**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 Proposal**

10 October 2022

Contents

[Background and Motivation 3](#_Toc116424506)

[Rules of Blackjack 3](#_Toc116424507)

[History of Blackjack 3](#_Toc116424508)

[Why Blackjack? 4](#_Toc116424509)

[Problem Statement (Why and What) 4](#_Toc116424510)

[Aims 4](#_Toc116424511)

[Complexity 5](#_Toc116424512)

[Flow Chart 5](#_Toc116424513)

[Approach 6](#_Toc116424514)

[Milestones 6](#_Toc116424515)

[Timeline (Gantt Chart) 6](#_Toc116424516)

[SDLC 8](#_Toc116424517)

[Walkthrough / Journey (Proposal to project) 8](#_Toc116424518)

[Model 8](#_Toc116424519)

[Requirements and Analysis 9](#_Toc116424520)

[System Design 9](#_Toc116424521)

[Implementation 11](#_Toc116424522)

[Verification and Testing 12](#_Toc116424523)

[Deployment and Maintenance 12](#_Toc116424524)

[Visual Guide 13](#_Toc116424525)

[GUI 13](#_Toc116424526)

[Code 15](#_Toc116424527)

[List of References 20](#_Toc116424528)

# Background and Motivation

## Rules of Blackjack

The rules of Blackjack may vary depending on the location, however, certain fundamentals remain constant. Blackjack is not limited to a particular number of players, varying from tables of 2 to 9 who play against the house/dealer. The primary objective of the game is to obtain a hand of cards whose sum comes as close to and including the number 21 whilst being greater than that of the dealer’s (Bennis, 2004). A standard 52 card deck is used to draw the hands (Blanchard, 2019), with the dealer providing each player two cards face up and themselves a single card face up at the start of each round (Kendall & Smith, 2003).

Each player will have the chance to hit or stand. A hit means that the player requests another card for their hand whilst a stand means the player is satisfied with their hand and makes no further requests (Bennis, 2004). A player can hit as many times as they please until they reach a hand sum greater than 21 in which case they go bust and lose their bet, which is also applicable to the dealer. Once all players are satisfied with their hands, the dealer will draw their second card. If their hand sum is less than 17, they are required to draw again (Kendall & Smith, 2003). Players whose hand sums are greater than the dealer’s as well as being less than or equal to 21, win their bets, all others lose.

## History of Blackjack

Blackjack is a card game derived from a family of games called “21”, and remains one of the most widely played card games in the world. The earliest account of “21” appears in the Spanish text “*Rinconete y Cortadillo*” which details some card cheaters playing the game of *veintiuna* (“21”), implying that the origin of the game rests in the Iberian Peninsula. *Veintiuna* would eventually find its way to France and Britain in the 1800s under the convention of *Vingt-et-Un* or *Vingt-Un* meaning “21” (Parlett, 1990).

The name “Blackjack” was thought to have originally been established by American gold-rush prospectors in reference to the best bonus hand obtainable, that being an ace and any 10-point card. The reason for the convention is thought to have come from the fact that the mineral zincblende, also called ‘blackjack’, had been associated with silver and gold deposits by such prospectors, thus highlighting the ‘wealth’ nature of the find/hand (Depaulis, 2010).

## Why Blackjack?

Blackjack was chosen primarily for its simple premise yet layered structure. This meant that the code could be built using the object orientated principles learnt throughout the semester. Each layer of the game could be constructed into its own class, for example, a deck class, player class and so on. Deconstructing a single project into multiple smaller projects also showcased the overall philosophy of the group, that being one which focusses on assignment of task based on competence and confidence.

Each group member could independently contribute to the design of the game within their own manner after which it could be proofed and amended to fit the overall game. An example of this process would be assigning one member with an already working shuffler to provide their code to a method. Blackjack allowed for the re-use of code already completed as practical work, with sometimes minimal conversion necessary.

