CS5100 Final Project Report

Project Name: Blackjack Card Game

Team Members: Shifu Xu, Feng Shi

Part One: Project Overview

Introduction to Blackjack:
Blackjack is a famous card game around the world. The player will bet money and paly against the computer. If the player loses all his/her money, then the player loses this game. Otherwise, if the player wins its fund after the game, then dealer (computer) loses the game. Each time, the player can click the "ai" button we provide to replace the human player to play the backjack. The "ai" robot will execute Q-learning and find the optimal policy to paly the game against dealer computer.

Rule of Blackjack:
The object of this game is to grab a hand of cards that has a value of no more than 21. If the total value is over 21, then we call it "bust". The player will play against the dealer (computer) here. If the player holds a hand of cards whose total value is not busting and greater than dealer's value, then the player wins. Otherwise, the computer wins.

It goes like this:

1. There are 52 cards in the deck, and they will be shuffled at start of every round.
2. Initially, each side will hold two cards. For the dealer, one card is face-down, and the other one is face-up.
3. Each card represents a value printed to the card, and we do not have to care about the set of the card. In addition, Jacks, Queens and Kings will represent value 10. However, there is a special case for Ace. Normally, Ace will represent value 11, but if the player will bust, then Ace can represent 1 instead of 11 in this special case.
4. If the player's initial cards are with value of 10 and an Ace, then the player holds a blackjack and the player will win the game automatically, unless the dealer also has a hand of blackjack. If the player uses the blackjack to win the game, then the player can get 2 times paid back. The same for the dealer, if the deal holds a blackjack, it will win this round unless player also has a blackjack.
5. If the dealer and the player hold the same value, this is tie. Nobody wins, the game will keep continuing.
6. The player can have several chooses every round, which is hit, stand and double.
   Hit: the player can have a new card from the deck

Stand: The player does not want any more cards
Double: The player will take one more card and double its bet, then stand.

7. If the player chooses to stand, and now the dealer will keep hitting until its value is larger and equal than 17, then deal must stand. If right now, the dealer bust, then it will lose this round automatically
8. Finally, the values of two hands from player and dealer will be compared, and the higher value will win the bet.

Project Description
In order to implement the blackjack game, we have to transfer this game to an algorithm that we learnt from this course. In this project, we use e-greedy improved Q-Learning algorithm to get the policy which is the action table based on current state. Once we have got the policy table, and then the "ai" robot can choose to hit or stand according to the hand of cards in player and in dealer.

The input for this problem is hand of cards in dealer (especially, the face up card), the hand of cards in player and the bet for this round. The output should be final hand of cards for player, final hand of cards for dealer, and who will win the bet for this round based on the result.

Development Tools and Environment
Developing Environment: PyCharm
Python Version: Python 3.4.3
Libraries: pygame, random, os, sys, _thread, time

Code Information:

1. blackjack.py : the main file to show the interface of the game and deal with every human and computer interactions
2. ai.py : core Q-Learning algorithm, it will return policy for blackjack.py
3. utils.py : provide support functions like createDeck(), shuffle(), etc
4. settings.py : provide basic initialize for loading the images and sounds
5. images/ : contains all the images for cards, background, buttons, etc
6. sounds/ : contains sound of click
7. docs/ : contains README, License and Credits files

Demo Link:
Here is the demo link:
https://www.youtube.com/watch?v=-sEg43ZJFvc&feature=youtu.be
(We also provide a mp4 video, please check in the root folder, it is named: CS5100-Final-Proj-Demo.mp4)

Part Two: Algorithm

Reinforcement Learning Introduction
Reinforcement learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward, where environment is typically formulated as a Markov decision process. Markov means action outcomes depend only on the current state. A Markov Decision Process is a discrete time stochastic control process. At each time step, the process is in some state s, and the decision maker may choose any action a that is available in state s. The process responds at the next time step by randomly moving into a new state s', and giving the decision maker a corresponding reward $R_a(s, s')$.

