Reinforcement Learning for Stochastic Games

Sheiladitya Kumar, B.Sc

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Future Networked Systems)

Supervisor: John Waldron

August 2022

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Sheiladitya Kumar

August 19, 2022

Permission to Lend and/or Copy

Sheiladitya Kumar

August 19, 2022

Reinforcement Learning for Stochastic Games

Sheiladitya Kumar, Master of Science in Computer Science

University of Dublin, Trinity College, 2022

Supervisor: John Waldron

Over the last few years there has been resurgence in the popularity of stochastic games. This can be seen with the growing popularity of some of these games such as poker and chess experiencing large growth. In this dissertation we look to evaluate a stochastic game, blackjack, in depth through the use of reinforcement learning (RL) methods such as value iteration and policy extraction. The project provides a thorough overview, covering the required theory and then implementing it to see how well gameplay within the environment of such a stochastic game can be optimized.

Acknowledgments

I would like to extend my thanks to my advisor, John Waldron, for his guidance and assistance. I would also like to thank my parents and sister for their never ending support throughout my academic career.

Sheiladitya Kumar

University of Dublin, Trinity College
August 2022

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Reinforcement Learning

Reinforcement learning is a large expanse of topics that covers many methods of model free and model based learning, usually in an attempt to optimize solutions for environments in the real world. It has shown prior success in many fields such as but not limited to developing agents for trading in competitive markets such as Forex and the stock market [1; 2]. Video game environments, such as in the works of [3; 4]. Reinforcement Learning (RL) algorithms and practices have been a staple in environment simulation and decision optimization problems for a very long time, however the recent boom in Machine learning have brought back an interest in both classical reinforcement learning methods as well as novel ideas that try and optimize both.

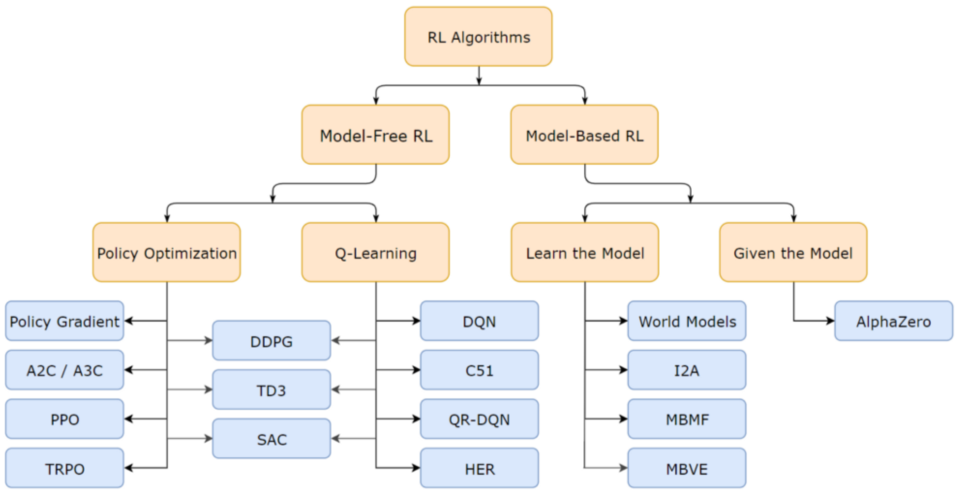

Figure 1.1: A broad overview of the various fields within Reinforcement Learning (RL) Source: https://miro.medium.com/max/1400/1*BsN4a2N1EDmgG19wWDd9CQ.png

1.2 Motivation

Our objective in undertaking this project is to better understand classical Reinforcement Learning methods, in particular, this dissertation will heavily focus on Markov Decision Processes and Value Iteration algorithms to understand and optimize solutions for an environment simulating Blackjack. The motivation behind choosing Blackjack, also known as 21, is due to its popularity and because it provides a low barrier of entry thereby allowing us to spend the majority of our time in trying to optimize for its well understood states and solutions. The goals of this dissertation are as follows:

- Conduct a heavy study on Markov Decision Processes and understand their algorithms both from a mathematical and computer science perspective.
- Research the value iteration algorithm particularly well and describe in detail the process during this project.
- Define an MDP environment for simulating Blackjack.
- Perform value iteration in a blackjack environment to develop optimal strategies and evaluate its success.

1.3 Flow of Project

In this section we describe how the rest of this project has been organized.

Chapter 2: State of the Art

In the state of the art we deeply explore Markov Decision Processes, we develop an understanding through precise use of the mathematics behind MDP's as well as providing algorithms for them.

Chapter 3: Design

In the Design, we explain the variation of Blackjack we have chosen for this project, we then explore how blackjack can be simulated as an MDP.

Chapter 4: Implementation

In the Implementation we try and explain some of the code behind how we have implemented the Markov Decision Processes.

Chapter 5: Evaluation

In the Evaluation, we provide a deep overview of the results of our experiments as well as our detailed results showing how well our models were optimized.

Chapter 6: Conclusion and Future Works

Finally in the conclusion, we provide some some thoughts on this dissertation as a whole and discuss how well we achieved our goals as well as providing some ideas for how these works could have been extended in the future.

Chapter 2

State of the Art

In this chapter we will introduce the Markov Decision Processes (MDP) a core Reinforcement Learning (RL) concept that will be crucial to our exploration in this report. We will explore the workings of a Markov Decision Process in depth while also covering other Reinforcement Learning concepts and drawbacks. At the end of the chapter we will provide a review of some of the closely related Literature for our topic.

2.1 Markov Decision Processes

Markov Decision Processes are an extension of Markov Chains introduced by Andrey Markov [5], they differ from one another since Markov Decision Processes require both actions and rewards as well, in the absence of those two factors, Markov Decision Processes can be reduced to Markov Chains. Markov Decision Processes help us model stochastic environments which consist of a finite number of states, actions, and rewards for those states and actions. With the use of Markov Decision Processes we can find optimal strategies/policies and find the optimal action to take in any given scenario within that environment.