# Problem Statement (Why and What)

## Aims

The aim of the project was to create a functioning game of Blackjack with an integrated GUI. Although simple in premise, whence converted into code, Blackjack can become rather complex with the use of the prior mentioned classes. For example, designing a method in which to draw a card from a list or array without replacement and inserting that card into hand objects in the player class required a great manner of intuition.

The initial aim of the project was to create a game of Blackjack where a user could customize aspects of the rules to fit their own playstyle and concept of the game. As prior mentioned, Blackjack has slightly different rules depending on where it is played and who is playing it. Thus, the initial aim was to establish a working base game, with the different types of rules acting as parameters before the initialisation of the game.

The initial proposal highlighted the distinction between low and high complexity, where the former related to the base game rules, and the latter referred to the use of rule parameters in order to create a modular version of Blackjack per user specification. An example of a rule parameter would be to decide to play with soft/hard 17s, allowing for double downing or not, restricting dealer draws to occur at either the start or end of the rotation and so on. The central premise of those concepts was to add to the complexity of the game programme.

## Complexity

The base game programme turned out to be much more complex than what was originally anticipated as each class grew to accommodate the different aspects of the game that required function, such as being able to draw cards from lists without manipulating the overall deck such that duplicates could be created on each shuffle. The original concept did not take into account how particular the classes would need to be in order to work together, which further complicated the workflow of the group as each member had a particular coding style which did not fit into the broader programme structure as easily as expected.

The increased complexity of the base game meant that the overall design and premise of the game needed to be re-examined in a more realistic sense and the idea of distinguishing between low and high complexity as well as the ability to create a modular game with varying rules was dropped in favour of a default game of Blackjack. As a result, more time would be spent on the GUI thereafter to increase the attractiveness of the game.

## Flow Chart

Each player dealt two cards including dealer (one hidden)

Blackjack?

Yes

No

Sum under 21?

Sum under 17?

Yes

No

Yes

No

Bust

Hit

Stay

D:sum < player?

No

Yes

Lose

Win

# Approach

## Milestones

The first step of the project involved setting out which particular elements were needed to satisfy obvious parts of the game’s functionality, such as an ability to shuffle a deck, which types of objects would need to interact with each other and so on. Thus, one of the first milestones necessary was to establish a central game class which could lay the structure of the subsequent classes pushing and pulling their variables.

Subsequent milestones took the form of each class thereafter, such as creating a card class, a deck, hand, player and a class for collection. Each class acted as an independent milestone. After the general layout of each class had been established, an iterative process of ironing out took place to make each class function accordingly and without much fault, essentially interconnecting that which had already been independently created.

During the phase of getting each class to work with each other, the decision was made to shift focus away from the previous low and high complexity philosophies previously discussed in the problem statement. The idea to accommodate custom user rule parameters was dropped and additional milestones relating to a GUI system were incorporated. These milestones revolved around using java swing and comprised of setting up a menu and display for the user.

Administrative milestones included completion of the project proposal which was specifically designed to make the project document easier to adjust later on given potential changes in the design philosophy. After each attempt of a milestone, the SDLC would be updated on the timeline and appended to highlight any overshoot.

## Timeline (Gantt Chart)

The workload was split into blocks, each a 5-day week from Monday to Friday.

* Grey = Planned Schedule
* Cyan Blue = Complete on time
* Green = Ahead of Schedule (Gained days)
* Purple = Complete behind Schedule
* Red = Behind Schedule (Lost days)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Duration | | | | | |
| Task | Week 1 | Week 2 | | | Week 3 | Week 4 |
| Req. Analysis |  |  | | | | |
| Actual |  |  | | | | |
| Sys. Design |  |  | | |  | |
| Actual |  |  |  | |  | |
| Implementation |  | | | |  | |
| Actual |  | | | 02 Sept - Start | | |
| Verification |  | | | | | |
| Actual |  | | | | | |
| Deployment |  | | | | | |
| Actual |  | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Duration | | | | | | |
| Task | Week 5 | Week 6 | | Week 7 | Week 8 | | |
| Req. Analysis |  | | | | | | |
| Actual |  | | | | | | |
| Sys. Design |  | | | | | | |
| Actual | Restructuring | |  | | | | |
| Implementation |  | | |  | | | |
| Actual |  | | | 10 Oct - End |  | | |
| Verification |  | | |  |  | | |
| Actual |  | | | Days lost |  |  | |
| Deployment |  | | | |  | |  |
| Actual |  | | | |  |  |  |

