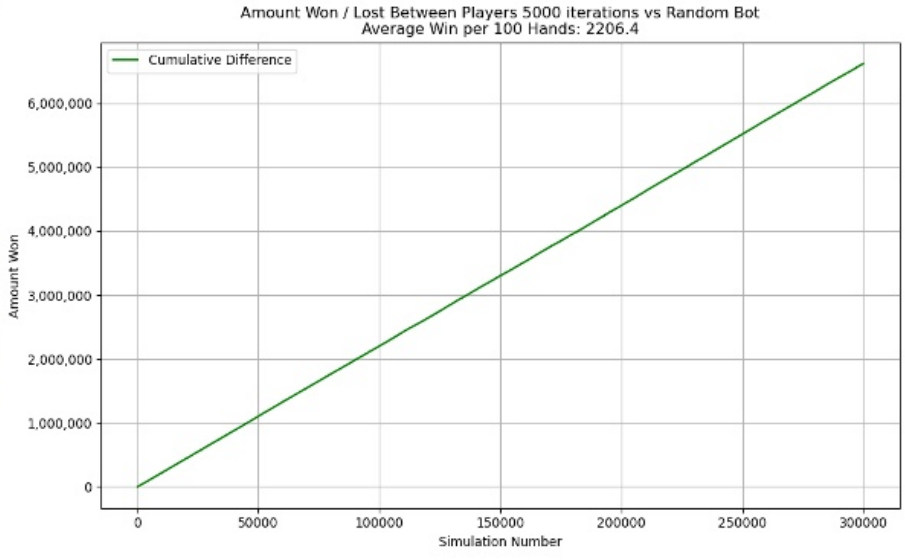


Figure 50: AI vs Random Bot 5000 itrs

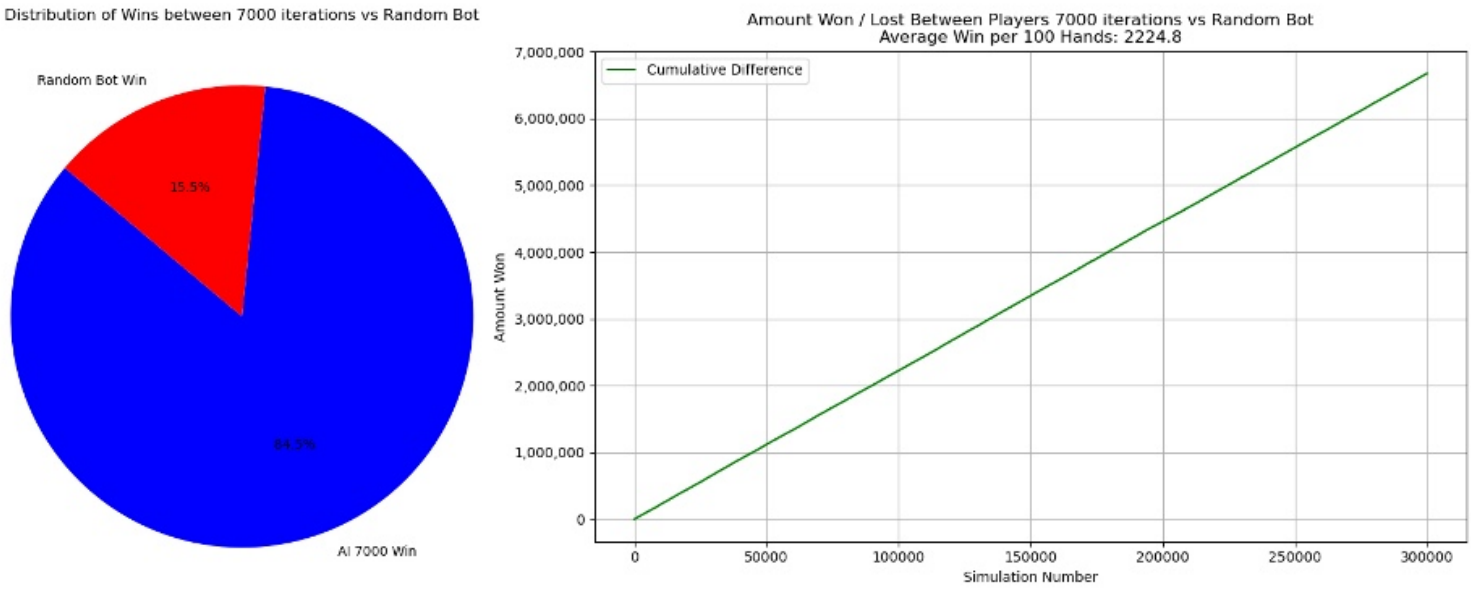


Figure 51: AI vs Random Bot 7000 itrs

The findings which are outlined in the table, as well as highlighted in more detail in the charts, indicate a clear upward in both win rate and average winnings as the number of training iterations increases. Based on the on the win rates from the 1,000 training iterations to the 3,000 iterations there was a 3.2% increase in the win rates and a $158.5 increase in the amount won every 100 hands, this is a considerable increase which highlights the considerable improvements in efficacy of the AI. The following two jumps in training iteration from 3,000 to 5,000 and then to 7,000, also displayed an increase in the overall performance and outcome of the AI's. Their win rates showed an increase of 0.9% and 0.5% respectively. Additionally, the amount won every 100 hands was recorded as $11.3 and $18.4 increase respectively.

The data illustrates a correlation between the number of iterations and the AI's performance, as the AI engages in more hands, its strategy refines, leading to increased wins and earnings. The results outline the AI's learning curve and its potential for increased performance if the training is continued and further improved and refined.

5.2.4.2 AI vs AI

AI (Iterations)	Avg. win per 100 hands	Big Blinds won per 100 hands
3000 vs 1000	239.4	11.97
5000 vs 1000	299.5	14.95
7000 vs 1000	333.4	16.67
7000 vs 3000	124.2	6.21
7000 vs 5000	42.9	2.145

Table 5: AI vs AI performance

This section looks at the dynamics between different iterations of the AI. Contrastingly, to the previous section which looked at AI vs Random bot evaluations, this set of tests places different volume of training iterations against each other to discover if the increased training volume leads to an improved strategy and therefore improved performance and outcome.
I conducted a series of 300,000 simulations with the 1,000 iterations as the provisional test case, I then used the most trained iteration to play against the other AI's to see how it performed. I would expect the improvement in performance to be much smaller than the random bot as all the AI system have some sort of basic strategy unlike the random bot.