We use reinforcement learning to train a Q-learning agent to play Blackjack against with dealer machine. Dealer machine adapts a fixed strategy: hit only when the total value of its cards is less than 17, otherwise stand. We train our Q-learning agent with a lot of learning episodes to compute the Q-value map where the key is the state-action pair. Once the training completes, we could play against with dealer machine using the policy derived by choosing the optimal action with the largest Q value from Q-value map according to the current state.

Q-Learning Algorithm Formula

Sample-based Q-value iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \; [R(s,a,s') \; + \; \gamma \; max_{a'} \; Q_K(s',a')]$$

In each episode, receive a sample (s, a, s', r) and then compute new sample estimate Q(s, a):

$$\text{Sample} = \text{R(s, a, s')} + \gamma \; max_{a'} \; Q(s', a')$$

Update Q-value map using running average:

$$\text{Q(s, a)} \leftarrow (1 - \alpha) \text{ Q(s, a)} + (\alpha)[\text{sample}]$$

Use e-greedy action selection strategy to deal with exploration and exploitation:
Act randomly with small probability ϵ, act on current policy (derived from Q-value map) with large probability 1- ϵ. And lower ϵ over time.

Derive the optimal policy for all possible states:
For each state, compare Q-values for all possible actions and choose the best action with the largest q-value.

Blackjack Q-learning definitions:
A state is (dealerCard, playerTotal, hasUseableAce)

dealerCard is dealer's face-up card that player can see.
playerTotal is the total value of player's cards.
hasUseableAce is true iff playerHand's ace can be count as 11 and playerHand's total value is less or equal to 21.

A hand is (total, hasAce)

total is the total value of the hand's cards.
hasAce is true as long as the hand has an ace.

An action can be either "hit" or "stand"

States space:

Dealer card: a card from 1 to 11
Player's total value of cards: from 2 to 21
HasUseableAce: True or false
So the total states space is 11 * 20 * 2 = 440

Pick cards method:

We randomly pick a card from 1 to 13, for the cards with values larger than 10, we count them as 10.

A counterMap is used to store how many times a particular state-action pair has been observed so that we can decrease α over time according to the counter.

Select actions:

We use e-greedy action selection strategy.

Rewards computing:

We compute the reward by comparing the playerHand and dealerHand.
If player's total value > 21, reward = -1
Else if dealer's total value > 21, reward = 1
Else if player's total value < dealer's total value, reward = -1
Else if player's total value == dealer's total value, reward = 0
Else if player's total value > dealer's total value, reward = 1

Blackjack Q-Learning Core Sudo Code

```
Initialize Q(s, a) map
Initialize counter map
Store all possible states

Repeat n learning times:

    Select a state randomly
    Initialize dealerFaceUpCard, dealerHand and playerHand

    While True (For each round):
        Select an action using e-greedy strategy
        Formulate state-action pair (s, a)
        Counter(s, a) + 1

        If player hits:
            Add a card to playerHand

            If player does not bust:
                Compute next state s' by dealerCard and current playerHand
                Compute max Q-value $\gamma\ max_{a'}\ Q(s', a')$ of the next state
                Reward is 0 since we only know dealer card
                Update Q(s, a) using running average, $\alpha$ divided by counter(s, a)
                Update state s to the next state s'

            If player busts:
                Terminal state, max Q-value is 0
                Reward is -1
                Update Q(s, a) using running average, $\alpha$ divided by counter(s, a)
                Break

        If player stands:
            Dealer play with a fixed strategy
            Dealer continue to hit if current total value is less than 17
            Terminal state, max Q-value is 0
            Get reward by comparing dealerHand and playerHand
            Update Q(s, a) using running average, $\alpha$ divided by counter(s, a)
            Break

    Return Q-value map
```