**Planned Milestones**

1. Requirements Analysis: Week 1
   * 1. Blackjack flowchart into boolean logic
     2. Ability to generate deck of cards from which to draw without replacement
     3. Number and type of classes
2. System Design: Week 2
   * 1. Game class (calling and base)
     2. Card definition class
     3. Use of array list to represent card collection (subclass: deck and hands classes)
     4. GUI (Java Swing)
3. Implementation: Week 3-6
   * 1. Integrate classes, work on logic
4. Verification and Testing: Week 7
   * 1. Test boolean logic for game state
     2. GUI layout
5. Deployment: Week 8 (Monday and Tuesday)
   * 1. Group test/play (iron out obvious issues missed during testing)

# SDLC

## Walkthrough / Journey (Proposal to project)

The original concept was to create a game whose main premise was the ability to customize rules to user preference, using a base game of Blackjack as a foundation. The proposal distinguished between the terms of ‘low’ and ‘high’ complexity to differentiate the base game from the additional modular rule features. A guiding theme would be to use object orientated programming to turn each rule into its own object which would could be called via parameter input by the user when starting a new game.

Upon commencement of the project however, it was quickly realised that attempting to add more features as parameters would be far too complex and take much more time than originally expected. The decision was thus made to cut the additional rule features from the programme in exchange for a general game of Blackjack. The game itself, although now void of the additional parameters, proved to be a complex undertaking nonetheless.

As per the project timeline, it is highlighted how the new approach to the programme increased the time taken to complete the implementation phase thus eating into subsequent activities which had already been planned prior, a week and one day was lost due to said restructuring. Post restructuring, the programme took the form of an aide, to represent in virtual what would normally be present on a physical Blackjack table.

Original plans also called for a pot or currency system where players could bet funds, however this was also dropped as it was in the context of a full tabletop where players could see other player’s cards and so forth. Hence with the removal of the extra features for a more simplified programme, the currency system was also dropped. Overall, the work required to operate a Blackjack game with multiple players and conditions which although intuitive initially, is rather complex whence put into code.

## Model

The model used for the project was that of a Waterfall technique, the rationale behind which was due to the assumed simple nature of the project and sequential nature of progression. Each task could be split between members and focussed on until completion after which the next task would be undertaken. In this sense, the main premise was to make the project a very simple and defined activity separated into chunks. The chapters of the SDLC are described below in sequence.

## Requirements and Analysis

**General Definition and 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 who can then have options to hit or stand. 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.

**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.

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()  +dealCard(CardCollection hand) |

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

## Implementation

The programme was developed in independent pieces, with each piece revolving around a particular class as is seen in the system design chapter. The game class acts of the driver and infrastructure for the rest of the game programme and was also used to establish what was required in addition to the initial requirements analysis.

Secondly the strategy for collecting cards was established and revolved around providing ways in which to extract cards from the deck as well as provide cards. This class would be revisited as the subsequent hand and deck classes were under construction in order to provide more functionality that would only be identifiable later on down the line. The hand and deck classes would be revisited in a similar iterative fashion.

The benefit of working on each class as an initial base and then iteratively working on them per the requirements of other classes means that the process of stitching them together becomes much easier and as a result only small changes need to be made post fact. These small changes then get processed during the verification and testing chapter.

