**Project 2**

**<BlackJack>**

CSC-17A

Jimmy Munoz

12/16/2022

**Introduction**

This project is based on the previous one I created in Project 1, which was BlackJack. Although it is the same idea, the code is very different. For Project 2, we were required to use classes instead of structures. We were also told to apply different concepts involving classes, such as inheritance, aggregation, polymorphism, and abstract.

**Version 1 Summary**

The first thing I did for this project was to convert the structures I created in Project 1 into classes. So, I converted the “Player”, “Dealer”, “Stats”, and “Deck” structures into classes. I also aggregated the “Deck” class to the “Dealer” class since the dealer “has-a” deck. I had many functions that involved dealing cards from the dealer, so I transferred those functions to the “Dealer” class. Also, I have the “Dealer” class be inherited by the “Player” class because the dealer “is-a” player. Same as the “Dealer” class, the “Player” class has functions that originate from Project 1, such as assigning an Ace, drawing cards, etc. The “Stats” class is aggregated to the “Player” class because the player “has-a” name and sum. I also copied the main function from Project 1 over to Project 2 as a template on what to do. The “rules” function remains the same, but I added a few adjustments to it, such as a multiplayer set of rules. I am planning on using an array of players so that I can have multiple players play. I also revised some old rules to make it look more organized and clearer to read. This concludes Version 1.

**Version 2 Summary**

For this version, I created the .cpp files for the “Player”, “Dealer”, and “Deck” classes. As stated before, I transferred over the old functions from the previous project into this one. One thing that I mainly wanted to utilize was the STL features. So, I included the algorithm file so that I can use the “random\_shuffle” and “swap” functions. I first used the random shuffle function to shuffle the deck of cards, and after a few trials I realized that the deck is always the same when shuffled. So, I added the swap function to swap a few cards during the shuffle. With this, it allowed me to have a fully shuffled deck. For the “Player” class, transferred a function that will handle the Ace inputs. For example, I know that an Ace has a value of either one or eleven. If a player draws an Ace, they will have to type in the value they want to give it, so they will enter either one or eleven. I created a member function for the “Player” class that handles those decisions. If the value entered is one or eleven, the value has been set successfully. If not, then a “WrongAce” class will be thrown. These are a few examples of the changes I made when I transferred over the functions. This concludes Version 2.

**Version 3 Summary**

For this version, I transferred over the functions that handle the player’s turn, the dealer’s turn, and the results of the game. One of the changes I made was changing the name of the structures into the name of the classes I am utilizing, and then adding their specified functions. I also overloaded three operators for the “Player” class: the “<<” operator, the “>” operator, and the "==” operator. For the “<<” operator, I am outputting the player’s cards and their totals. I also have an if/else-if ladder that is used to handle if an Ace has been dawn. For the “>” operator, I am using it to compare the sums of the player and the dealer. Like the previous, the “==” operator is used to compare two different player’s sums to see if they are the same. This is used to see who had the highest sum out of all the players. I was starting to run out of ideas on how to add features to this project since we are required to have 1000+ lines of code. After a bit of thinking, I decided that I wanted to create different game modes. One mode I wanted to add is to simulate a game between an AI and the dealer, which is also an AI. The other mode I wanted to create is a “bet” mode, which allows players to bet on a game between an AI and the dealer. With these ideas in mind, I first wanted to create the AI mode. This concludes Version 3.

**Version 4 Summary**

As mentioned previously, I wanted to create the AI mode and the “bet” mode. To begin with the AI mode, I wanted to create an AI class, so I named this class “AIPlayer.” Since the AI “is-a” player, I have it inherited from the “Player” class. After this, not much has changed. This concludes Version 5.

**Version 5 Summary**

For this version, I created a function that handles all the AI’s inputs. Since I am already using an AI for the dealer, I am re-using that same algorithm to implement the AI. I created a separate “results” function that handles the results between the AI and the dealer. I also have the user decide if they want to enter a name for the AI. If they do, that name will be set to the AI via a copy constructor. Else, the default constructor will set it to a default name already set. This concludes Version 5.