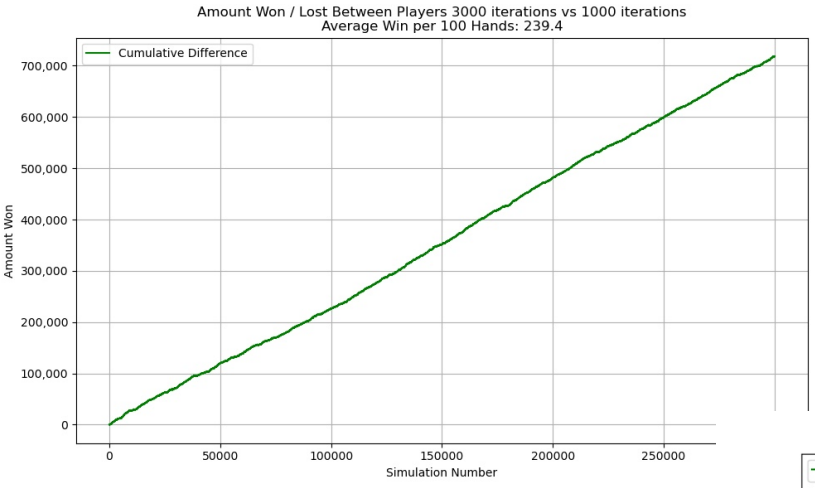


Figure 52: AI vs AI 3000 itrs vs 1000 itrs - Line Graph

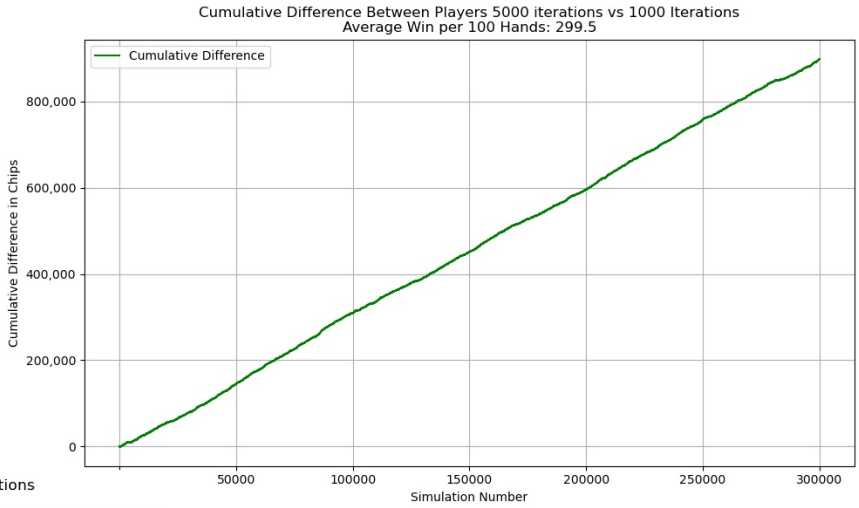


Figure 53: AI vs AI 5000 itrs vs 1000 itrs - Line Graph

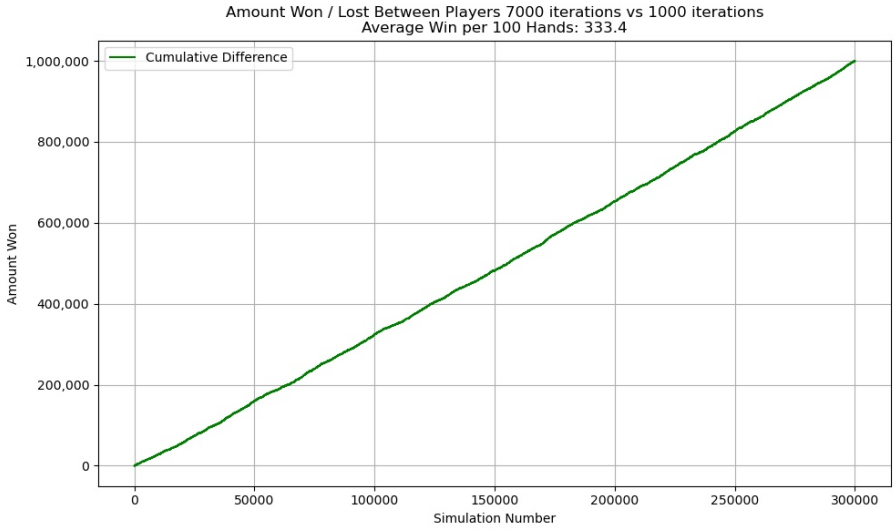


Figure 54: AI vs AI 7000 itrs vs 1000 itrs - Line Graph

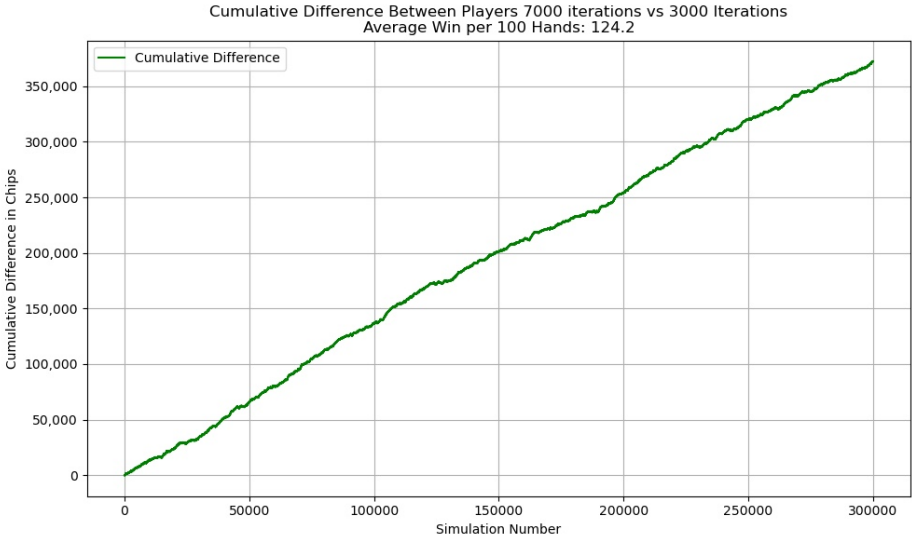


Figure 55: AI vs AI 7000 itrs vs 3000 itrs - Line Graph

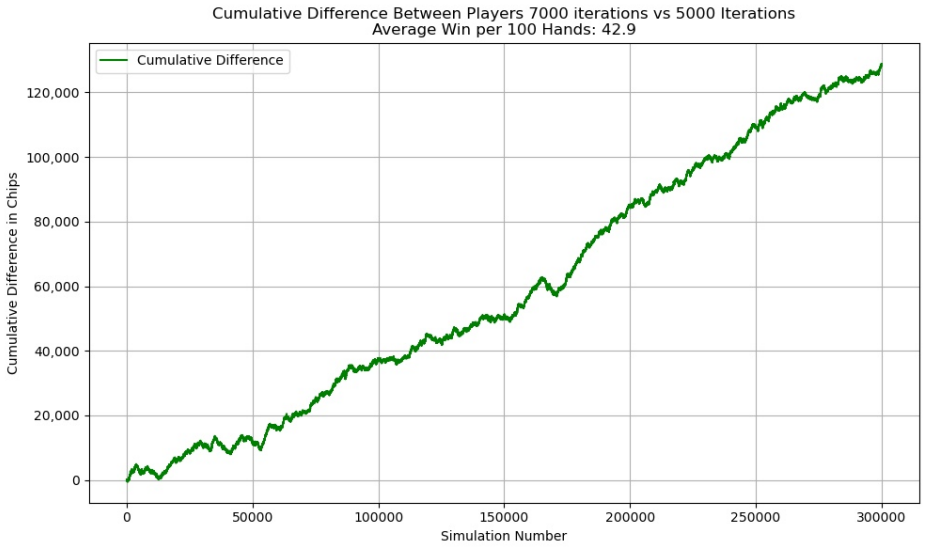


Figure 56: AI vs AI 7000 itrs vs 5000 itrs - Line Graph

The results illustrate an increase in the amount of chips won per 100 hands from the first test which was the 3,000 iteration to the 7,000 iterations AI. Each of these showed an extremely strong win rate, anything over 10 big blinds per 100m hands is considered very strong for online cash games, according to tightpoker.com(39).

The results of the 7,000 iteration AI decreased slightly as the opponent it was facing improved, when facing the 3,000 iteration AI, it still achieved a 6.21 big blinds won per 100 hands which I considered a strong win rate. That win rate decreased further to 2.145 when the AI faced the 5,000 iteration AI.