## Verification and Testing

After stitching the programme together into a coherent game, smaller details could be thought of such as making sure certain exceptions are not triggered or that players cannot do certain actions that do not make sense in the context of the rules of the game. This process also worked in close alignment with the GUI design process. The reason why the GUI and testing phases occurred sometimes simultaneously is because approaching the game from a player’s perspective highlighted obvious problems that would otherwise go unnoticed whence focussing solely on the code of the game. An example would be realising certain hand values do not add up when attempting to proceed through a round.

Overall, the testing phase revolved around finalising any remaining items of concern and making the programme playable for anyone. Examples of some of the areas of concern would be testing the GUI system to ensure it functioned without hassle (broken windows and alignment issues), testing the boolean logic used to determine the game progression, such as when to treat aces as ones or elevens as well as when to skip a player who is already bust from a rotation.

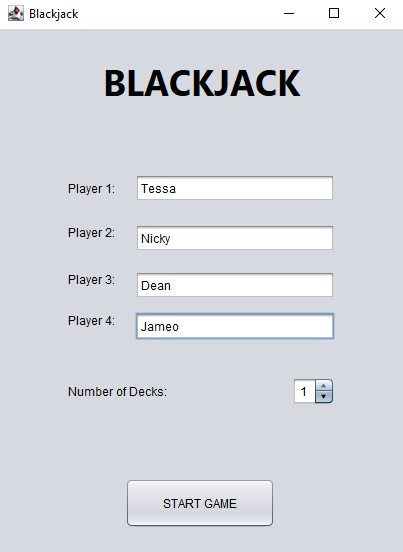
## Deployment and Maintenance

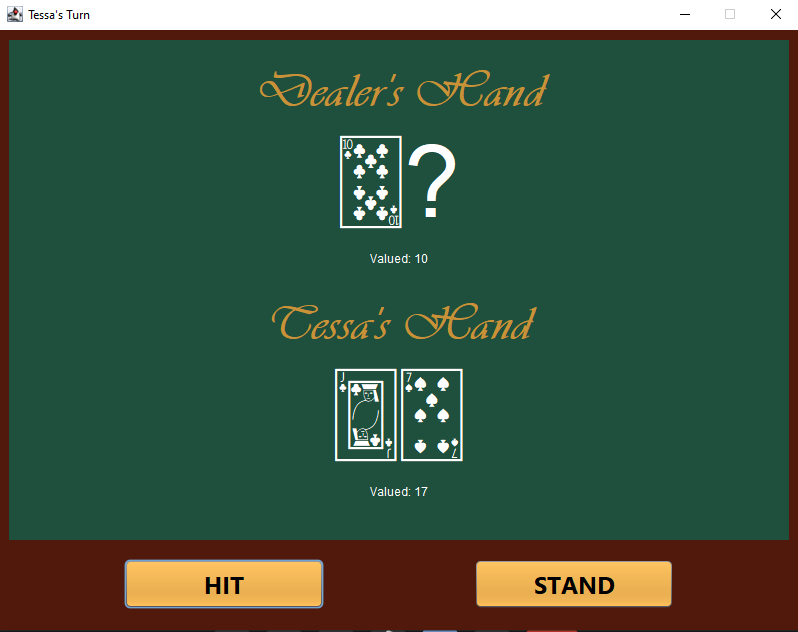
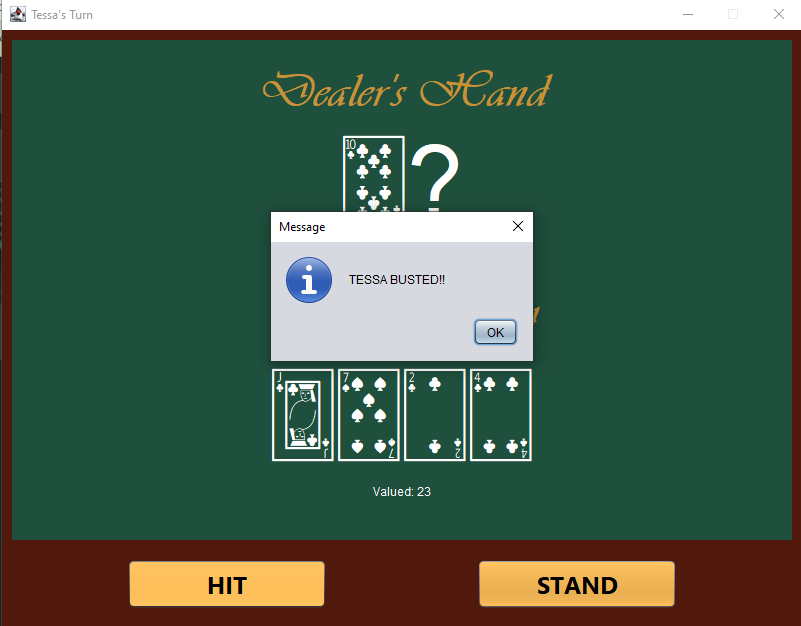
The final phase of the project revolved around running the programme between group members to check if any obvious problems detrimental to the game function persisted through the testing phase. The code had also been created in such a way to be modular to each assigned class i.e., a lot of emphasis had been put onto the object orientated nature of the programme and thus compartmentalisation, making future maintenance presumably easy.