2.1.1 Defining an MDP

To define any Markov Decision Process we require a set of parameters that will define the MDP. First, we naturally require an environment or game we are trying to model, we can then define **S** to be the finite set of all states that the environment/game can be in. We can also define **A** to be the finite set of all actions that an agent acting in the environment can take. Thus, when we set up an MDP we will have an agent which will act on the environment for us, the agent acts on the environment at time t by taking some action $a_t \in \mathbf{A}$ while the environment is in some state $s_t \in \mathbf{S}$ which causes the environment to

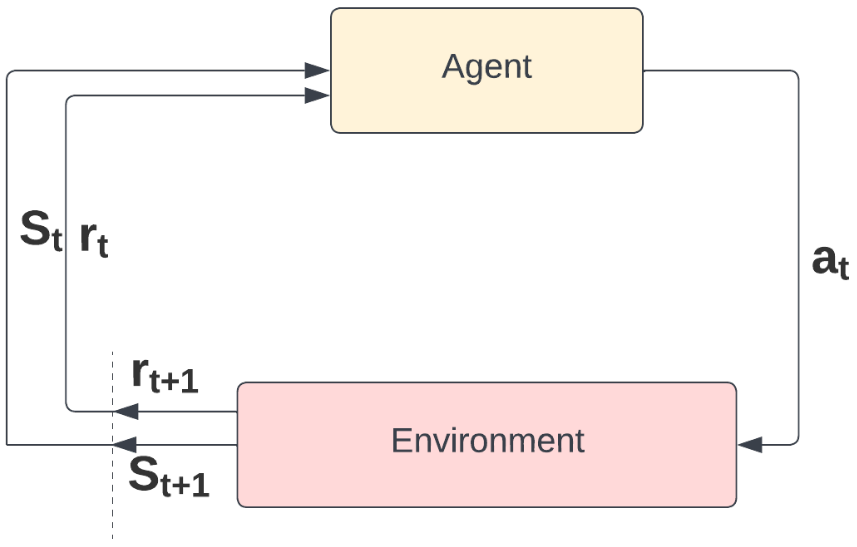

Figure 2.1: An Agents interaction with an Environment in a Markov Decision Process

transition to some state $s_{t+1} \in \mathbf{S}$ thereby causing a change in the environment. The agent continues to act on the environment by taking actions and moving to a new state until a predetermined time or until it reaches a terminal state. An overview of the operation of a Markov Decision Process is shown in Fig. 2.1. You may also notice that Fig. 2.1 contains the symbol r_t and r_{t+1}, this symbol corresponds to the rewards at time t and $t + 1$ respectively and we will elaborate on it further in the following sections.

2.1.2 Transitions

As shown in Fig. 2.1, after an agent acts on an environment in state s_t the environment transitions to state s_{t+1}, please note that states s_t and s_{t+1} can be the same state. To define the chances of transitioning between states after an action let us define $s, s' \in \mathbf{S}$, once again we have $a \in \mathbf{A}$, we can then define $T(s, a, s')$ to be the transition function/probability that returns the probability of transitioning from state s to state s' after an agent takes action a, as can be seen by Eqn. 2.1. As can be seen by this, transitioning from one state to another depends only on the previous state and the action taken. Thus we have Eqn. 2.2, which surmises the fact that probability of transitioning to any next state depends only on the current state and action.

$$T(s, a, s') = P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t) \tag{2.1}$$

$$\begin{aligned}P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, ..., S_0 = s_0) \\ = P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t)\end{aligned} \quad (2.2)$$

2.1.3 Rewards

Any Markov Decision Process also requires some motivation for the agent to make decisions in an informed sense. To achieve this aim we also have rewards for transitioning between states. Let us take $s, s' \in \mathbf{S}$, $a \in \mathbf{A}$, we can then define $R(s, a, s')$ to be the reward function for transitioning from state s to state s' by performing actions a.

2.1.4 Objective

An agent acts in an environment with a specific objective, and that objective is usually to maximise the reward over some time period. In this manner any action that yields a positive reward within that time period will be considered a positive action, similarly any action that yields a negative reward will yield a negative action. The objective of the agent is usually to find the most optimal path within that time period in order to grant the highest reward possible.

2.1.5 Discount Factor

There is one final parameter an agent must consider while it acts on an environment or game, the discount factor, $\gamma \in [0, 1]$. The discount factor is applied to each reward in the time step thereby decreasing the utility of awards at larger time steps. By introducing a discount factor we can motivate the agent to take actions that are beneficial now as opposed to indefinitely in the future. Having a high discount factor, usually $\gamma \geq 0.9$, will motivate the agent to think about the future, also known as a long horizon. Whereas, a smaller discount factor will motivate an agent to act in the present and take the action that will generate the highest reward currently, also known as a short horizon.

2.2 Solving Markov Decision Processes

So far we have established that a Markov Decision process can be defined by a 5 element tuple, $< \mathbf{S}, \mathbf{A}, T(s, a, s'), R(s, a, s'), \gamma >$. Where $\mathbf{S}$ is the state space, $\mathbf{A}$ is the action space, $T(s, a, s')$ defines the transition probability, $R(s, a, s')$ is the reward function, and γ is the discount factor. An agent takes account of all these parameters and acts upon an environment thereby causing changes to the state of the environment and receives rewards for the actions it takes. We will now explore how an agent can begin to act optimally, primarily we will discuss how we can find optimal actions for every state, or an optimal policy.

2.2.1 Policies and Value Functions

A policy, often denoted as π, can be referred to as a map from the state space to the action space, $\pi : \mathbf{S} \longrightarrow \mathbf{A}$. That is, a policy instructs an agent on which action to given any state $\pi(s) = a \ \forall s \in \mathbf{S}, a \in \mathbf{A}$. We can then define the value function $V_\pi(s)$ which returns the expected utility of starting in state s and then following policy π. $V_\pi(s)$ is defined in Eqn. 2.3, as can be seen, the value function is defined recursively and essentially provides the expected value of the discounted rewards received from following policy π.

$$V_\pi(s) = \sum_{s'} T(s, \pi(s), s')\big(R(s, \pi(s), s') + \gamma V_\pi(s')\big) \tag{2.3}$$

Similarly, we can define the Q-Value function, $Q_\pi(s, a)$ as shown in Eqn. 2.4. The difference between the value function, 2.3, and the Q-Value function, 2.4, is that the Q-Value function returns the expected reward of starting in state s, **taking action** a and then following policy π.