The incremental improvements observed as the iterations increase suggest that the AI is effectively learning and adapting its strategy over time. The reduced return on average winnings as the AI iterations become closer together indicates a more challenging competitive game as AIs become more advanced.

5.2.4.3 AI vs Human

This section looks at the performance of the AI which has been trained for 7,000 iterations against 2 human players. The test will be for 100 hands each, even though it's not exactly adequate to get a true representation of the performance of the AI, it allows the users to get a sense of the AI's tendencies and any specific strategic nuances they notice.

I requested both users leave a comment about the playing experience, mentioning the AI's performance and what they noticed about it, as well as the overall user experience when playing against the AI.

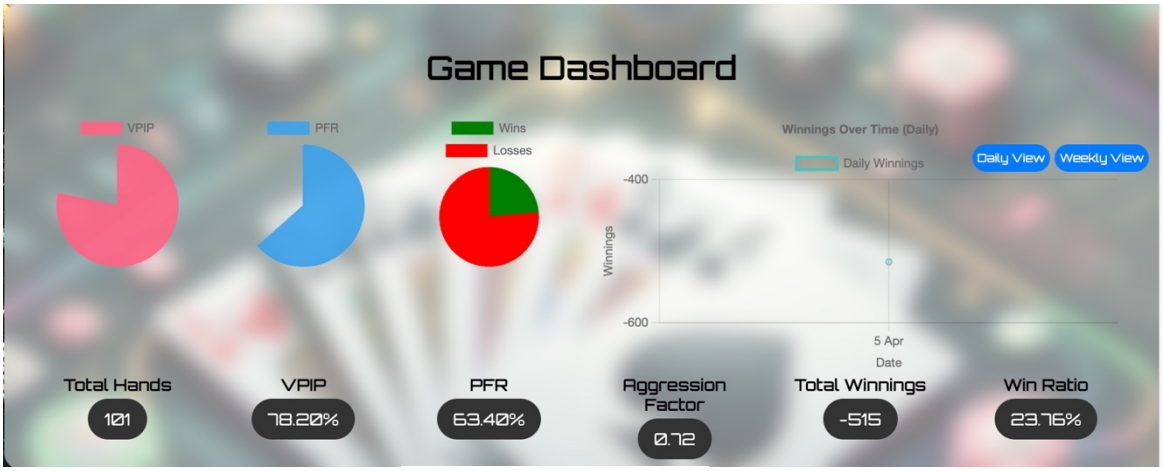


Figure 57: User 1 - Testing AI

The first user faired quite poorly against the, loosing 76.24% percent of hands and a total of 515 chips, in a game of 5/10. Which equals 51.5 big blinds lost in that 100 hands. Given the results of these 100 hands the AI has performed quite well.