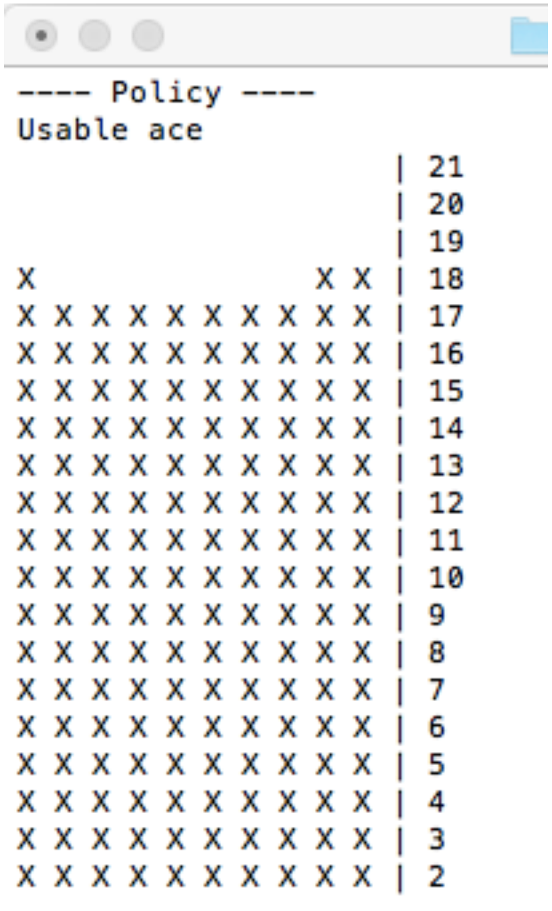
Part Three: Results

We computed the average gain for our blackjack q-learning algorithm. For each round, if our q-learning agent wins, gain + 1. If it's a tie, gain remains unchanged. If dealer wins, gain -1. For different number of training episodes, the average gain is the total gain / the total number of rounds. Here we set the total number of rounds as 100000.

X-axis is the number of training episodes. Start with 500 episodes, every time we multiply 2 as the next number of training episodes. We set the maximum number of training episodes as 4096000 episodes. Y-axis is the average gain of 100000 rounds.

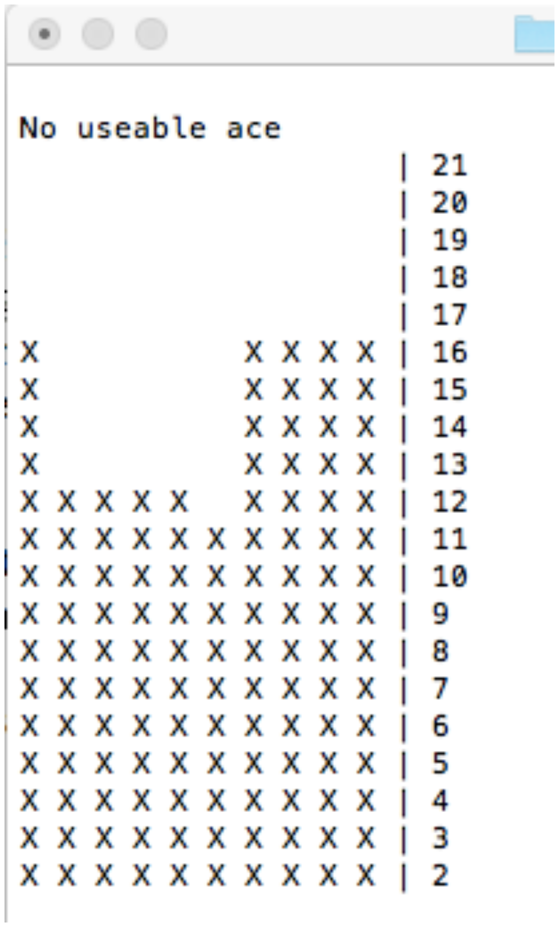
The average gain becomes stable around -0.05 after 2048000 training episodes.