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')\big(R(s, a, s') + \gamma V_\pi(s')\big) \tag{2.4}$$

2.2.2 Optimal Solutions

Our objective in defining an MDP in the first place is to find the optimal solution to problems, in that case, let us define π, π' to be two distinct policies. We can then order the policies based on their value function, Eqn. 2.5.

$$\pi \geq \pi' \text{ if } V_\pi(s) \geq V_{\pi'}(s) \ \forall s \in \mathbf{S} \tag{2.5}$$

This ordering of policies then allows us to define an optimal policy, π_* which is greater than or equal to all other policies, Eqn. 2.6

$$\pi_* \geq \pi \ \forall \pi \tag{2.6}$$

We can then go back to the value function and assert that any value function that follows the optimal policy must be an optimal value function, Eqn. 2.7.

$$V_*(s) = V_{\pi_*}(s) = \max_{\pi} V_\pi(s) \tag{2.7}$$

Now, if we recall that the optimal policy is obtained from following the actions that maximizes the reward of the value function we get the Bellman Optimal Equation, Eqn. 2.8.

$$V_*(s) = \max_a \sum_{s'} T(s,a,s')\big(R(s,a,s') + \gamma V_*(s')\big) = \max_a Q_*(s,a) \tag{2.8}$$

We can also extract an optimal policy by optimizing over $a \in \mathbf{A}$ for $Q_*(s,a)$, Eqn. 2.9.

$$\pi_*(s) = \arg\max_a Q_*(s,a) \tag{2.9}$$

2.3 Iterative Methods

Now that we have introduced the Bellman Equation, Eqn. 2.8 and discussed how an optimal policy can be extracted, Eqn. 2.9 we can introduce the primary iterative methods of arriving at these optimal solutions.

2.3.1 Value Iteration

Value iteration [6] is an recursive method, in which we modify the Bellman Optimal Equation, Eqn. 2.8, until the expected values from the iteration converge for all known states. At this point we can ascertain that we have found an optimal solution for the environment. The value iteration formula is slightly modified, Eqn. 2.10, from the Bellman Equation.

$$V_{k+1}(s) \longleftarrow \max_a \sum_{s'} T(s,a,s')\big[R(s,a,s') + \gamma V_k(s')\big],\ V_0(s) = 0\ \forall s \in \mathbf{S} \tag{2.10}$$

A step-by-step method to conducting value iteration is defined in Algorithm 1.

2.3.2 Policy Extraction

Policy extraction actually follows simply from Value Iteration, we can even perform both in the same convergence and this is shown in Algorithm 2. It should be noted that the policy might converge well before the values themselves converge.

Algorithm 1: Algorithm for value iteration until convergence.

```
   Input  : $\mathbf{S}$, $\mathbf{A}$, $T(s,a,s'), R(s,a,s')\ \forall s,s' \in \mathbf{S},\ a \in \mathbf{A},\ \gamma \in [0,1]$
   Output: $V_*(s)\ \forall s \in \mathbf{S}$
1  $V_0(s) \longleftarrow 0, \forall s \in \mathbf{S}$ ; /* Initialize $V_0(s)$ to be 0 for all possible
     states within the state space                                        */
2  $k \longleftarrow 0$ ; /* Set a variable k to count number of iterations       */
3  while $V_{k+1}(s), \forall s \in \mathbf{S}$ has not converged do
4      $V_{k+1} \longleftarrow 0$ ; /* Set the next iteration to 0, so it can be updated
         during the loop                                                  */
5      foreach $s \in \mathbf{S}$ do
6          foreach $a \in \mathbf{A}$ do
7              $V_{\pi_a}(s) \longleftarrow T(s,a,s')\big(R(s,a,s') + \gamma V_k(s)\big)$ ; /* Calculate the
                 expected utility for this state and action                */
8          end
9          $V_{k+1}(s) \longleftarrow \max(V_\pi(s))$ ; /* Set the next value iteration value to
             be the highest value found from all the available actions in
             the current state                                            */
10     end
11 end
```

2.3.3 Q-Learning

Q-Learning [7] provides another way for agents to act optimally within the environment of a Markov Decision Process. The Q-learning method introduces a new parameter, $\alpha \in (0,1]$, which is the learning rate. This is also an iterative method that works as shown in Eqn. 2.11.

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\big(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\big) \tag{2.11}$$

2.3.4 Challenges with Iterative Methods

There are a number of challenges that hinder Iterative methods and MDP's in general difficult at times.

High Cost

Usually, both the time and space complexity of an algorithm such as Value iteration can be very costly. Especially if one doesn't make attempts to reduce the State and/or action space.

Algorithm 2: Algorithm for value iteration and policy extraction.

```
   Input  : $\mathbf{S}$, $\mathbf{A}$, $T(s,a,s'), R(s,a,s')\ \forall s,s' \in \mathbf{S}$, $a \in \mathbf{A}$, $\gamma \in [0,1]$
   Output: $V_*(s), \pi_*(s)\ \forall s \in \mathbf{S}$
 1 $V_0(s) \longleftarrow 0, \forall s \in \mathbf{S}$ ; /* Initialize $V_0(s)$ to be 0 for all possible
    states within the state space                                        */
 2 $k \longleftarrow 0$ ; /* Set a variable k to count number of iterations        */
 3 while $V_{k+1}(s), \forall s \in \mathbf{S}$ has not converged do
 4     $V_{k+1} \longleftarrow 0$ ; /* Set the next iteration to 0, so it can be updated
        during the loop                                                   */
 5     foreach $s \in \mathbf{S}$ do
 6         foreach $a \in \mathbf{A}$ do
 7             $V_{\pi_a}(s) \longleftarrow T(s,a,s')\big(R(s,a,s') + \gamma V_k(s)\big)$ ; /* Calculate the
                expected utility for this state and action                */
 8         end
 9         $V_{k+1}(s) \longleftarrow \max(V_\pi(s))$ ; /* Set the next value iteration value to
            be the highest value found from all the available actions in
            the current state                                             */
10         $\pi_*(s) \longleftarrow \arg\max_\pi V_\pi(s)$
11     end
12 end
```

Difficult Implementation

In real world tasks, such as game theory or real world scenarios such as optimizing traffic, the transition probabilities and/or frequencies are not easily deducible and we often have to rely on Monte Carlo methods to simulate the situation and then extrapolate a probability density or function from those simulations. While these methods are quite reliable in terms of their accuracy to the real world state, they do pose an even higher cost on an already costly process.