According to the first user, "The opponent very rarely folded, and was extremely aggressive. There was definitely some spots it should have folded. I found it quite difficult to take advantage of the aggression at the start but eventually when I had a strong hand I was able to capitalise on the aggression. Unless you match the loose and aggressive it's hard not to get overwhelmed but I do think if I played another 100 hands I would be able to exploit the opponent and win quite big. I enjoyed the overall look of the app but I think it was a bit slow to start a new hand. "

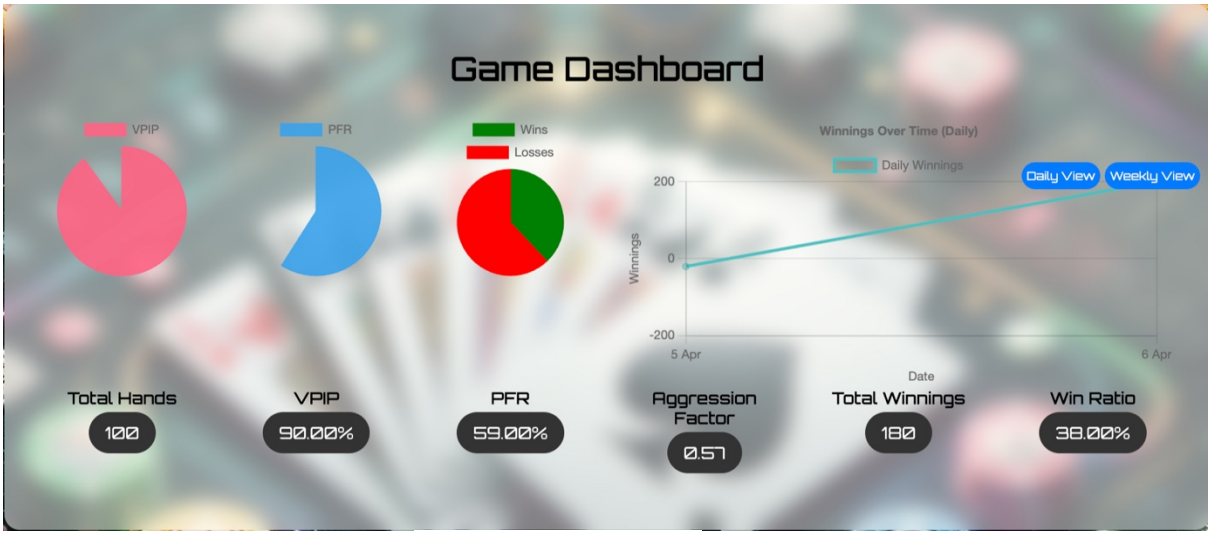


Figure 58: User 2 - Testing AI

Alternatively, the second user performed much better against the AI. He won a total of 180 chips, which results in 18 big blinds per 100 hands. The AI won more hands, winning 62% of hands, that is likely due to the AI unwillingness to fold.

This user stated that, "you have to play many more hands than usual and call more often in order to exploit the AI's aggressive nature.".

In conclusion, the AI agent has performed quite well and the reluctance to fold could be due to the exploration nature of the learning process, as it's only been trained for 7,000 iterations, the system has yet to fully exploit the opponents and figure out the best strategy so it's still exploring the possibility of each situation and raising and calling allows the AI agent to do that.

5.2.4.4 AI Decision Making

Figure 59: Pre-Flop Hand Decision Chart - Sixplusholdem

This sections compares the decision making process of each AI trained for a different number of iterations from 1000 to 7000. It will use two examples to show the evolving and progressing strategy as the amount of training increases. I will using the only available pre-flop percentage chart available for heads up short deck poker by Sixplusholdem (40), to compare and analyse the percentage of each version of the AI.

Firstly, I will look at J9 suited, the preflop percentages depicted in the chart serve as a probabilistic guide for the AI's actions, indicating the frequency with which certain moves should be made in a given situation. When the player has J9 suited it should raise 79% of the time, which suggests an aggressive playstyle, the hand is considered strong enough to raise the pot. However, a check is considered optimal 21% of the time, which indicates a degree of caution, possibly to disguise the strength of the hand. These percentages portray a strategic balance based on game theory optimisation, the slightly mixed strategy aim to make the player unexploitable by opponents.

Furthermore, by comparing the AI's various strategy against the game theory optimal strategy, it allows me to see how well the AI is learning and improving as the training increase.

