ANIKET JOSHI

Blackjack

This design doc discusses a fully functioning console-based blackjack game that adheres to the rules below:

1. The goal of blackjack is to beat the dealer's hand without going over 21
2. Face cards are worth 10. Aces are worth 1 or 11, whichever makes a better hand
3. Each player starts with two cards, one of the dealer's cards is hidden until the end
4. To 'Hit' is to ask for another card. To 'Stand' is to hold your total and end your turn
5. If you go over 21 you bust, and the dealer wins regardless of the dealer's hand
6. Dealer will hit until his/her cards total 17 or higher
7. To start each round, you bet a certain amount of money
8. If you win the hand, you will earn the amount of money you bet on the hand

Execution Instructions

Follow this URL to install golang on the target machine: https://golang.org/doc/install

Navigate to the folder where the "go" folder was installed, usually in the user's home directory or a system folder

Run the following commands:

```
go get github.com/anktjsh/blackjack

cd go/src/github.com/anktjsh/blackjack

go run main.go
```

Design

All of the code for this program is written in golang. I decided to use golang because it has minimal syntax and it is very easy to get started with a project in this language. I also did not need to rely heavily on object-oriented principles such as inheritance which golang does not support. I primarily need a struct/object type that I could use and some list-like data structures which golang supports easily.

I also used the external library:

github.com/gookit/color - utility library that defines the colors I used for the different playing cards and outputs

The overall game runs inside of a loop waiting for user input to decide whether to play the game, show the rules or quit. After selecting to play the game, the game initializes a slice, golang's version of a dynamic list to represent the deck of cards as well as two more slices for each player's hands. The deck is shuffled, the cards are dealt and the game commences. As described by the rules, the player keeps playing until they either run out of money, or decide to quit the game. I defined unique types for a PlayingCard, Hand, Deck, and the whole Game to make the representations of each of these objects and operations on each of them much easier to handle. The code is organized into small snippets to represent each separate part of the game, for example, all logic related to a single PlayingCard is in its own file, or all logic related to the Deck of cards is in its own file.