2.4 Related Works

2.5 Works Utilizing Markov Decision Processes

There are a large number of works focusing on Markov Chains and Markov Decision Processes. They are often used to investigate real world phenomenon such as weather forecasting, stock prediction, and of course game theory optimization. Works such as[8] explore the uses of Markovian processes for convertible bonds, [9] also consider Markovian processes in different markets. Others like [10] discuss and associative criteria for MDMDP's (Mutually Dependent Markov Decision Processes). Moreover, [11], consider how well MDP's work in uncertain environments. The works of [12] consider some of the drawbacks of MDP's and try to propose a new look at improving their efficieny through the use of alternating decision trees (AD Trees). Even military and rescue interests have found MDP's useful as [13] tries to use Markov Decision Problems to better understand search and rescue tasks. Farming practices can also benefit from MDP's as demonstrated by the works of [14]. They have been clearly demonstrated to be useful in other fields such as cyber security [15; 16], amongst many other works.

2.6 Works Exploring Stochastic Games and Environments

Stochastic games are also a large number of works specializing in the solving of Stochastic Games such as chess [17; 18; 19; 20; 21], discrete time games and environments with stochastic properties such as [22; 23; 24; 25; 26]. There are also a large number of works which specialize specifically on Blackjack as has been done in this project as well [27; 28; 29]. Of these works, a number rely on Markov Processes, while some use other RL methodologies and a few present ideas using neural networks and gradient based machine learning.

Chapter 3

Design

3.1 Blackjack

Blackjack, sometimes also referred to, is the chosen stochastic game we will try to optimize for with our Reinforcement Learning (RL) agent. Blackjack is a very popular casino game played throughout the world, usually with multiple regular decks of playing cards, one may expect to know the rules of the game but due to the variations in play depending on a country or casino, We have decided to highlight the rules of the game we will be using for the experiments in this project.

3.1.1 Rules

Card Values

The total score of a player and dealer is decided by the value of the cards they hold, the numbered cards 2-10 inclusive retain their numbered value. The face cards - Jack (J), Queen (Q), King (K) - are all given a value of 10. The Ace (A) is a special card which can take a value of either 1 or 11 depending on the situation of the game.

Objective

The objective of blackjack player is to have as high a score as possible using two or more cards, however the score cannot exceed 21. If the combined score of a players hand is higher than 21, then the player is considered to have gone bust and the player loses that particular hand.

Gameplay

The gameplay of blackjack is quite straightforward, the game consists of one more players and a dealer, for the purposes of this project, we will be dealing with one player, so we will refer to a single player and agent henceforth. Then, the player will continue to perform actions until he chooses to take no more actions, referred to as standing, or until he busts. Assuming the player does not bust, the dealer will then continue to perform actions until the dealer is forced to stand or until he busts. Between the dealer and the player, the highest score wins. If the dealer wins, then the player loses his bet. If the player wins, then the player receives twice the value of his wager, this includes the original wager.

Rules for the Dealer

The dealer must wait until the player is either bust, in which case the dealer wins by the default, or until the player stands. Once the player stands, the dealer must draw cards until the dealer has a score of at least 17 or until the dealer busts.

3.2 Blackjack as an MDP

To simulate an environment for blackjack as a Markov Decision Process, we need to create a 5 element tuple $< \mathbf{S}, \mathbf{A}, T(s,a,s'), R(s,a,s'), \gamma >$ as described earlier. We can already set $\gamma = 1$ no discounting on the rewards, sine we do not intent to motivate an immediate reward, simply the most optimal outcome in the long run.

3.2.1 State Space

Describing the state space for blackjack is quite difficult, as the number of spaces grows depending on the number of cards we assume to be in the players hand at any given time. While it would be feasible to simulate all possible states within blackjack, it would not be beneficial to any human as any human would be unable to remember the optimal policy/action within those states in the first place. Instead it is more beneficial to think of what to do based on the starting hand of a player. As Fig. 3.1 shows, the explored state space of initial states would then consist of the players hand and the exposed card of the dealer.

Choosing only the initial states to explore already reduces the score significantly, but the state space can still be reduced further if we actually consider the score of the players hand against the face card of the dealer. For example, let us imagine a situation where the dealer has an exposed King (K), and the player has a hand $< 7, 4 >$, which is a score

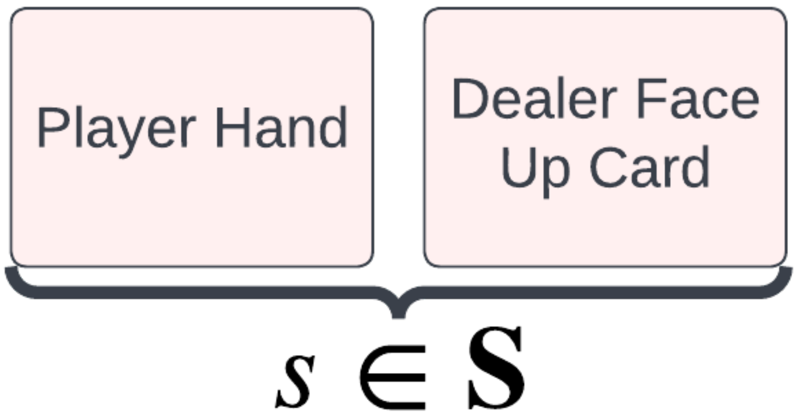

Figure 3.1: An initial state of the system; a tuple consisting of the players starting hand and the dealers exposed card.

of 11, there is no true difference between a hand consisting of $< 7, 4 >$, a hand consisting of $< 6, 5 >$, and a hand consisting of $< 8, 3 >$ since each hand still retains a score of 11. Applying the same logic to the dealers face card, a King (K), Queen (Q), Jack (J), or 10 return the same score of 10, thus we can reduce the dealers possible face cards to a total of 10 total card - Ace (A) and 2-10.

3.2.2 Action Space

The action space of the system essentially consists of the actions that a player - now the agent - can take. The available actions are entirely dependent on the cards the agent receives and all possible actions are listed below.

Stand

The agent can choose to stand at any point, choosing to stand makes it the dealers turn to act who will then draw till at least 17 unless he busts. Taking the stand action causes the system to transition to a terminal state after.

Hit

The agent can choose to hit, this action draws another card for the agent and increases the agents score. If the agents score exceeds 21 then he has busted and system transitions to a terminal state, else, the agents turn continues and the agent can choose from the available action.

Double Down

This action can only be taken in the initial state. Doubling down doubles the wager and draws the agent one extra card after which the agents turn ends. Doubling down causes a transition to a terminal state after.