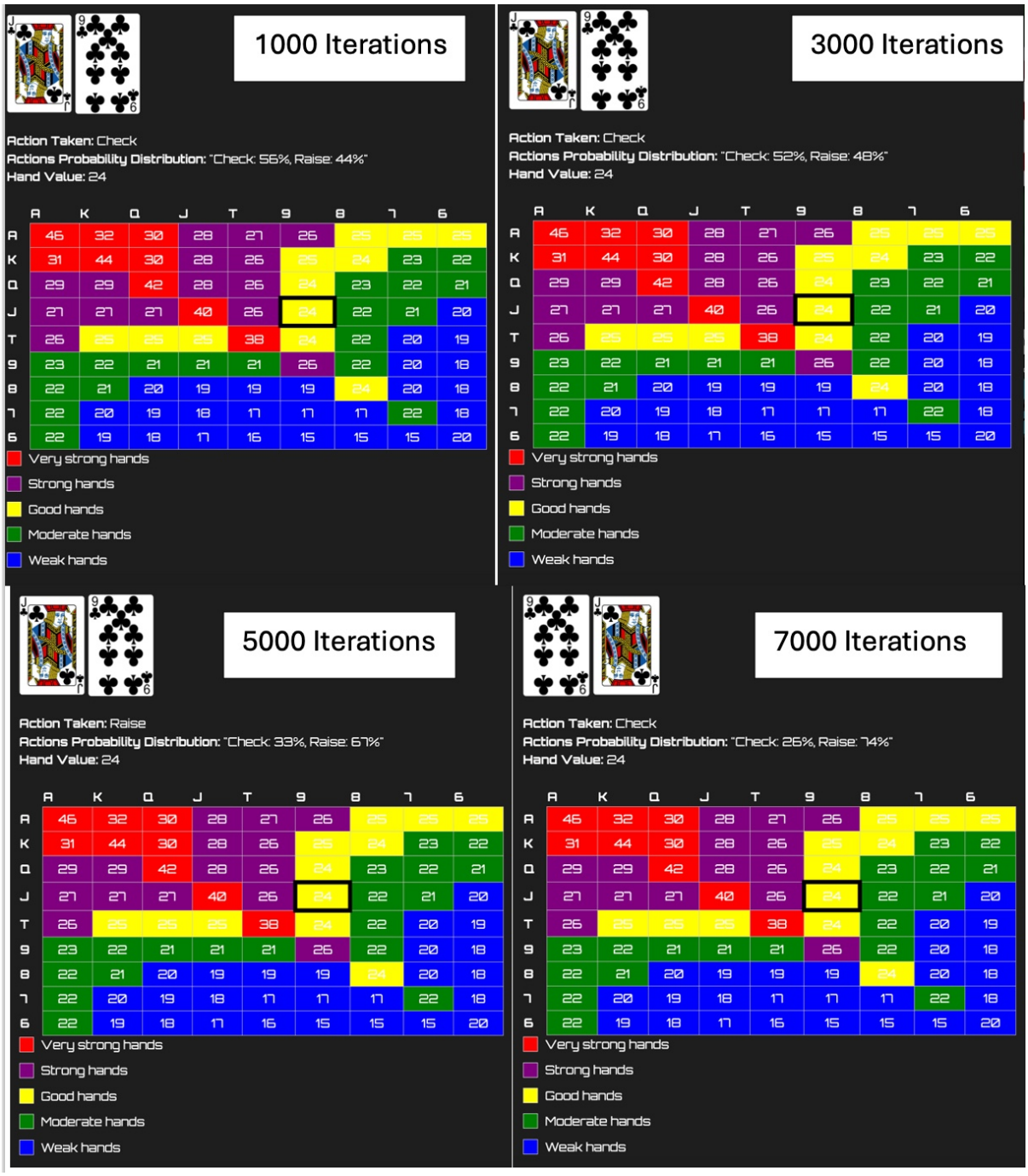


Figure 60: Application - Decision data w/ J9 for different AI's

As the number of training iterations increase the aggression with this given hand increases and converges closer to the optimal strategy for this given hand. When the final version of the trained AI (7000 iterations) is tested in this situation, it raises 74% of the time, which is in line with the optimal strategy. This displays the growth in the AI's sophistication and poker acumen. At the earlier stages of training it played with a much more mixed strategy in this instance.

Additionally, the instance of A9 off suit is treated with a more defensive approach as it's a weaker hand than the J9 suited. According to the chart by Sixplusholdem, the player should check with a 79.1% frequency and raise 19.9 % of the time. Even though the chart indicates a mainly defensive strategy, it still leaves some room for aggression which is necessary for a balanced playing style.

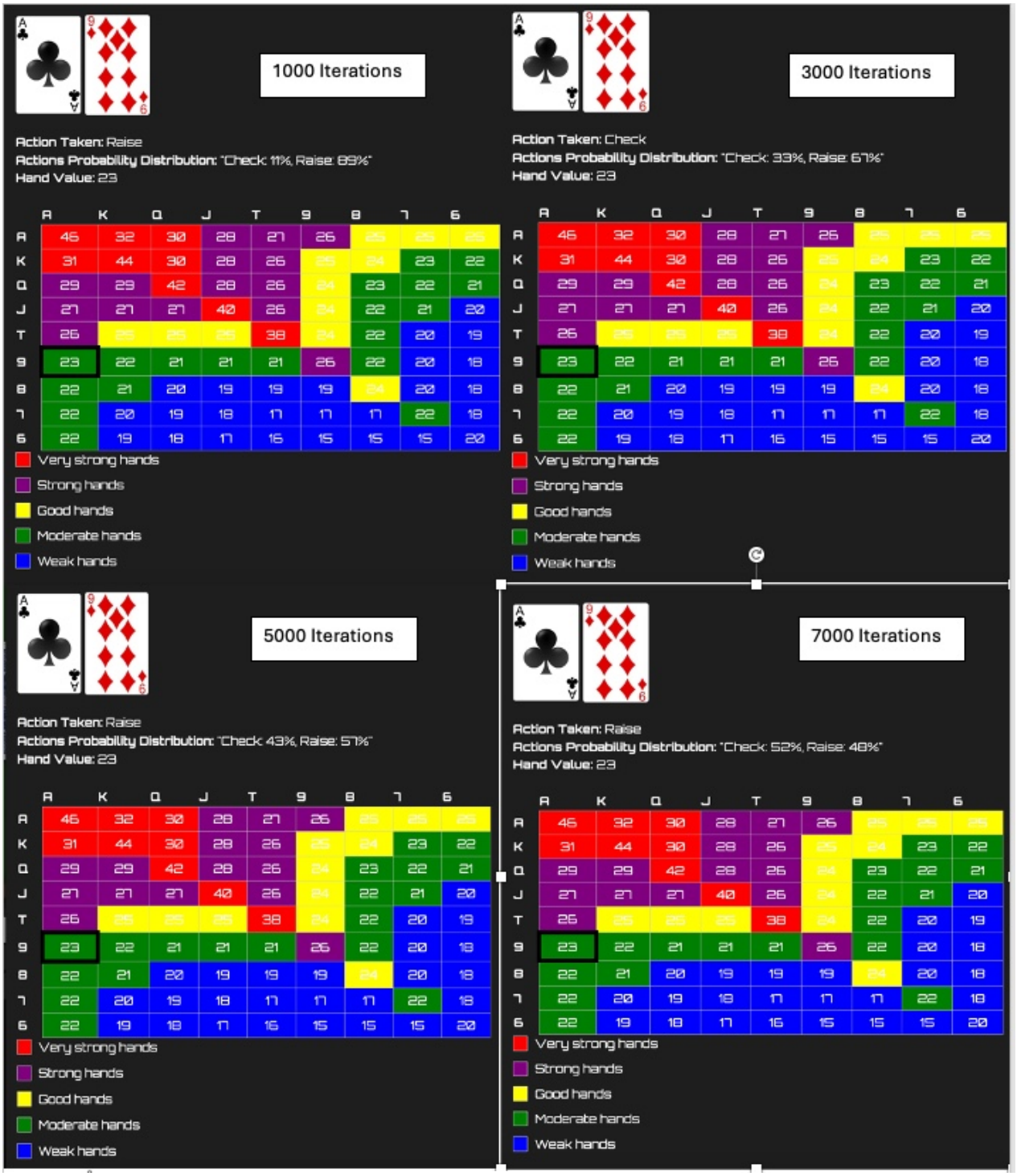


Figure 61: Application - Decision data for different AI's

Initially, this strategy isn't at all reflected by the AI, at its early stages of training, the approach is too aggressive. This is likely due to the AI's exploration and attempting to learn the best way to play a hand of this type. As the training increases the strategy begins to converge towards a more defensive strategy. Once the AI has been trained for 7000 iterations, the strategy is balanced, yet slightly defensive. I'm sure if the training would continue the it would continue to converge to the towards the game theory optimal strategy.
These two examples show the improvement in the AI's decision making as the number of training iterations increase, which was the goal of the AI agent.

