Project 4

Title

Blackjack

Course

CIS-7

Section

27168

Due Date

December 16th, 2021

Author

Matthew Mickey

1. Introduction

The chosen project, Blackjack is a common game found at Casinos. Skipping through the history of Blackjack, I have personally barely played it a handful of times, so at best, I'm only slightly familiar with the game. However, when I did play it, I never heard of splitting hands or doubling.

Regardless, as a warning, this Blackjack game is incomplete as I ran into problems attempting to create it. It is intended to be a CLI/terminal based game with extremely simple game logic, in fact, barebones.

1a. Proposal

This game will attempt to incorporate some concepts found in the CIS-7 course. Due to working on this project solo and reviewing some of the requirements for this project when working solo (e.g. limited things needed to be added, etc.), I have elected to focus on the following:

* Summation - Specifically vector summation (inbuilt functions, etc.)
* Probability - Chances calculation

This program will be a command-line interface game that one person can play,
solo.

2. Game flow

The game is intended to work as follows:

Run the game, immediately start with the option to hit or stand. Then it will be the dealer's turn to start against the player. Depending on the score, a victor will be determined. The game will then exit after this.

3. Development Approach

Goal

The goal of this project was to solve a few problems.

1. Figure out how to implement some of the abstract discrete structure concepts within this game.

2. Attempt to have a MVP/prototype that will not seg fault or crash.

3. Attempt to improvise and turn flowchart & psuedocode into working code.

Structs

I personally have a problem with Object Oriented Programming, as I elect and prefer the functional/modular approach. However...I feel that classes or structs at a bare minimum should be used. There are two structs:

There will be seven classes:

1. **Cards** - Holds the types of cards; number, face, ace, and their amounts.
2. **Entity*** - Holds the score and cards held of the entity.

**(it is noted that this struct, and many others are defunct due problems that I encountered, mentioned later on)*

4. Development Summary

Total lines: **300+**
Lines of code: **146**
Useful Comment lines: **52**

Background on line count

I try to be as upfront as possible, so a proper breakdown is always done for these projects. There were too many deleted & wasteful lines, so the only ones counted were ones that were actively used in the program when turned in on Canvas. However, bracket lines are counted but not spaces.

Setbacks

There were too many problems and setbacks that I encountered, mostly due to programming implementation and personal problems in real life. However, the latter is not relevant to this document and will be omitted.

Programming implementation

I attempted to use object oriented programming as this is something that I need to improve on, however, this was a terrible decision since I cannot manipulate data the way I intend to. In essence, I created spaghetti code that sort of works as a proof of concept, but not in practice.

Connecting things together

I had too many moving parts at one point a couple weeks ago before submitting this project. Passing way to many pointers, things by reference, etc. without accurately keeping track of what was going where. This led to scrapping and re-writting the project too many times to count. However, some basic functionality such as passing by reference & pointers for vectors without using structs was accomplished.

Limitations

The limitations of the program are numerous.

For one, it's not a complete program or game.

Two, it's limited by my choice of electing a more simpler approach to the card suits and pooling them all together. I think if I had to do this again, I would seperate them, possibly using an enum to make it easier to reference. There is also the fact that I did not include splitting or doubling as I did not realize/forgot that those mechanics in Blackjack existed, which is completely my fault (despite it being in the specs).

Three, it's not really technically playable.

Four, there is no input.

Five, there is no AI.

Six, there is no probability calculation. I did have a probability function checker to see what was the chance to get an Ace from the current hand, but scrapped it a few restarts ago.

Improvements

To start off, I should have focused on finishing the game. I tried to over engineer it and my skills are not up to task without blatantly copying other code from outside resources. Instead, if I were to attempt this again, I would either stick with purely object oriented programming, or completely stick with functional programming with no structs/classes of any kind.

Sticking with my flow chart & psuedocode would have been better, as without sticking to it makes the whole point of making it moot. An easier solution for a needed discrete structure implementation would be just putting in binary search for cards. If I had separated the cards into 52 different variables (in an array or not, enum, etc.), searching through it would have been an easier concept to implement. Breaking them down into strings, then parsing the strings with a set value would have made summation and keeping track of cards simpler overall in the end.

Instead of worrying about so much, fixing scope creep would have been the most prudent thing to do. If sticking with a functional approach, implementing one bite size function at a time would have lead to more progress and less stalling.

I should have utilized git. Instead of completely commenting out sections of code, deleting and restarting, and other silly things, I could have just used git to manage my versions. This way, it would have been trivial to roll back or branch if I felt like x was a better direction to take the program than y.

Musings

For a simple game, I found it difficult to implement, possibly due to attempting to over-engineer it, when the game could be technically written in less than 150 lines of code, as a complete MVP.

Project time breakdown:

- **34 hours** for actual programming
- **3 hours** for flow charting & UML
- **2 hours** for planning and designing scope
- **17+ hours** for debugging
- **3 hours** for writing documentation

Total: ***59 hours***