Reinforcement Learning algorithm for playing Black Jack.

Course project

Applied Artificial Intelligence

Petar Bechev

Mihail Nenchev

Aleksi Stoyanov

Table of Contents

[Introduction 2](#_Toc134554735)

[Blackjack game rules 3](#_Toc134554736)

[Implementation 4](#_Toc134554737)

[Gameplay engine 4](#_Toc134554738)

[**Action Space** 4](#_Toc134554739)

[**Observation** **Space** 4](#_Toc134554740)

[**Rewards** 4](#_Toc134554741)

[RL Algorithm 4](#_Toc134554742)

[Additional features in the project 5](#_Toc134554743)

[Saving the training model 5](#_Toc134554744)

[Real-time simulator 5](#_Toc134554745)

[Bulk test 6](#_Toc134554746)

[Statistics 6](#_Toc134554747)

[Achievements 7](#_Toc134554748)

# Introduction

We used the concept of Reinforcement learning to train an algorithm to play Blackjack.

Reinforcement Learning (RL) is a type of machine learning that involves an agent learning to make decisions through interactions with an environment. In RL, the agent receives feedback in the form of rewards or penalties based on its actions, which guides its decision-making process. The goal of the agent is to learn a policy that maps states to actions, maximizing its cumulative reward over time.

The algorithm is not aware of the rules of the game so that it tries in the beginning with random actions in order to receive an outcome whether or not the decision is correct or not (win or lose). To better understand the concept let’s look at the components of an RL problem:

1. Environment

The environment is the world in which the agent operates. It is defined by a set of states, actions, and rewards. The agent takes actions in the environment and receives rewards or penalties based on those actions.

This is the Blackjack game with its rules and set of action, etc.

2. State

The state represents the current situation of the agent in the environment. It is a snapshot of the environment at a particular time. The agent selects actions based on the current state.

In our algorithm the state is the sum of all cards that we have and the card of the dealer. Based on these two values the algorithm will make a decision whether to stand or get a new card.

3. Action

An action is a decision made by the agent in response to the current state. Actions can change the state of the environment and result in rewards or penalties.

The actions in our case are to take a new card or to stand

4. Reward

The reward is a signal from the environment that tells the agent how well it is performing. The agent's goal is to maximize the cumulative reward over time.

-1 (lose), 0 (draw), 1 (win) are the possible rewards that our algorithm receives based on the actions it made.

5. Policy

A policy is a mapping from states to actions. It tells the agent what action to take in each state to maximize its reward.

6. Value

The value represents the expected cumulative reward the agent will receive if it starts in a particular state or takes a particular action. It is an estimate of how good a state or action is.

# Blackjack game rules

Blackjack hands are scored by their point total. The hand with the highest total wins as long as it doesn't exceed 21; a hand with a higher total than 21 is said to bust. Cards 2 through 10 are worth their face value, and face cards (jack, queen, king) are also worth 10. An ace's value is 11 unless this would cause the player to bust, in which case it is worth 1. A hand in which an ace's value is counted as 11 is called a soft hand, because it cannot be busted if the player draws another card.

The goal of each player is to beat the dealer by having the higher, unbusted hand. Note that if the player busts he loses, even if the dealer also busts (therefore Blackjack favors the dealer). If both the player and the dealer have the same point value, it is called a "push", and neither player nor dealer wins the hand

**The player's** **options** for playing his or her hand are:

**Hit**: Take another card.

**Stand**: Take no more cards.

When **the dealer** has served every player, the dealers face-down card is turned up. If the total is 17 or more, it must stand. If the total is 16 or under, they must take a card. The dealer must continue to take cards until the total is 17 or more, at which point the dealer must stand. If the dealer has an ace, and counting it as 11 would bring the total to 17 or more (but not over 21), the dealer must count the ace as 11 and stand. The dealer's decisions, then, are automatic on all plays, whereas the player always has the option of taking one or more cards.

# Implementation

We used Python programming language for implementing this project as it is the easiest and the most suitable programming language for Machine Learning

## Gameplay engine

As we explained the rules of the game, these rules should be implemented in a gameplay engine to simulate a real game with all sets of actions and to return the reward for every action.

We used Gym as open-source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments, as well as a standard set of environments compliant with that API.

We modified the code for Blackjack so that we can display a real-time simulation of RL playing with the dealer and showing all the actions that are taken from the algorithm.

### **Action Space**

There are two actions: stick (0), and hit (1).

### **Observation** **Space**

The observation consists of a 3-tuple containing: the player’s current sum, the value of the dealer’s one showing card (1-10 where 1 is ace), and whether the player holds a usable ace (0 or 1).

### **Rewards**

* win game: +1
* lose game: -1
* draw game: 0

## RL Algorithm

We used value-based reinforcement learning algorithm that learn the value of each state or state-action pair and use these values to make decisions.

Q-Learning is a widely used value-based reinforcement learning algorithm. It learns the value of each state-action pair by updating a Q-table based on the rewards it receives for different actions in different states. The Q-table is a matrix that stores the expected value of each state-action pair.

The transition rule of Q learning is a very simple formula:

**Q(state, action) = R(state, action) + Gamma \* Max[Q(next state, all actions)]**

According to this formula, a value assigned to a specific element of matrix Q is equal to the sum of the corresponding value in matrix R and the learning parameter Gamma, multiplied by the maximum value of Q for all possible actions in the next state.

