**Group: *Awooga***

Dean Shumbanhete g18s0463

James Thompson g22t1578

Nicholas Szczawinski g17s0395

Theresa Rautenbach g22r8334

***“21”***

***A simply advanced game of Blackjack***

**Computer Science 102 Project Document**

20 September 2022

(Requirements and Design)

Contents

[Requirements & Analysis 3](#_Toc114609266)

[General definition & Guidelines 3](#_Toc114609267)

[Deadlines 3](#_Toc114609268)

[What is required to make it work 3](#_Toc114609269)

[System Design 4](#_Toc114609270)

[Architecture (How) 4](#_Toc114609271)

[Class Diagrams 5](#_Toc114609272)

# Requirements & Analysis

## General definition & Guidelines

The central premise of the programme discussed in this chapter revolves around imitating an in-person game of Blackjack. Using a series of classes, the plan is focussed on generating a deck of cards which can be dealt to a player, with plans to include multiple players down the line, who will have the ability to place bets using an in-game currency system. Furthermore, and in compliance with Blackjack rules, said players will be able to ‘hold’ or ‘hit’. Specifically, the player has the opportunity to obtain more cards to their hand or choose to remain with the cards they have already been dealt. This process continues until the player’s hand sum surpasses a score of 21 whereupon they become ‘bust’, thus losing their bet, in turn removing them from the game cycle. Or, the player chooses to hold, in the hopes that their hand sum is higher than that of the dealer’s whilst being lower or equal to 21 at the same time, thus doubling the return on their bet and adding that value to their wallet. Further plans for the programme include a GUI to add flair and visualise the game’s function. It is planned that each component of the programme, from deck, to player will be encapsulated into its own class.

## Deadlines

Initial Project Proposal: 4/9/22

Design and Requirements: 21/9/22

Final: TBD

## What is required to make it work

- Any Windows64 system

- Capacity to download and install a setup package from the internet (initial)

The primary language for which this programme will be developed is Java, with heavy emphasis on encapsulation. Rationale behind these choices depend on the base concept of user-programme interaction, I.e Game > Deck > Hand > Card > Player and so on. In this fashion, the programme is expected to be clean, detailed, fun and interactive whilst being easy to maintain.

# System Design

## Architecture (How)

The primary vision of the design of the programme rests in the use of specific classes to subdivide operations and functions into workable parts. Six current (potential room for more) classes are seen as being integral to the design of the programme. Namely that of an overall Game class, card class, card collection class, a deck and hand, and lastly the player. At current, development of a dealer class is in question as it can fit within the player class with extra conditions.

The following descriptions of each class follow in order of broadest to smallest, as would be the case from a top-down perspective of determining what needs to be done (see the requirements chapter) and using those to determine the next design element. For example, you start with a deck, which gets drawn to a hand, which gets put in the hands of a player, who needs a way to circulate said cards, which need an attached value in order to play. In this sense, this design rationale follows a sequential format which addresses obvious problems first, and using those to discover smaller issues which need implementing or addressing.

The rationale behind each class rests in the ‘how’ in terms of the manner in which the programme would be designed to interact with the user. For example, there needs to be a deck/decks from which to draw cards for the players and dealer. This deck must act in such a way so as to be shuffle-able, and provide non-repeating values per card. Other attributes include a built in Shannon entropy for said shuffles.

From the deck, a hand is derived, which acts as a value holder of the drawn cards from which another class, the player, is able to pull. The operations which dictate the drawing of cards from the deck/hand is the card collection class, which acts as the intermediary function to circulate the values of cards in rotation amongst all players and the dealer without repetition (repeated cards).

Lastly, the card class itself associates values of the suite and rank to the card to make a unique value which can be drawn from a given deck, placed into a given hand, and whose value can be used to play the overall game (sum total < 21 etc.).

## Class Diagrams

|  |
| --- |
| Game |
| -deck : Deck  -dealer : Dealer  +PLAYER\_COUNT : int  -players : Player[] |
| +Game(int playerCnt, int numDecks, Player[] players)  -doGameCycle()  -firstDeal()  +playersToString()  +getDeck() : Deck  +getDealer() : Dealer  +getPlayer(int pID) : Player |

|  |
| --- |
| Card |
| -RANK : int  -SUIT : int  -RANKS : String[]  -SUITS : String[] |
| +Card(int rank, int suit)  +getValue() : int  +toString() : String |

|  |
| --- |
| CardCollection |
| -name : String  -cards : ArrayList<Card> |
| +CardCollection(String name)  +addCard(Card card)  +removeCard() : Card  +cardCount() : int  +getCard(int i) : Card  +getName() : String  +toString() : String |

|  |
| --- |
| Deck |
| -numDecks : int |
| +Deck(int n)  -populate()  +riffleShuffle()  +getShannonEntropy() : double  +dealCard(CardCollection hand) |

|  |
| --- |
| Hand |
|  |
| +Hand(String name)  +handValue() : int |

|  |
| --- |
| Player |
| -name : String  -hand : Hand  -balance : int |
| +Player(String name)  +getHand() : Hand  +toString() : String |

|  |
| --- |
| Dealer |
| *\*Potential class* |
| *Possibly redundant (see player class condition)* |