Split

This action can only be taken in the initial state when the agent is dealt a pair of card, for example $< 5, 5 >$ or $< K, K >$. Splitting the cards separates the agents hand into 2 separate hands. Both hands will be played with the equal wager and another card will be dealt to each hand. For example, if the agent chooses to split $< 5, 5 >$ then the hand will be split into two $< 5, K >$ and $5, 9$ for example. The second card given to each hand is randomly drawn from the deck.

3.2.3 Transition Probabilities

To define a transition probability it is important to understand that the environment can only transition to a state where the agents hand has a higher score. The probabilities of drawing a card between 2-9 are equal to one another, Eqn. 3.1.

$$P(\text{Drawing a card} \in [2, 9]) = \frac{8}{13} \tag{3.1}$$

Since, there are 4 cards that result in a value of 10, the probability of drawing a card with score 10 is 4 times as great, Eqn. 3.2.

$$P(\text{Drawing a card} \in \{K, Q, J, 10\}) = \frac{4}{13} \tag{3.2}$$

Finally, the Ace (A) is the special card which can act as 1 or 11, whichever produces a higher score less than or equal to 21, therefore it has a $\frac{1}{13}$ chance of being drawn, Eqn. 3.3.

$$P(\text{Draw an } A) = \frac{1}{13} \tag{3.3}$$

This allows us to define a transition function $T(s, a, s')$ between any two states that are not initial states. Since only initial states have special actions once a transition is made to a next state it can simply be considered as the initial state the new state corresponds to. For example, assume an agent is given starting hand $< 6, 5 >$, score 11, and chooses to take the hit action thereby transitioning to state with hand $< 6, 5, 4 >$, score 15. The agent can simply consider this state to be the same as any state with initial score 15, for example $< 9, 6 >$ or $< 8, 7 >$ since the only actions the agent can take at this point are to stand or hit. Thereby giving us Eqn. 3.4.

$$T(s, a, s') = \begin{cases} \frac{8}{13} & \text{score increases by 2-9} \\ \frac{4}{13} & \text{score increases by 10} \\ \frac{1}{13} & \text{score increases by 1 or 11} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

3.2.4 Reward Function

The reward function, $R(s, a, s')$ is actually quite simple, there is a reward of -1 if s' is a terminal state in which the agents hand scores higher than 21. There is a reward of 1 if s' is a terminal state and the agents hand scores lower than or equal to 21 **but** higher than the dealers hand. Else, there is a reward of 0, Eqn. 3.5.

$$R(s, a, s') = \begin{cases} \text{if } s' \text{ is terminal} & \begin{cases} 1 & \text{Agent score} \leq 21 \text{ and Agent score} > \text{dealer score} \\ -1 & \text{Agent score} > 21 \\ -1 & \text{Agent score} < 21 \text{ and Agent score} < \text{dealer score)} \end{cases} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

3.3 Overview of the Approach

As discussed in earlier sections, the objective of an Markov Decision Process is to find the best possible decisions to maximise the reward received within a certain time period in an environment. Now that we have established the dynamics of Blackjack as an MDP, we will continue on to perform Value Iteration and Policy extraction on a simulated environment of blackjack with the State and Action space as discussed. We will perform policy extraction and value iteration as described in Algorithms 1 and 2. The best policy for initial states will then be discussed alongside the other results.

Chapter 4

Implementation

4.1 Monte Carlo Simulations

The State Space for our project, **S**, is a rather large State Space considering that the probability distribution for each transition, especially considering the difference between hard and soft states or pairs who have entirely different options available to them makes it difficult to define the transitions themselves. Thus, instead we can use Monte Carlo simulations for each state within the state space. We carry out the simulation 1 million times for each state within the state space and observe how many times the state transitions to a different state. Using a hash table we can count the number of times these transitions occurs thereby allowing us to approximate the true distribution extremely accurately.

```python
def generatePossibleStates():
    initialStateToOutcomeMap = defaultdict(lambda:
        defaultdict(int))
    possibleScore = list(range(4,21))
    drawableDeck = list(range(2,12))
    drawableDeck.extend([10,10,10])
    aceCount = 0
    for initialScore in possibleScore:
        for i in range(monteCarloIterations):
            aceCount = 0
            drawnCard = random.choice(drawableDeck)
            currScore = drawnCard + initialScore
            if drawnCard == 11:
                aceCount += 1
            while currScore > 21 and aceCount > 0:
                currScore = currScore -11 + 1
                aceCount -= 1
            if currScore > 21:
                currScore = -1
            initialStateToOutcomeMap[initialScore][currScore]
                += 1
    return initialStateToOutcomeMap
```

Listing 4.1: Sample snippet of a state generating helper function. Creates all the possible initial states that do not include an Ace in the initial state and uses Monte Carlo simulations to derive their transition frequencies and therefore their transition probabilities as well.

As can be seen in Listing. 4.1, we use Monte Carlo Methods, to generate the transition frequencies for all the possible transition any of those initial states can make by using Monte Carlo Simulations. Once the frequency has been derived, we can easily approximate the transition probability by dividing by the number of iterations, in this case 1 million.

4.2 Value Iteration

Value iteration was carried out as described earlier in Algorithm 1. Listing. 4.2 shows the same algorithm in action with slight modification to accommodate the different handling methods for each different action as well as the change in State space.

```
for i in range(1000):
    for currState in agentInitialStates:
        for faceCard in dealerFaceCards:
            Vs_i = 0
            maxAction = None
            Vcurr_Double = 0
            Vcurr_Hit = 0
            Vcurr_Stand = 0
            for currAction in availableActions:
                if currAction == "Stand":
                    Vcurr_Stand = 0
                    for currDealerState, currDealerStateCount in
    faceCardtoDealerOutcomeMap[faceCard].items():
                        currDealerStateProbability =
    currDealerStateCount/monteCarloIterations
                        reward = calculateReward(currState,
    currDealerState)
                        Vcurr_Stand += currDealerStateProbability*(
    reward)
                elif currAction == "Hit":
                    Vcurr_Hit = 0
                    for nextState, nextStateCount in initialStateToOutcomeMap[currState].items():
                        nextStateProbability = nextStateCount/monteCarloIterations
                        reward = calculateReward(nextState, None)
                        Vcurr_Hit += nextStateProbability*(reward + V_map[(nextState, faceCard)])
                elif currAction == "DoubleDown":
                    Vcurr_Double = 0
                    for nextState, nextStateCount in initialStateToOutcomeMap[currState].items():
                        nextStateProbability = nextStateCount/monteCarloIterations
                        for currDealerState, currDealerStateCount in faceCardtoDealerOutcomeMap[faceCard].items():
                            currDealerStateProbability = currDealerStateCount/monteCarloIterations
                            reward = 2*calculateReward(nextState, currDealerState)
                            Vcurr_Double += nextStateProbability*currDealerStateProbability*(reward)
                Vs_i = max(Vcurr_Double,Vcurr_Hit,Vcurr_Stand)
                if Vs_i == Vcurr_Double:
                    maxAction = "DoubleDown"
                elif Vs_i == Vcurr_Hit:
                    maxAction = "Hit"
                else:
                    maxAction = "Stand"
            V_map[(currState, faceCard)] = Vs_i
            V_maxAction[(currState, faceCard)] = maxAction
```

