Gil Abarca Ramos

Spandan

Alex Lujan

John Allen

CMPE 135 – Object Oriented Analysis and Design

November 22, 2022

Blackjack Project

**Overview**

Blackjack also known as Twenty-One is a casino game that consists in using a deck of 52 cards. The objective of the game is for the player to beat the dealer by getting get as close as possible to the number 21 without going over. Our game was design to follow the standard rules of the game. First, the user will have a max of $1000 to bet and can increase their bet by $5 increments. An ace worth is up to the player it can chose to have the value of 1 or 11 at their best interest. Once the player has put in his/her bet the game will start and the dealer in this case our program will give two cards face up and the dealer will have one card face up and the other face down. Based on what the sum of their cards is they can decide to either stand (not ask for another card) or hit (ask for another card). It is known as “bust” when the player goes over the desired number 21. The dealer has a limit of summation of 17 to not take a new card if his summation is 16 or below it can hit. Whoever is closer to 21 or gets the number is the winner and receives the amount they betted and an equal amount.

Polygon

Description automatically generated with medium confidence

**Functional Requirements**

* The application will allow the player to start the game.
* The application will allow the player to end the game.
* The application will be able to shuffle its deck.
* The application will allow the player to see its current hand.
* The application will allow the player to add a card to its hand.
* The application will allow the player to place a bet.
* The application will allow the player to see its balance.
* The application will allow the player to see the winner of the round.
* The application will be able to add or take away betting cash to the player’s balance.
* The application will be able to show the current record of the player.
* The application will be able to show the current summation of player’s and dealers’ cards.
* The application will be able to show if the player surpassed the desired value.
* The application will be able to show if the dealer surpassed the desired value.
* The application will be able to show if there was a blackjack from either player.

**Nonfunctional Requirements**

* The application will display cards within 2ms.
* The application will be able to run in Linux, Windows, and Mac.
* The application will not store any data once the user exits the application.
* The application will have an effective and organized graphical user interface.

**UML DIAGRAMS**

**UML Class Diagram**

**Diagram, schematic

Description automatically generated**

**Description:**

The class diagram provided shows how our systems is organized and what interactions the game may have:

* The game “has a” player and a Dealer the game cannot exist without both
* The player “is a” human
* The dealer “is a” human
* The human “has a” card
* The Dealer “has a” deck

**Sequence Diagram**

**Chart

Description automatically generated with low confidence**

**Description:**

The sequence diagram provides an overview of the flow of the player and its interactions with different classes used to complete our project. As you can see the flow is simple and it has alternative path depending on which choice the player makes.

**State Diagram for Player**

**Diagram

Description automatically generated**

**Description:**

This is the state diagram for the player, and it will only have 4 states.

|  |  |  |
| --- | --- | --- |
| Current State | Input | Next State |
| S0 | X | S1 |
| S1 | Hit | S2 |
| S2 | Hit | S2 |
| S2 | Stand | S3 |
| S3 | X | S0 |

**OBJECT ORIENTED DESIGN PRINCIPLES**

**Encapsulation**

The purpose of encapsulation is to hide sensitive information from users, and they can only be modified or read by using setter and getter functions. For our project we decided to hide a few variables from the users. We decided to provide setters and getter functions for each class so that the game which controls the flow of information accesses the information through the getter functions. The game cannot access the users betting cash directly, it only uses a getter function to read the value. The game class only has access to the player and dealer objects and cannot access other classes like card, deck, or human.

**Cohesion**

This was a design principle used in our code since all the classes in our project only have one purpose. All our classes perform their own distinct actions that are independent from one to another. Dealer class only displays their cards and nothing else, our player class keeps track of our only takes care of recording wins/loses, name, bet, and their balance. These our methods that only the player can do. Card class only has methods that describe a card like their number, suit, mapping card method that will store the location of all card pictures, and a card info method that returns a string to the location of the image of the card that will be used to display the card in the GUI. Game class is the control of data flow of all the classes created in our project.

**Loose Coupling**

This was a design principle used in our code since all the classes in our project have few dependencies. The card class is independent from all the classes used in the project; however, they are used to build a deck class, a deck is vector made of card class. Player class is independent of the dealer class; however, both use methods of the human class because they both have a summation of cards, they both add a card, etc. All these classes were done so they work together as a system. This was done by coding a Game class that will control the outputs based on the other classes.

**Law of Demeter**

The Law of Demeter can be summarized with the phrase “only talk to your nearest friend”. Our game class only gets results from the player and dealer class and does not know anything about the deck, card, and human class. The player class does not depend on the dealer class. The dealer class does not depend on the player class. Both of these classes are used in the game class to get their results.

**DESIGN PATTERNS**

**Composite Pattern**

Composite pattern is when a group of objects are treated the same way as single instance of the same type of object. In our project we decided to follow this pattern because we had two objects that would essentially have the same type of function. Our project consisted of blackjack and in a blackjack game there are two humans the player and the dealer. These two players have essentially the same functions with only a couple of differences. They both have a sum of their cards that enables them to know where they are at in the game. Both players can switch their ace to be either 1 or 11 and finally both can add and clear their cards. In the image below it is shown that human object is using has all these functions.

**Human object:**

**Text

Description automatically generated**

In the image below you can see that our dealer and player object use the human methods. A player and dealer can clear their cards, add a card.

**Player and dealer using human object**

**Text

Description automatically generated**

**Instructions to build and run project**

This project was completed using Visual Studio Code in Ubuntu. To build and run this code, there is a serious of steps needed:

1. Assuming wxWidgets is already installed open the project in Visual Studio Code or a terminal
2. If you opened the project in Visual Studio Code, then open the terminal inside the editor. If you did open the terminal only then cd to the project folder
3. Enter the command “make run”

**Results**

Once we completed our project we went ahead and tested the system as a whole and was able to verify that our project worked well. All the requirements that the game has were able to be implemented in our project. The player can input their name, increase/decrease their bet once they are done making a bet, they can click done and start playing the game. Depending on what the player gets as their first two set of cards they can either hit (get a new card) or stand (end the game). If the player wins or loses it will be reflected in their cash balance. Once the round has been completed the player had the opportunity to either play a new game or exit. The images below show some of the rounds played by different members of our group.

**Dealer wins**

**A picture containing qr code

Description automatically generated**

**Player Wins**

**A picture containing qr code

Description automatically generated**

**Dealer Blackjack**

**A picture containing qr code

Description automatically generated**

**Player Blackjack**

**A picture containing calendar

Description automatically generated**