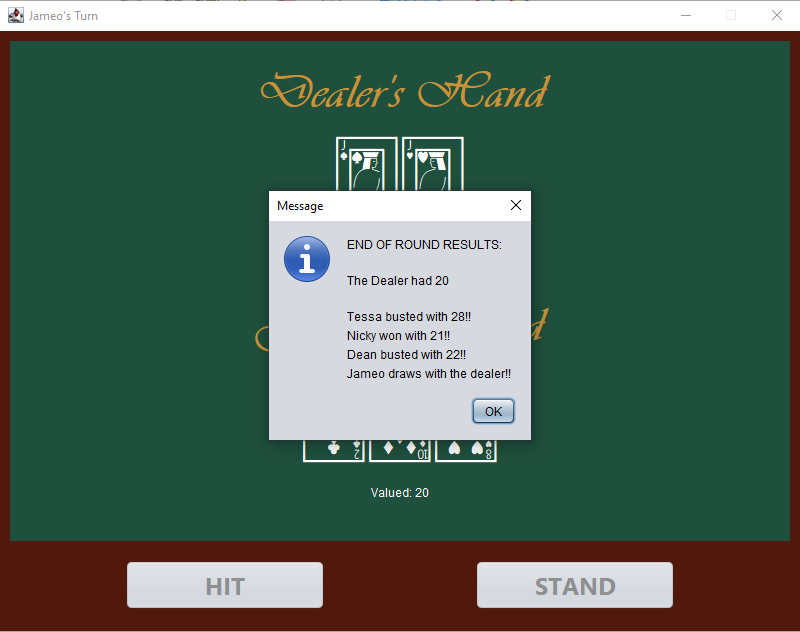
# Visual Guide

## GUI

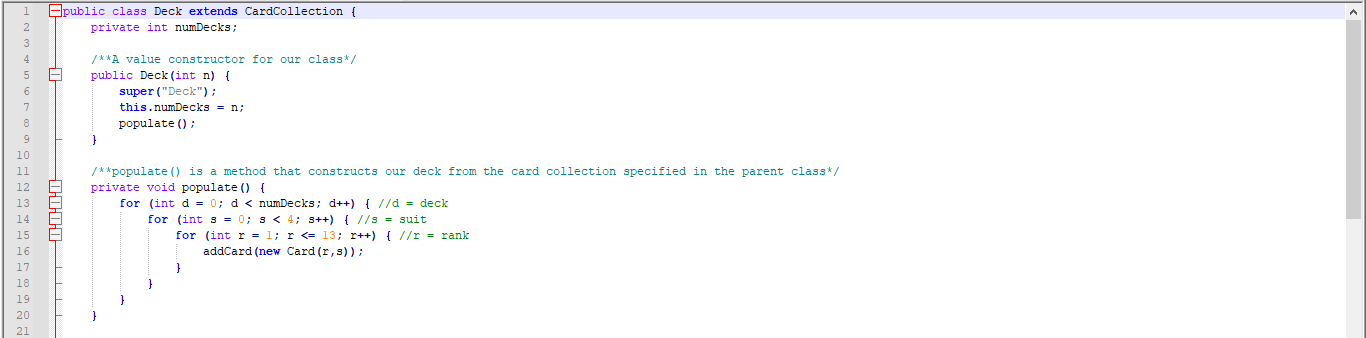
The programme opens with a main menu block which takes up to four players. The number of decks can also be altered to increase/decrease odds in each game. The game board details the current dealer’s hand as well as the player who sits in rotation. The question mark represents the dealer’s second card which remains hidden from all players until bets have been placed and a rotation has been made. Every player has the opportunity to hit or stay their hand during their turn. After a full rotation, the dealer’s second card is revealed, followed by a notice of winners and losers.