**Version 6 Summary**

For this version, I started to create the “bet” mode for the project. My first step was to create two more classes, the “Money” and “Gambler” classes. I created these as templates, since I want the user to enter their bets either in integers or whole numbers. I have the “Gambler” class inherit the “Money” class and the “Money” class is an abstract class, since it is never instantiated. For the “Gambler” class, I created new functions that handle the betting. I have a function that sets a bet, a function that will take away your total by the bet if you lose, a function that adds the bet to your total if you win, and another function that handles the choice they make. This concludes Version 6.

**Version 7 Summary (Final)**

For the last version, I started to create the “bet” mode. The rules are simple. All players will start with $500. An AI will play against the dealer and before that game starts, the players will enter their choice and their bet. If one player chooses the correct guess, their bet will be added to their total. For example, if I chose that the AI would win and bet $400, and the AI wins, my new total would be $900. If the AI were to lose however, my new total would be $100. If the game ended in a draw, I will not lose my money so my total will still be $500. So, after all players enter their bets, the AI will play a game against the dealer and the result of the round will be displayed. A message will be displayed on the screen that will ask the user if they want to play another round. If they enter yes, the whole process will loop again. Else, the mode will end, and the final scores will be displayed. I made some final touches to the project such as formatting output and adding comments. This concludes Version 7 and my project.

**CheckList**

The checklist will be found on the next page.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Chapter | Section | Topic | Line #’s | Points | Notes |
| 13 | Classes |  |  |  |  |
|  | 1 to 3 | Instance of a class | 274 | 4 | In main: Instance of a dynamically created class |
|  | 4 | Private data members | In Gambler.h: 9 - 11 | 4 |  |
|  | 5 | Specification vs. Implementation |  | 4 | I have a .cpp file for each .h file |
|  | 6 | Inline | 14 - 20 | 4 | In Gambler.h:  The functions on these lines have inline members |
|  | 7,8,10 | Constructors | 14 | 4 | In Player.h |
|  | 9 | Destructors | 17 | 4 | In Player.h |
|  | 12 | Arrays of Objects | 270 – 275 | 4 | In main:  Process of dynamically created an array of objects |
|  | 16 | UML |  | 4 | Found in the folder |
|  |  |  |  |  |  |
| 14 |  | More About Classes |  |  |  |
|  | 1 | Static | 10 || 4, 12 | 5 | In Dealer.h:  Line 10: Static variable created  In Dealer.cpp  Line 4: Static Variable initialized  Line 12: Variable incremented, keeps |
|  | 2 | Friends |  | 0 |  |
|  | 4 | Copy Constructors | 7 - 8 | 5 | In AIPlayer.h  Line 7: Default destructor  Line 8: Copy constructor |
|  | 5 | Operator Overloading | 6, 25, 26  17 – 75, 91 – 94, 96 – 99  479, 611, 660 | 8 | In Player.h  Line 6: << overloaded  Line 25: > overloaded  Line 26: == overloaded  In Player.cpp  Lines 17 – 75: << function definition  Lines 91 – 94: > function definition  Lines 96 – 99: == function definition  Line 479: Use of <<  Line 611: Use of >  Line 660: Use of == |
|  | 7 | Aggregation | 12 | 6 | In Player.h  Line 12: Stats class is instantiated |
|  |  |  |  |  |  |
| 15 |  | Inheritance |  |  |  |
|  | 1 | Protected members | 8 – 12 | 6 | In Player.h  Protected members |
|  | 2 - 5 | Base class to derived | 7 | 6 | In Dealer.h  Dealer inherits the Player class |
|  | 6 | Polymorphic associations | 17 | 6 | In Player.h  Virtual destructor |
|  | 7 | Abstract classes | 7 | 6 | In Gambler.h  Gambler inherits the Money class  Money class is never instantiated |
|  |  |  |  |  |  |
| 16 |  |  |  |  |  |
|  | 1 | Exceptions | 55 – 65  85 – 89 | 6 | In Player.cpp  Lines 55 – 65: Try/catch statements  Lines 85 – 89: Throw conditions |
|  | 2 - 4 | Templates | 4, 7 | 6 | In Money.h  Line 4: Template class initialized  Line 7: Template variable initialized |
|  | 5 | STL | 2, 9, 17, 21 | 6 | In Deck.cpp  Line 2: Include algorithm library  Line 9: push\_back function used  Line 17: random\_shuffle function used  Line 21: swap function used |
|  |  |  |  |  |  |
|  | Sum |  |  | 100 |  |