5.3. System Evaluation

Throughout the development of the system an agile methodology was utilised, with iterative improvement being the priority. It was centred around completing tasks from the predefined feature list and having precise time targets and goals for the coming week or fortnight. The weekly supervisor meeting and Sunday evening check-in email provided a structure to my week which enabled me to stay on track with my progress.

Throughout the process, I've examined the white box testing results, ensuring that all the functions perform as intended, which is essential for the systems integrity. Once I was content with the white box testing results, I began the user testing process to obtain a true reflection of the systems functionality and potential flaws. I got two friends who have some experience with playing poker to firstly test the overall web application functionality and fill out the usability survey, and then play a number of hands against the AI to get a feel for its performance.

Furthermore, I spent a large amount of time tackling the slight speed issue I was having with the application. I mentioned a few of the changes I made in the speed performance testing section, these changes led to a roughly 0.9 second improvement in poker application page and a few tenths of a second improvement in the other pages due to the improvement in code quality and memory usage. Overall, I was satisfied with the improvements and the final page speeds that were recorded by the audit.

Lastly, the evaluation of the AI's performance an effective decision making is the most important aspect of the system. The testing process illustrates the continuous learning that the agent participates in. It's evident that the more training that the AI does the more effective the decision making and overall performance.

In conclusion, I would be confident this tool would be useful for poker players that are looking to improve their overall performance and understanding of the strengths of specific hands. This application provides players with a free test ground to improve their skills against an agent which has proven to make informed decisions that could challenge players in improving their skills.

5.4. Conclusions

Concluding this vital phase of testing and evaluation, the web application emerges more robust and user-friendly. While there's room for further improvement, especially in making complex statistics more accessible and the AI's decisions more transparent, I remain satisfied with the results and overall system functionality.

6. Conclusions and Future Work

6.1. Introduction

This chapter outlines the primary challenges faced during the development of this project, it also highlights the main achievements. Furthermore, it describes the overall project conclusion that was reached by completing this project. Additionally, I will outline the potential future work that could be done to enhance the system, outlining any features that could be added and what they would add to the system.

6.2. Challenges & Achievements

Throughout the development of this application I encountered a range of challenges, some of which I overcame and others I was unable to implement. The challenges which were overcame resulted in a great sense of personal achievement and now stand as an integral aspect of my project.

The primary difficulty I faced was extracting and displaying the necessary AI decision data for every action the AI takes within the hand review page. This wasn't a feature that was initially outlined in the interim report design but as I began building out the application, it became evident that describing this decision making process was integral for this application which is focused around training and education in the short deck poker variant.

Furthermore, I spent a large amount of time developing the same decision data for the users decisions and the consideration and thought process the AI has when faced with the users scenarios. I got this feature mostly working but at times the AI's suggested strategy would appear as equal for all actions, which would indicate that it hasn't encountered the hand before. This was due to the way the user actions were recorded in the database and the information sets the user experienced were invalid, therefore, at times the AI wasn't actually giving a recommendation to the user, so I was forced to remove this feature from the application. To get this feature working it would like require the functionality to be completely changed and considering I already spent just under a week trying to get this working I couldn't sacrifice more time to this feature.

Additionally, due to the limited computation power on hand and lack of training resources, I was only able to train the AI for 7000 iterations which was over 20 hours of training on my computer. This amount of training displayed evident progression in the AI's decision making and intelligence but with access to sufficient resources it would have been beneficial to train the AI for possibly 100,000 iterations and see where it would have gone. I believe the success of the AI agent considering the low number of training iterations is due to the hand abstraction employed and described in 4.2.2.1 Abstraction. I consider the development of the AI to be a strong achievement considering the limiting factor of insufficient computational power.

In concluding this section, the user experience emerges as the integral aspect of the application, and it's an aspect I have found significant difficulty when developing. I've dedicated considerable effort to ensure that players receive necessary feedback through intuitive animations and real-time interactions. Reducing the AI's response time and improving the loading time of the poker game was a significant challenge which I spent a considerable amount of time improving but it could still do with some improvements in future iterations.