Listing 4.2: Sample snippet for the value iteration algorithm used to derive both $V_*(s)$ as well the optimal policy $\pi_*(s)$.

Listing 4.2 demonstrates how costly value iteration usually is $O(|S|^2A)$ time complexity. This is the reason one needs to think of complex ways to reduce the scope of the problem itself instead of trying to brute force the entire solution.

Chapter 5

Evaluation

5.1 Experiments

The parameters for any value iteration algorithm are almost always going to be the parameters described in Table **??**. The state and action spaces, respectively $\mathbf{S}, \mathbf{A}$, and the transition probabilities and the reward function, respectively, $T(s, a, s'), R(s, a, s')$ however may or may not change depending on the specific experiment.

Table 5.1: Parameters for value iteration until convergence for the systems within this project.

PARAMETERS					
Episodes	$\mathbf{S}$	$\mathbf{A}$	$T(s, a, s')$	$R(s, a, s')$	γ

5.1.1 Hard Initial States

A hard state in blackjack is simply a state where the value of a hand cannot be two separate values. Quite simply, it means that there are no Aces in the agents initial hand. The parameters for value iteration over such states are as follows.

Episodes

We continue to iterate until convergence, $V_*(s)\ \forall s \in \mathbf{S}$.

States

Recalling that the state of an initial hard state in this system is defined by the combination of the initial hand/score of the agent and the face card of the dealer. Let us call i_A to be the initial score of the agent and i_D to be the face card of the dealer. We can then deduce that $i_A \in [4, 20]$ - which are all the possible scores for an initial hand not containing an Ace. We also have $i_D \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A\} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, A\}$ - which are all the possible face cards a dealer can get - the subset $\{10, J, Q, K\}$ can be simplified to just $\{10\}$ since the face cards Jack (J), Queen (Q), and King (K) simply have a score of 10 and no other special properties. This gives us a total State Space of $||S|| = 17(10) = 170$.

Actions

The actions for this experiment are simply $\mathbf{A} = \{\text{Hit, Double Down, Stand}\}$.

Transitions and Rewards

The rewards and transitions are as described in Eqn. 3.5, 3.4.

5.1.2 Soft Initial States

Soft initial states add an additional element to the game. We now need to account for if we transition to another soft state or if we transition to a hard state. In case we transition to a soft state then the optimal actions should still be drawn from the optimal policy for soft states. For example, imagine a starting hand $< A, 5 >$, which has score 16 or 6, if we hit and draw a 2 thereby transitioning to $< A, 5, 2 >$, which has score 18 or 8. We need to account for the additional options present in this scenario, the way to manage that is by drawing from the probabilities for a state which is formed with score 18 or 8, thus we would then draw from the probabilities generated for state $< A, 7 >$. Essentially for all soft states s, if $s < s' \leq 21$ then we know that s' is also a soft state. Otherwise it is a hard state and we draw from the probabilities used previously in this project.

States

Since we have reduced the state space by forcing one card to be an Ace (A). We now have $i_A \in [13, 21]$ and $i_D \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, A\}$ as before. This gives us a total State Space of $||S|| = 9(10) = 90$.

Actions

The actions remain the same; $\mathbf{A} = \{\text{Hit, Double Down, Stand}\}$.

Transitions and Rewards

Additional changes are made to account for transitioning to a hard or soft state. The rewards however remain the same as in Eqn. 3.5.

5.1.3 Allowing Splitting on Pairs

We expand the actions section by alllowing splitting of the hand. This increases the search space since that is proportional to both **S** and **A**.

States

We now have $i_A \in \{4, 6, 8, 10, 12, 14, 16, 18, 20, AA\}$.

Actions

The action space is increased; $\mathbf{A} \in \{\text{Hit, Stand, Split, Double Down}\}$

Transitions and Rewards

Rewards and Detriments are doubled in the case of a Split as each hand takes on an equivalent wager. The transitions between states are unchanged.

5.2 Results

5.2.1 Hard Initial States

For hard initial states, initial state not containing an Ace, the results of our value iteration can be seen in Table. 5.2. Our results are quite interesting, they mainly show that the agent was able to stay profitable for a number of a initial states including $i_A \in \{20, 19, 18, 11, 10, 9, 8\}$, while only considering the actions of Hit, Double Down, and Stand. The exact actions taken to achieve the results can be seen in Table. 5.3

5.2.2 Soft Initial States

As can be seen in Table. 5.4, 5.5. The soft states are significantly more profitable, with most states being fairly rewarding. There is also a fairly simple strategy for playing a soft