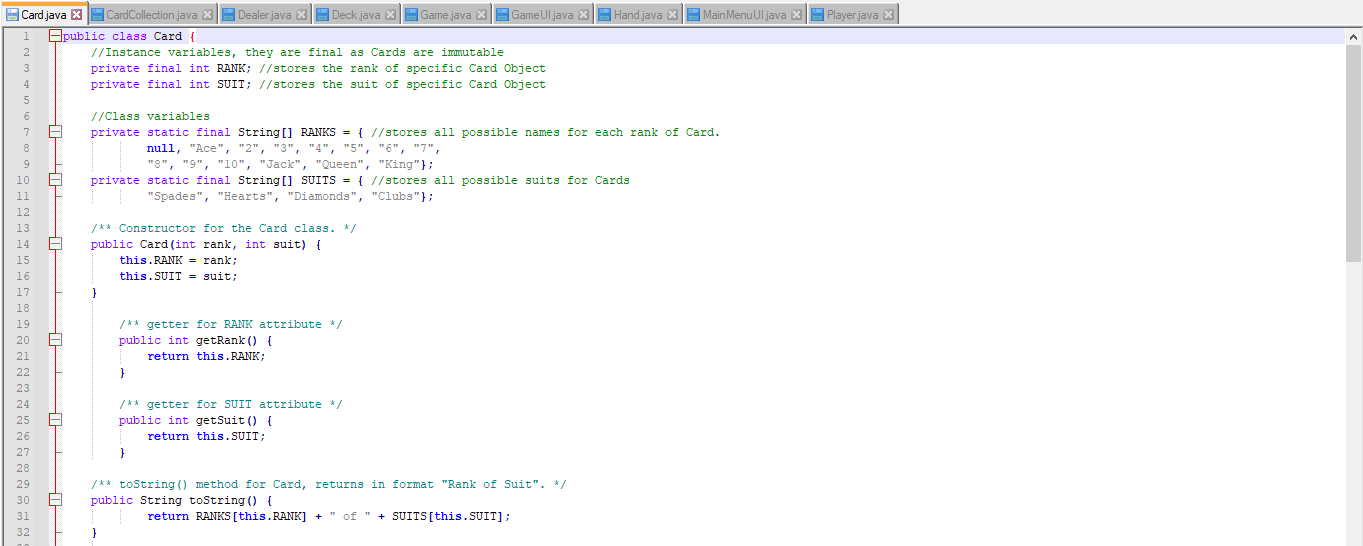


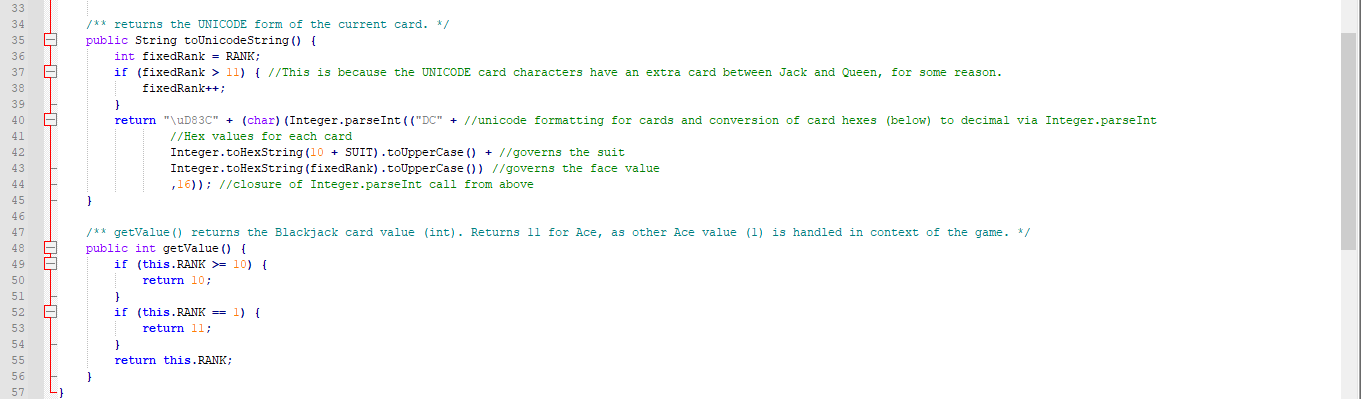


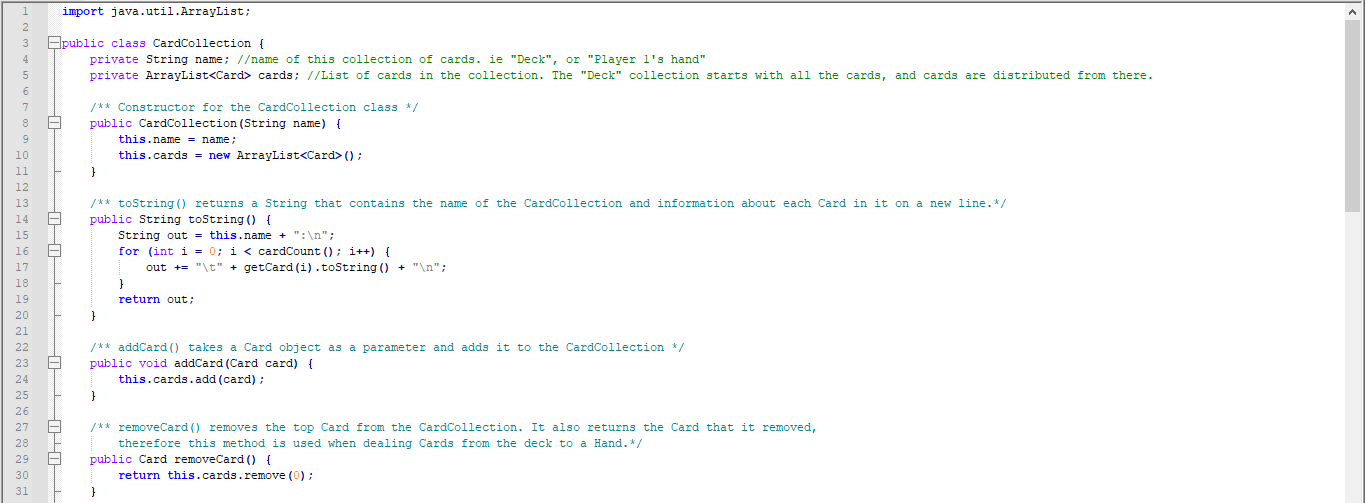


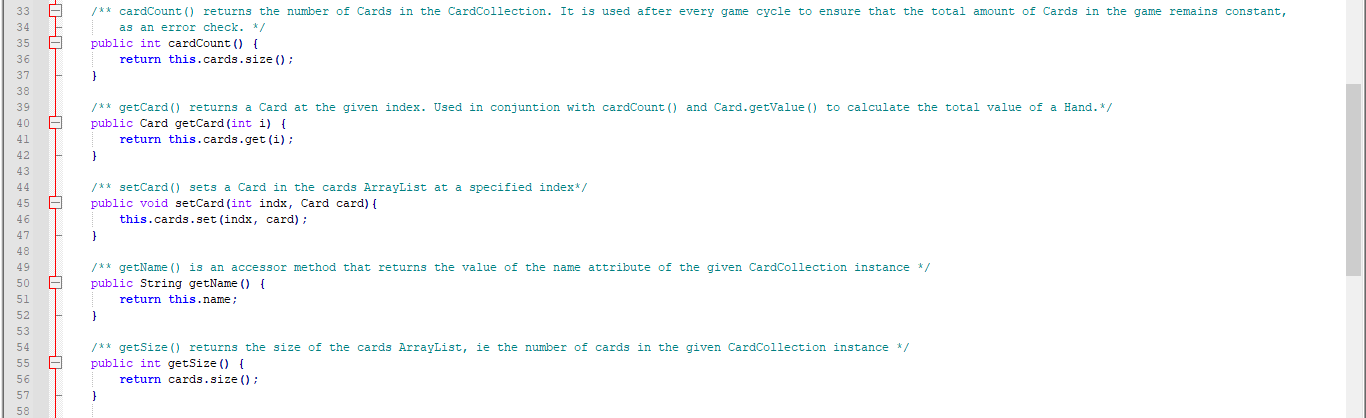