6.3. Gannt Chart

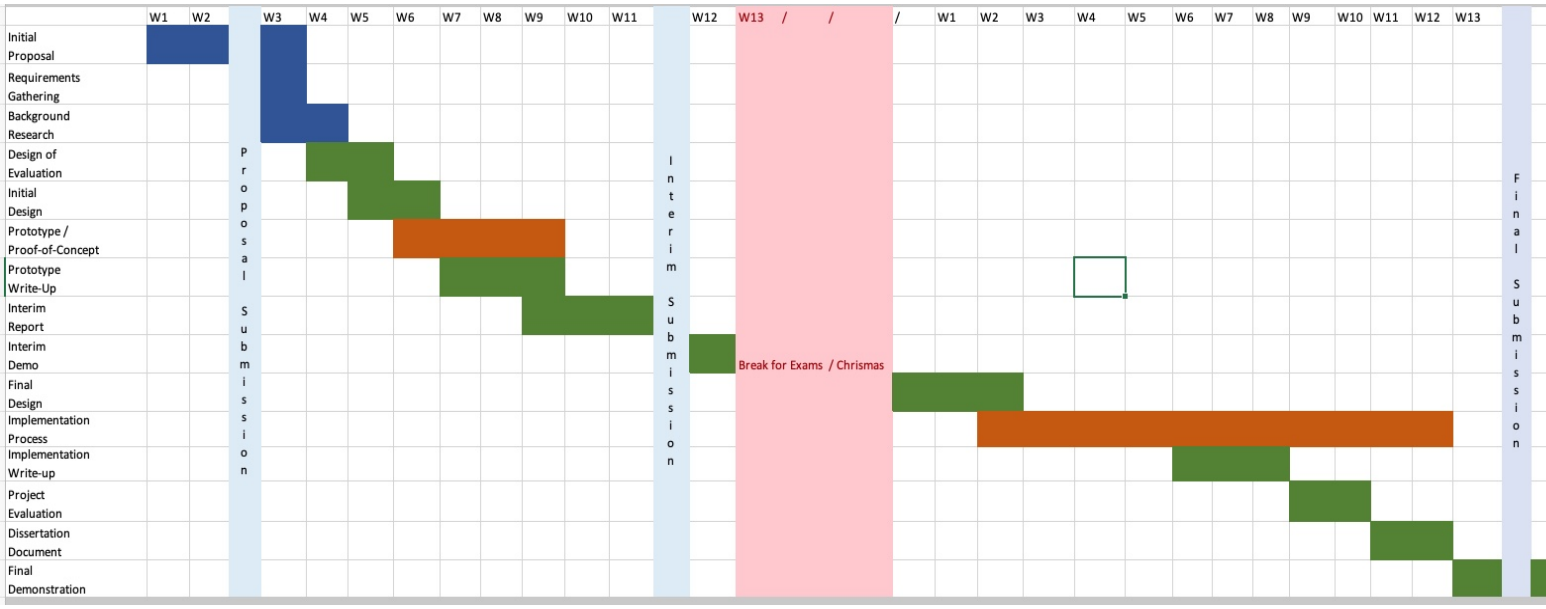


Figure 62: Gannt Chart

6.4. Future Work

Looking ahead, there are many additions that could be made to the application to elevate the overall functionality and performance. A big improvement would be to transition from a local environment to a server-based system, which would result in enhanced performance and a more scalable platform. This shift will enable the application to be utilised by poker players around the world who wish to elevate their skills by playing against the AI in this training environment and educate themselves using the AI's decision making data.

In an effort to improve the AI, I aim to train the agent on the cloud, which will enable me to utilise the significant computational power available. However, using this power does come at a cost, for example, considering AWS as the cloud provider and using the 'p3 3xlarge' instance to train the AI, which costs $3.06 per hour, given the previous training completed, takes approximately 10 seconds per iteration and training for 100,000 iterations will cost around $850. This is a significant one off payment that would have to be made to make the AI intelligent enough to produce as a highly effective educational poker training ground for players.

Furthermore, there would be a lot less hand abstraction used and maybe even none at all depending on the effectiveness of the training. Rather than having 5 pre-flop hand buckets, I would try to create 20 buckets, where all the hands inside the buckets employ the exact same strategy.

Additionally, getting the user decision making consideration mentioned in the previous section working still remains a priority for the application. By achieving a fully functional and accurate representation of the AI's thought process in user scenarios, it can offer a more informative and educational experience. This feature's potential to enhance learning and strategic understanding is immense. It will require a significant change in the structure of the

poker application and how the data is handled and recorded and at what point the AI makes its decision.

6.5. Conclusions

Over the course of this project, I developed a poker web application which involved many complex features such as the hand review page, the game dashboard and the overall poker game. It enables the users to create an account, customise their profile, track their progress, analyse their previously played hands and the decision making process which led to the AI's decisions and finally play Short Deck Poker against the AI agent.

The agile based methodology which is a combination of feature driven development and scrum was utilised as the projects design methodology, this enabled a focused and time efficient approach to execute the development. The application was rigorously tested using a range of testing methods to evaluate the performance of the application and the AI agent. White box testing was used to test the functionality, usability testing was employed to consider the user feedback, speed performance testing to identify how the application functionally performs and lastly, the AI agent testing was done to verify the improvement in the AI's performance as the amount of training increased.

In conclusion, this application perfectly reflects my original design, it has progressed and blossomed from the development of the original prototype. In the end, an application was developed which tailors for poker players who wish to improve their game in a fun environment which promotes education and learning.

Bibliography

1. D'Agostino S. DeepMind's David Silver on games, beauty, and AI's potential to avert human-made disasters [Internet]. Bulletin of the Atomic Scientists. 2022 [cited 2023 Nov 30]. Available from: https://thebulletin.org/2022/01/deepminds-david-silver-on-games-beauty-and-ais-potential-to-avert-human-made-disasters/

2. Corporate Finance Institute [Internet]. [cited 2023 Dec 7]. Nash Equilibrium. Available from: https://corporatefinanceinstitute.com/resources/economics/nash-equilibrium-game-theory/

3. MIT. https://pokerbots.org/ [Internet]. PokerBots. Available from: https://pokerbots.org/

4. David Silver [Internet]. [cited 2023 Oct 19]. Teaching. Available from: https://www.davidsilver.uk/teaching/

5. Brown N, Sandholm T. Superhuman AI for multiplayer poker. Science. 2019 Aug 30;365(6456):885–90.

6. Billings D, Papp D, Schaeffer J, Szafron D. Opponent Modeling in Poker. In 1998. p. 493–9.

7. Billings D, Burch N, Davidson A, Holte R, Schaeffer J, Schauenberg T, et al. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker.