Table 5.2: Converged $V_*(s)$ values for all possible initial hard states

AGENT	**DEALER FACE CARD**									
SCORE	2	3	4	5	6	7	8	9	10	11
20	0.641	0.650	0.661	0.669	0.704	0.773	0.791	0.759	0.435	0.145
19	0.386	0.404	0.423	0.438	0.496	0.615	0.593	0.288	-0.018	-0.116
18	0.122	0.148	0.176	0.198	0.283	0.399	0.105	-0.182	-0.241	-0.378
17	-0.152	-0.117	-0.079	-0.045	0.011	-0.107	-0.382	-0.423	-0.464	-0.640
16	-0.292	-0.253	-0.210	-0.167	-0.153	-0.476	-0.511	-0.544	-0.575	-0.692
15	-0.292	-0.253	-0.210	-0.167	-0.153	-0.446	-0.493	-0.544	-0.575	-0.668
14	-0.293	-0.253	-0.210	-0.167	-0.153	-0.402	-0.452	-0.511	-0.572	-0.642
13	-0.293	-0.253	-0.210	-0.168	-0.153	-0.356	-0.410	-0.4733	-0.539	-0.614
12	-0.293	-0.253	-0.210	-0.167	-0.153	-0.308	-0.367	-0.434	-0.504	-0.585
11	0.472	0.518	0.568	0.614	0.669	0.463	0.352	0.229	0.061	-0.322
10	0.360	0.408	0.461	0.511	0.576	0.391	0.286	0.143	-0.135	-0.289
9	0.090	0.120	0.183	0.242	0.318	0.154	0.078	-0.074	-0.235	-0.374
8	0.007	0.042	0.079	0.113	0.163	0.075	-0.074	-0.231	-0.330	-0.468
7	-0.080	-0.042	0.001	0.042	0.086	-0.078	-0.226	-0.314	-0.398	-0.548
6	-0.121	-0.082	-0.036	0.007	0.039	-0.188	-0.270	-0.346	-0.431	-0.567
5	-0.115	-0.076	-0.030	0.014	0.045	-0.179	-0.254	-0.338	-0.425	-0.555
4	-0.102	-0.062	-0.017	0.027	0.060	-0.146	-0.226	-0.315	-0.413	-0.539

hand, in that you can just hit when lower than $< A, 6 >$ if the dealer has a 9 or higher.

5.2.3 Allowing Splitting on Pairs

Allowing splitting on the pairs, Table. 5.6, makes the $< A, A >$ delightful, and other weaker hand such as $< 8, 8 >$, score 16, which is usually weak as seen earlier become more profitable. This is likely due to the fact that splitting the pair allows you the chance to hit an Ace which significantly improves the hand as any pair that draws an Ace can be considered as a soft hand with the only drawback being that one cannot Double Down on it, this however is not an issue as Table. 5.5 clearly shows, we do not want to Double Down on a soft hand anyway.

Table 5.3: Action from the optimum policy $\pi_*(s)$ for all possible initial hard states

AGENT	**DEALER FACE CARD**									
SCORE	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	S	S	S	S	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	S	S	S	S	S	H	H	H	H	H
11	DD	DD	DD	DD	DD	DD	DD	DD	DD	H
10	DD	DD	DD	DD	DD	DD	DD	DD	H	H
9	H	DD	DD	DD	DD	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

*H: Hit, S: Stand, DD: Double Down

Table 5.4: Converged $V_*(s)$ values for all possible initial soft states

AGENT	**DEALER FACE CARD**									
HAND	2	3	4	5	6	7	8	9	10	11
$< A, 10 >$	0.882	0.885	0.888	0.891	0.902	0.926	0.930	0.939	0.889	0.638
$< A, 9 >$	0.641	0.650	0.661	0.669	0.704	0.773	0.791	0.759	0.435	0.145
$< A, 8 >$	0.389	0.405	0.423	0.439	0.496	0.616	0.594	0.289	-0.019	-0.116
$< A, 7 >$	0.122	0.149	0.177	0.198	0.283	0.399	0.106	-0.129	-0.222	-0.361
$< A, 6 >$	-0.003	0.027	0.059	0.090	0.128	0.023	-0.103	-0.180	-0.272	-0.420
$< A, 5 >$	-0.024	0.007	0.041	0.072	0.099	-0.052	-0.111	-0.189	-0.283	-0.414
$< A, 4 >$	-0.003	0.027	0.060	0.092	0.119	-0.013	-0.076	-0.161	-0.260	-0.388
$< A, 3 >$	0.019	0.049	0.081	0.112	0.140	0.031	-0.035	-0.124	-0.235	-0.360
$< A, 2 >$	0.043	0.072	0.103	0.133	0.162	0.075	0.007	-0.086	-0.202	-0.332

Table 5.5: Action from the optimum policy $\pi_*(s)$ for all possible initial soft states

AGENT	**DEALER FACE CARD**									
HAND	2	3	4	5	6	7	8	9	10	11
$< A, 10 >$	S	S	S	S	S	S	S	S	S	S
$< A, 9 >$	S	S	S	S	S	S	S	S	S	S
$< A, 8 >$	S	S	S	S	S	S	S	S	S	S
$< A, 7 >$	S	S	S	S	S	S	S	H	H	H
$< A, 6 >$	H	H	H	H	H	H	H	H	H	H
$< A, 5 >$	H	H	H	H	H	H	H	H	H	H
$< A, 4 >$	H	H	H	H	H	H	H	H	H	H
$< A, 3 >$	H	H	H	H	H	H	H	H	H	H
$< A, 2 >$	H	H	H	H	H	H	H	H	H	H

*H: Hit, S: Stand, DD: Double Down

Table 5.6: Action from the optimum policy $\pi_*(s)$ for all possible initial pair states

AGENT	**DEALER FACE CARD**									
HAND	2	3	4	5	6	7	8	9	10	11
$< A, A >$	SP	SP	SP	SP	SP	SP	SP	SP	SP	SP
$< 10, 10 >$	S	S	S	S	S	S	S	S	S	S
$< 9, 9 >$	S	S	S	S	S	S	S	S	S	S
$< 8, 8 >$	SP	SP	SP	SP	SP	SP	SP	SP	SP	SP
$< 7, 7 >$	SP	SP	SP	SP	SP	SP	SP	SP	SP	SP
$< 6, 6 >$	SP	DD	DD	DD	DD	SP	SP	H	H	H
$< 5, 5 >$	DD	DD	DD	DD	DD	DD	DD	DD	H	H
$< 4, 4 >$	H	H	H	H	H	H	H	S	H	H
$< 3, 3 >$	SP	SP	SP	SP	SP	SP	SP	SP	SP	SP
$< 2, 2 >$	SP	SP	SP	SP	SP	SP	SP	SP	SP	SP

*H: Hit, S: Stand, DD: Double Down, SP: Split

Chapter 6

Conclusions & Future Work

In this dissertation we explored the benefits of Markov Decision Processes in optimizing how an agent can optimally play blackjack. We provided a deep background on Markov Decision Processes, followed by a detailed design showing how they can be used to simulate the a blackjack environment. Our implementation, was rather expensive in a computing cost, but considering the extremely large game state, it could be argued that we did a fine job.

Considering the success of our model, the agents behavior was evaluated over a number of episodes and was shown to be profitable in comparison to what is expected in blackjack where the house has an edge. In fact, our models were tested in the hardest blackjack conditions possible, since the larger then number of decks the larger the house edge, up to a certain extent - asymptotic around 0.55. We used a deck which was technically infinite thereby making it the hardest environment possible.