The Q-Learning algorithm goes as follows:

*1. Set the gamma parameter and environment rewards in matrix R.*

*2. Initialize matrix Q to zero.*

*3. For each episode:*

*Select a random initial state.*

*Do While the goal state hasn’t been reached.*

* *Select one among all possible actions for the current state.*
* *Using this possible action, consider going to the next state.*
* *Get maximum Q value for this next state based on all possible actions.*
* *Compute: Q(state, action) = R(state, action) + Gamma \* Max[Q(next state, all actions)]*
* *Set the next state as the current state.*

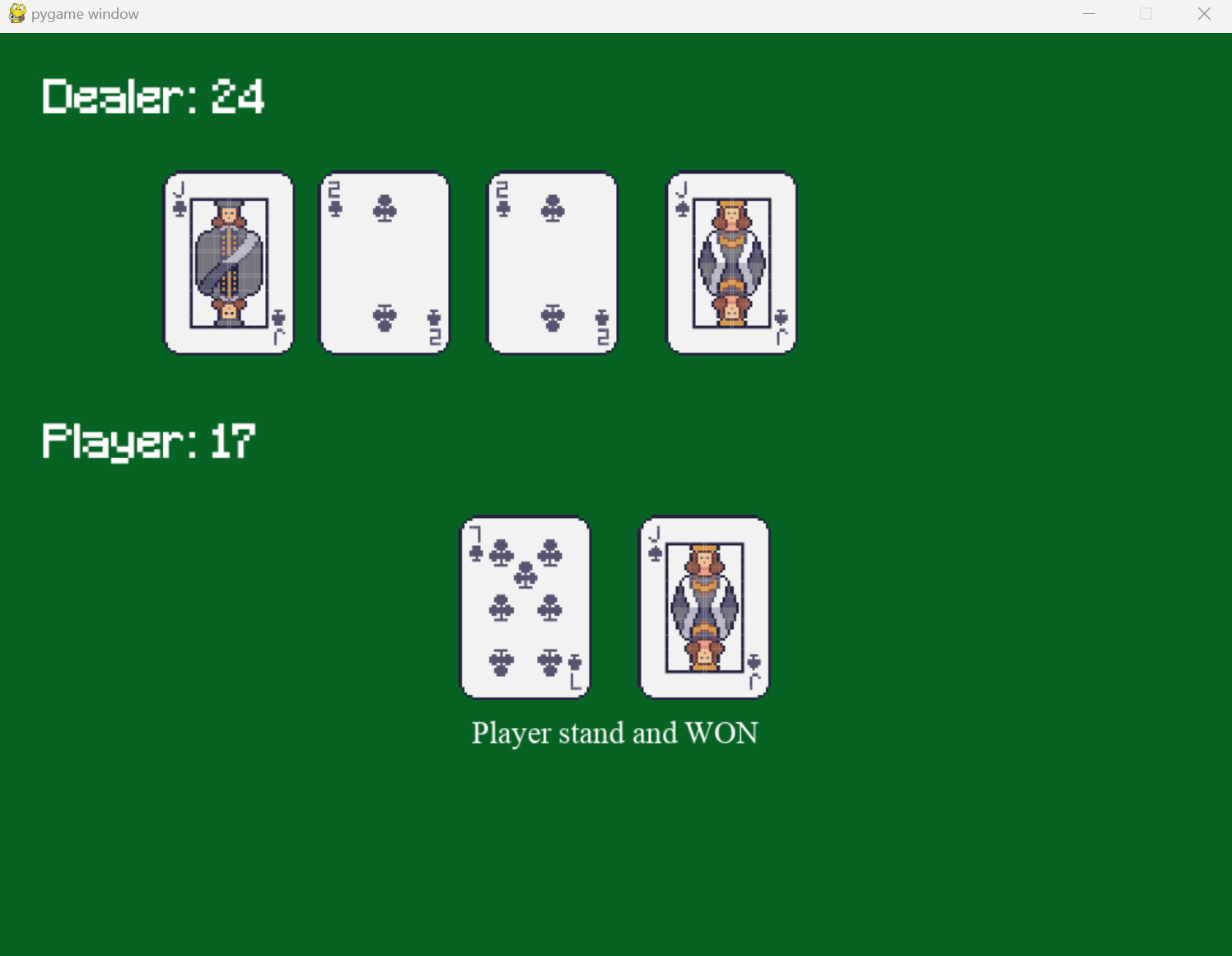
## Additional features in the project

As we implemented the base logic and the RL algorithm, we implemented several features:

### Saving the training model

As learning takes time and resources, we implemented a way to store the trained model in a file on the local computer’s drive so that it can be reused or retrained starting from this point. For this purpose, we used **dill package** that extends python’s **pickle module** for serializing and de-serializing python objects to the majority of the built-in python types.

### Real-time simulator



We built a simulator in order to test and demonstrate how the RL algorithm is taking action in real-time, and based on those actions what is the outcome.

### Bulk test

We implemented this feature to be able to evaluate the success rate of the RL algorithm playing 1000 games. Based on those observations we made comparisons between untrained and trained model.

### Statistics

Картина, която съдържа диаграма

Описанието е генерирано автоматично

Картина, която съдържа диаграма

Описанието е генерирано автоматично

These data charts show the strategy the RL algorithm is using when playing Blackjack, when it stands and when it hits.

# Achievements

In order to evaluate the success rate of the RL algorithm, let’s look at the charts of trained and untrained model:

As we see the trained model is much more accurate and wins more frequently that the untrained one.

However, you might notice that the trained model does not wins more that 50% of the times, this has its answer. This is a gambling game so it is made in a way that the casino should make a profit, even with the best strategy possible it is a theory of probability.

When we check in the internet for the odds of winning in Blackjack based on mathematicians and compare it with our results, this is what happens:

As we can see our RL algorithm is performing slightly better that what the theory of probability says, meaning that the algorithm has found the best possible strategy for playing this game.

Изготвили:

Петър Бечев, 471220023 (76)

Михаил Ненчев, 471220024 (76)

Алекси Стоянов, 471219075 (76)