Below are the converged policy table and the chart of the average gain:



```
---- Policy ----
Usable ace
                    | 21
                    | 20
                    | 19
X               X X | 18
X X X X X X X X X X | 17
X X X X X X X X X X | 16
X X X X X X X X X X | 15
X X X X X X X X X X | 14
X X X X X X X X X X | 13
X X X X X X X X X X | 12
X X X X X X X X X X | 11
X X X X X X X X X X | 10
X X X X X X X X X X | 9
X X X X X X X X X X | 8
X X X X X X X X X X | 7
X X X X X X X X X X | 6
X X X X X X X X X X | 5
X X X X X X X X X X | 4
X X X X X X X X X X | 3
X X X X X X X X X X | 2
```

table1. Ace is useable



```

No useable ace
                    | 21
                    | 20
                    | 19
                    | 18
                    | 17
X           X X X X | 16
X           X X X X | 15
X           X X X X | 14
X           X X X X | 13
X X X X X   X X X X | 12
X X X X X X X X X X | 11
X X X X X X X X X X | 10
X X X X X X X X X X | 9
X X X X X X X X X X | 8
X X X X X X X X X X | 7
X X X X X X X X X X | 6
X X X X X X X X X X | 5
X X X X X X X X X X | 4
X X X X X X X X X X | 3
X X X X X X X X X X | 2
```

table2. Ace is not useable

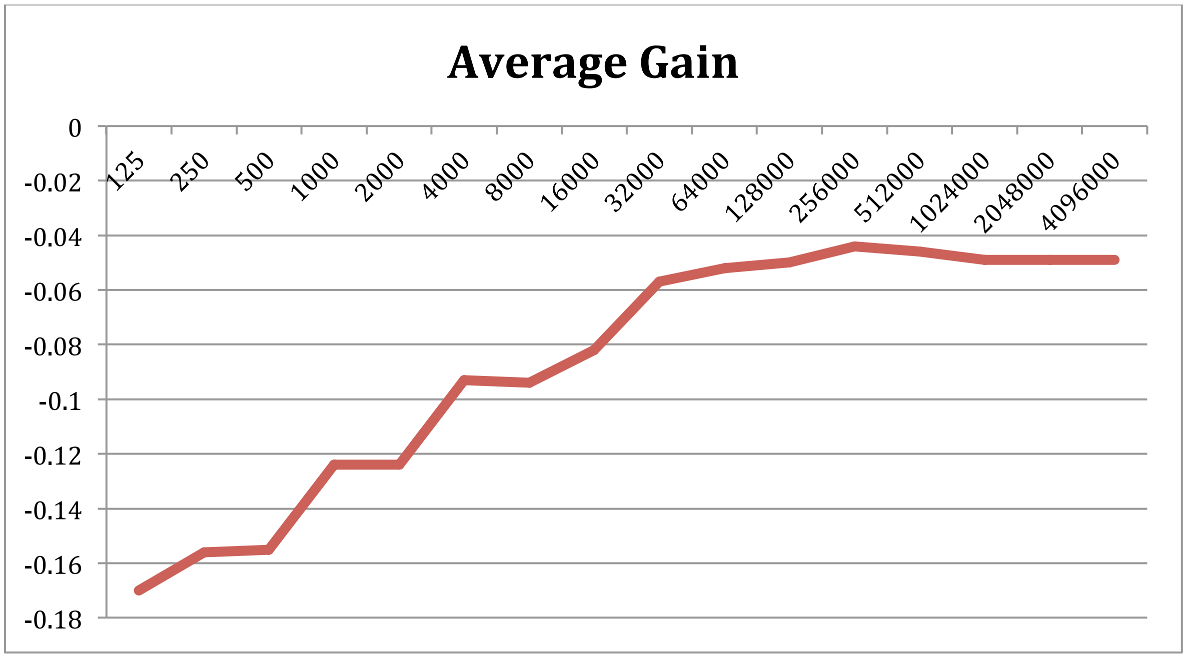


chart1. Converge situation