8. Zinkevich M, Bowling M, Johanson M, Piccione C. Regret Minimization in Games with Incomplete Information.

9. Bowling M, Burch N, Johanson M, Tammelin O. Heads-up Limit Hold'em Poker is Solved.

10. Neller TW, Lanctot M. An Introduction to Counterfactual Regret Minimization. 2013;

11. ArmanMielke/simple-poker-cfr: Counterfactual Regret Minimization for a simplified version of Texas Hold'em poker [Internet]. [cited 2023 Oct 20]. Available from: https://github.com/ArmanMielke/simple-poker-cfr/tree/master

12. Zinkevich M, Johanson M, Bowling M, Piccione C. Regret Minimization in Games with Incomplete Information. In: Advances in Neural Information Processing Systems [Internet]. Curran Associates, Inc.; 2007 [cited 2024 Apr 6]. Available from: https://proceedings.neurips.cc/paper/2007/hash/08d98638c6fcd194a4b1e6992063e944-Abstract.html

13. Online Poker | Play The Worlds Biggest Poker Room at GGPoker [Internet]. [cited 2023 Dec 7]. Available from: https://ggpoker.com/

14. SmartHand - Poker stats service [Internet]. [cited 2024 Mar 25]. Available from: https://smarthand.pro/en

15. ICMIZER Replayer: Beautiful & Easy-to-use Hand History Replayer [Internet]. [cited 2024 Mar 25]. Available from: https://www.icmizer.com/icmizer/replayer-features/

16. Poker Cheat Sheet: Preflop Charts for All Disciplines [Internet]. [cited 2024 Mar 25]. Available from: https://cardmates.org/chart_of_poker_starting_hands

17. Python Language advantages and applications [Internet]. GeeksforGeeks. 2017 [cited 2023 Nov 16]. Available from: https://www.geeksforgeeks.org/python-language-advantages-applications/

18. Advantages and Disadvantages of C++ [Internet]. GeeksforGeeks. 2020 [cited 2023 Nov 16]. Available from: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-cpp/

19. McKerns M. dill: serialize all of Python [Internet]. [cited 2024 Mar 25]. Available from: https://github.com/uqfoundation/dill

20. NumPy - [Internet]. [cited 2024 Mar 25]. Available from: https://numpy.org/

21. Python documentation [Internet]. [cited 2024 Mar 25]. itertools — Functions creating iterators for efficient looping. Available from: https://docs.python.org/3/library/itertools.html

22. Learning G. Great Learning Blog: Free Resources what Matters to shape your Career! 2022 [cited 2023 Nov 16]. Flask Vs Django: Which Python Framework to Choose? Available from: https://www.mygreatlearning.com/blog/flask-vs-django/

23. What is PostgreSQL? Introduction, Advantages & Disadvantages [Internet]. 2023 [cited 2023 Nov 16]. Available from: https://www.guru99.com/introduction-postgresql.html

24. HTML Tutorial [Internet]. [cited 2024 Apr 11]. Available from: https://www.w3schools.com/html/

25. CSS. In: Wikipedia [Internet]. 2024 [cited 2024 Apr 11]. Available from: https://en.wikipedia.org/w/index.php?title=CSS&oldid=1217996434

26. JavaScript. In: Wikipedia [Internet]. 2024 [cited 2024 Apr 11]. Available from: https://en.wikipedia.org/w/index.php?title=JavaScript&oldid=1217164265

27. Chart.js [Internet]. [cited 2024 Apr 11]. Available from: https://www.chartjs.org/

28. SmartRoutes [Internet]. 2022 [cited 2023 Nov 16]. Route optimization for solid waste collection. Available from: https://smartroutes.io/blogs/route-optimization-for-solid-waste-collection/

29. Xia W, Jiang Y, Chen X, Zhao R. Application of machine learning algorithms in municipal solid waste management: A mini review. Waste Manag Res. 2022 Jun 1;40(6):609–24.

30. O'Neill S. MindGame: A Machine Learning Framework for Electronic Games Final Year Project Report.

31. Technological University Dublin Library [Internet]. [cited 2024 Apr 6]. Available from: https://library.tudublin.ie/search?/Xandrew+mccormack&SORT=D/Xandrew+mccormack&SORT=D&SUBKEY=andrew+mccormack/1%2C7%2C7%2CB/frameset&FF=Xandrew+mccormack&SORT=D&2%2C2%2C

32. Singh R. Institute of Project Management. 2022 [cited 2023 Nov 16]. Waterfall Methodology. Available from: https://www.projectmanagement.ie/blog/waterfall-methodology/

33. What is Scrum? | Scrum.org [Internet]. [cited 2023 Nov 16]. Available from: https://www.scrum.org/resources/what-scrum-module

34. Planview [Internet]. [cited 2023 Nov 16]. What is FDD in Agile? Available from: https://www.planview.com/resources/articles/fdd-agile/

35. Dall-e Free [Internet]. [cited 2024 Mar 27]. DALL-E Free AI Image Generator | Powered by Dall-E. Available from: https://www.dall-efree.com

36. ExePokerWeb/playPoker at main · Doglum/ExePokerWeb [Internet]. [cited 2024 Apr 6]. Available from: https://github.com/Doglum/ExePokerWeb/tree/main/playPoker

37. GeeksforGeeks [Internet]. 2018 [cited 2024 Apr 1]. White box Testing - Software Engineering. Available from: https://www.geeksforgeeks.org/software-engineering-white-box-testing/

38. Chrome for Developers [Internet]. [cited 2024 Apr 1]. Overview | Lighthouse. Available from: https://developer.chrome.com/docs/lighthouse/overview

39. Seaton P. Tight Poker. 2023 [cited 2024 Apr 5]. What Are the Challenges of Maintaining a Consistent Win Rate in Poker? Available from: https://www.tightpoker.com/what-are-the-challenges-of-maintaining-a-consistent-win-rate-in-poker/

40. GTO Heads-up Preflop Charts | Six Plus Hold'em [Internet]. 2021 [cited 2024 Mar 30]. Available from: https://sixplusholdem.com/shortdeck-six-plus-gto-headsup-hand-charts/

Appendices

User survey Responses

Q1: Was it easy to create register and login to your account?
easy
easy

Q2: How did you find navigating the application?
Easy
Very easy

Q3: Did you find the application aesthetically pleasing?
Yes, Very pleasing
Kind of

Q4: Were you able to easily change your avatar?
Yes, I worked it out
yes, very straightforward

Q5: Did the find the game statistics easy to understand?
Yes, I understand was most of it means
Yes, I understand was most of it means

Q6: Did the AI's decsion making process in the hand review page provide useful information?
Kind of
Yes

Q7: Was playing poker against the AI agent intuitive, did you receive the necessary feedback from the system?
No, I was left guessing at times
Yes, it was very intuitive to play against