Blackjack Design Document

Section 1 - Project Description

1.1 Project

Simple BlackJack Game

1.2 Description

This game is a simple implementation of the Blackjack game. It follows many of the traditional rules of Blackjack, (hit, stand, winning and losing, special treatment of Aces, etc.) while leaving out some of the deemed less important qualities (betting, splitting). This is due to the focus of the game on the relative strengths of competing strategies. This version of Blackjack is a single-player game. Players will play against one computer, which depending on the difficulty selected, will feature a different competitive strategy of Blackjack. The difficulty increase - the lowest being one and the highest 3 - will increase the complexity and sophistication of the strategy.

Contents

Section 2 - Overview

2.1 Purpose

This project focuses is on the various types of competitive strategies for twenty-one a.k.a. Blackjack.

2.2 Rules

The aim of the game is simply to gain a higher total hand total than the computer, without going over 21.

When the game starts, you will be prompted to choose a difficulty level for the computer. Afterward, you and the computer will be dealt a hand.

Both players (computer and player) will start with two cards in their respective hands. The cards of the computer will start face-down.

A **hand** refers to the player's cards. A **hand count** refers to the number of cards in a respective player's hand. A **hand total** refers to the total sum of all individual card values within a respective player's hand. Face cards count as 10, and Ace counts as either 1 or 11. The game will calculate which value to use as most advantageous.

A **round** is defined as when both player and computer decide whether to hit or stand.

Hitting means the respective player decides to draw one card from the deck and add to their hand. The card will add to your hand total. **Standing** means a player decides to abstain from hitting.

The game ends when either both players decide to stand, when any player first receives a hand total of 21, or if any player busts.

A player **busts** if they receive a hand total over 21. At that point, the other player automatically wins. If the player ties with the computer, the computer wins (the computer acts as house in this situation and based on traditional rules of Blackjack, the house wins).

Depending on the **difficulty** the player selects, ranging from 1 to 3, the computer will have a different strategy:

(1) if the computer's hand total is less than 16, he will continue to hit, stopping immediately if the computer's hand total reaches or goes over 16;
(2) if the computer's hand total has exactly 12 or less than 12 points and the player's hand count is less than 4 cards, then the computer hits;
(3) if the computer's hand total is 16 or above and the player's hand total is 10 or less, stand. If the computer's hand total is 6 or more behind the player, the computer will hit.

2.3 How To Play

To be install the game, you must have a pipenv and python installed (see below if not installed):

Pipenv installation command:

```
pipenv install --dev --python=/Users/<username>/.pyenv/versions/3.6.5/bin/python
```

git clone [github project link]
cd blackjack
pipenv install

To play the game, you can follow the following commands:

Pipenv shell
Python -m blackjack

When you start the game, you will be prompted to type in and enter a number between 1 and 3 for the difficulty of the game.

When you play the game, you will be prompted via user input whether you are to hit or stand.

Section 3 - Software Design

3.1 Software Domain Design

The software domains used for this application are python (as the language and virtual environment of choice)

3.2.1 Terminal Playing Cards

I used **pwildenhain**'s framework for building cards within the terminal as a basis of how the cards would appear. The library features a basis for the look and feel of playing cards within a terminal. It also defines the basis of a deck variable. One of the hardest parts of building a terminal card game is that it takes a long time to get to the - in my opinion - fun parts: building the game logic and strategies, because one can easily get caught up in how to construct a deck (deal cards and values randomly) and display the visual UI of the cards.

3.2.1.1 Deck Variable

The UI provided is useful and the deck variable is useful. The deck variable allowed me to skip past the dealing of random cards (which while necessary in constructing a card game is mostly tedious) and move toward constructing roles, such as the hand, the player, and the values.

3.2.1.1.1 Visual UI

The given visual UI allowed me to use a good functioning hand UI without having to reconstruct one from scratch myself.

3.2 Algorithmic Design

Algorithmically, the design of the basic functionality of the game (hitting, standing, dealing hands and calculating sums) was abstracted by the roles I had constructed. I used python and pipenv within the virtual environment.

3.2.1 Main, Roles

Within **Main** is the functionality of the game flow. The main constructs how the game starts (beginner's message), how a user wins (the messages and calculation if the user wins), the general game logic of how the game is played (as long as both players haven't won/reached/gone over 21), and user decisions (users type in and enter 's' or 'h' to stand or hit).

This allows me to abstract through other functions, such as player_hit and should_play more complex functions.

Player_hit goes through the logic of the player choosing to hit. **Should_play** is exclusive to the computer, deciding if the computer (based on the strategies explained above in "Rules") should continue to hit or stand. This is defined circularly through a variable that is true if to play and false if not.

Within **Roles** is the functionality for hitting and hand/card values. For instance, it declares the cases for whether to decide an Ace as either 11 or 1. It also finds the value of the hand of the player and does the general arithmetic for finding out the value of a hand before and after a hit.

The **hand** variable within Roles has various attributes: defining value (through the function .value), hit, and initialization. It is a semi-stack like structure that allows users to view the hands, hit variables, and find value. Since this is a large part of the functionality of Blackjack, abstracting it within one variable that stores most of the data in the game allows main to be less cluttered and focus on game decisions.

The hand variable encapsulates all the data needed to play a general card game, which is why I implemented and extended implementation of it to account for all logistical parts of a card game, while focusing main on Blackjack-specific components.

3.2.2 Data Structures, Storage

All Data is stored locally. This is to ensure the game runs as quickly as possible. In order to lessen the clutter, all storage/data structures used are within the implemented hand variable, which acts as a sort of semi-stack structure. Other than that, there are no other data structures implemented.