6.1 Future Work

For future works there are many options an interested party could take to build on the works within this game. If one is interested in exploring other Reinforcement Learning methods then they could look to explore some of those for blackjack, they could instead focus on a different algorithm such Q-Learning or one of its variants such as Deep Q-Learning to evaluate how other algorithms compare. If another is more interested in exploring MDP's for other stochastic games, then there are opportunities in games such as Backgammon and poker, although one should be prepared if they choose to undertake such a task. The extremely large State Space makes poker a very hard game to optimize for, someone could compare their implementation against solvers available on the web that are extremely well reputed even by professional player. Finally, if someone wants to build directly on the works conducted in this dissertation, then they could compare how an agent improves with limited decks, this would require someone to describe an exact transition probability function which could be a rewarding and challenging endeavor.

Bibliography

[1] G. Borrageiro, N. Firoozye, and P. Barucca, “The recurrent reinforcement learning crypto agent,” *IEEE Access*, vol. 10, pp. 38590–38599, 2022.

[2] G. Borrageiro, N. Firoozye, and P. Barucca, “Reinforcement learning for systematic fx trading,” *IEEE Access*, vol. 10, pp. 5024–5036, 2022.

[3] G. Borrageiro, N. Firoozye, and P. Barucca, “Reinforcement learning for systematic fx trading,” *IEEE Access*, vol. 10, pp. 5024–5036, 2022.

[4] Y. Sun, “Performance of reinforcement learning on traditional video games,” in *2021 3rd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, pp. 276–279, 2021.

[5] P. A. Gagniuc, *Markov chains: from theory to implementation and experimentation.* John Wiley & Sons, 2017.

[6] E. Pashenkova, I. Rish, and R. Dechter, “Value iteration and policy iteration algorithms for markov decision problem,” in *AAAI’96: Workshop on Structural Issues in Planning and Temporal Reasoning*, Citeseer, 1996.

[7] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[8] W. Zhufang and Z. Shengjun, “Decision-making on investment in convertible bond based on markov process,” in *2009 Chinese Control and Decision Conference*, pp. 6020–6024, 2009.

[9] X. Yang and G. Tang, “Deep study on marketing decision under the price fluctuation of pig market based on bayesian method,” in *2020 16th Dahe Fortune China Forum and Chinese High-educational Management Annual Academic Conference (DFHMC)*, pp. 73–76, 2020.

[10] T. Fujita, "Associative criteria in mutually dependent markov decision processes," in *2014 IIAI 3rd International Conference on Advanced Applied Informatics*, pp. 147–150, 2014.

[11] M. N. Moghadasi, A. T. Haghighat, and S. S. Ghidary, "Evaluating markov decision process as a model for decision making under uncertainty environment," in *2007 International Conference on Machine Learning and Cybernetics*, vol. 5, pp. 2446–2450, 2007.

[12] V. Gupta, A. Trivedi, and W. Godfrey, "Enhancing structure learning of markov network using alternating decision trees," in *2017 IEEE 7th International Advance Computing Conference (IACC)*, pp. 904–908, 2017.

[13] K.-M. Wang, K.-H. Liang, M.-C. Tsou, and S.-S. Wai, "Develop sara markov model for evaluating the decision on search and rescue task," in *2011 Defense Science Research Conference and Expo (DSR)*, pp. 1–4, 2011.

[14] T. T. Lin and C.-S. Hsieh, "A decision analysis for the dynamic crop rotation model with markov process's concept," in *2013 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 159–163, 2013.

[15] Y. Yang, L. Weimin, Z. Wei, and B. Ying, "A novel ims qos strategy decision method based on markov model for multipath transmission," in *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pp. 990–993, 2013.

[16] T. K. Toai and V. M. Huan, "Implementing the markov decision process for efficient water utilization with arduino board in agriculture," in *2019 International Conference on System Science and Engineering (ICSSE)*, pp. 335–340, 2019.

[17] H. M. Luqman and M. Zaffar, "Chess brain and autonomous chess playing robotic system," in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 211–216, 2016.

[18] S. Sarker, "Wizard chess: An autonomous chess playing robot," in *2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, pp. 475–478, 2015.

[19] M. Levene and J. Bar-Ilan, "Comparing typical opening move choices made by humans and chess engines," *The Computer Journal*, vol. 50, no. 5, pp. 567–573, 2007.

[20] N. Upasani, A. Gaikwad, A. Patel, N. Modani, P. Bijamwar, and S. Patil, “Dev-zero: A chess engine,” in *2021 International Conference on Communication information and Computing Technology (ICCICT)*, pp. 1–6, 2021.

[21] D. Meng, B. Jianbo, Q. Yizhong, F. Yao, and L. Shuqin, “Design of amazon chess game system based on reinforcement learning,” in *2019 Chinese Control And Decision Conference (CCDC)*, pp. 6337–6342, 2019.

[22] B. Gravell, K. Ganapathy, and T. Summers, “Policy iteration for linear quadratic games with stochastic parameters,” *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 307–312, 2021.

[23] T. E. Duncan and B. Pasik-Duncan, “A solvable stochastic differential game in the two-sphere,” in *52nd IEEE Conference on Decision and Control*, pp. 7833–7837, 2013.

[24] T. E. Duncan and B. Pasik-Duncan, “Some stochastic differential games with state dependent noise,” in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 3773–3777, 2015.

[25] A. Tcheukam and H. Tembine, “One swarm per queen: A particle swarm learning for stochastic games,” in *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 144–145, 2016.

[26] H. Sun and L. Jiang, “Linear infinite horizon quadratic differential games for stochastic systems: Discrete-time case,” in *2011 Chinese Control and Decision Conference (CCDC)*, pp. 1733–1737, 2011.

[27] G. Kendall and C. Smith, “The evolution of blackjack strategies,” in *The 2003 Congress on Evolutionary Computation, 2003. CEC ’03.*, vol. 4, pp. 2474–2481 Vol.4, 2003.

[28] D. Fogel, “Evolving strategies in blackjack,” in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 2, pp. 1427–1434 Vol.2, 2004.

[29] A. Perez-Uribe and E. Sanchez, “Blackjack as a test bed for learning strategies in neural networks,” in *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, vol. 3, pp. 2022–2027 vol.3, 1998.