## Code

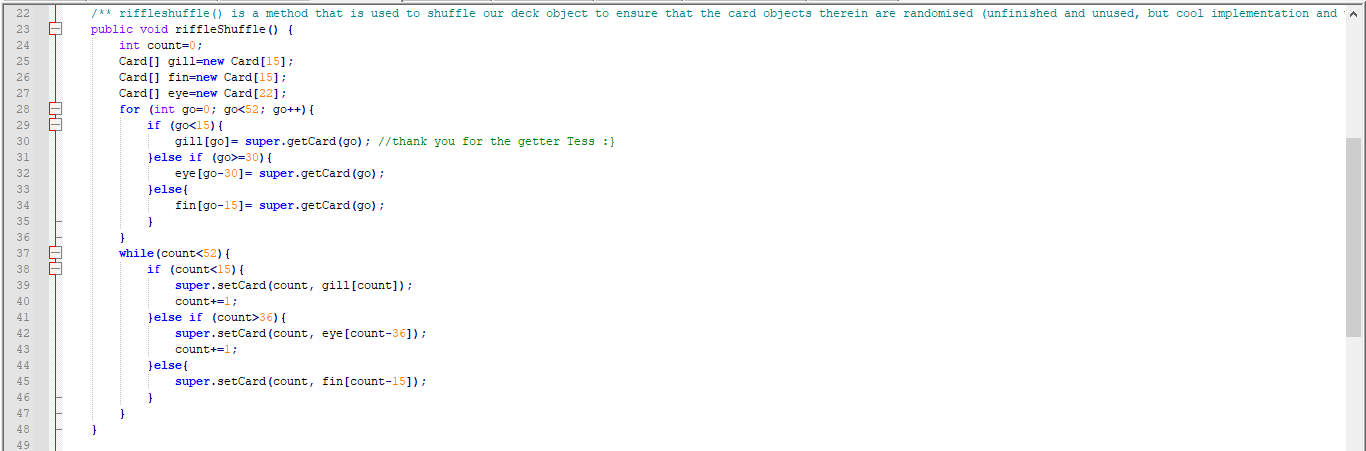
A defining theme of the structure of the programme is the compartmentalisation via object orientation principles learnt throughout the semester. Each factor identified during the requirements analysis process has been modulated into a respective class. The comments mostly describe the function of each block.

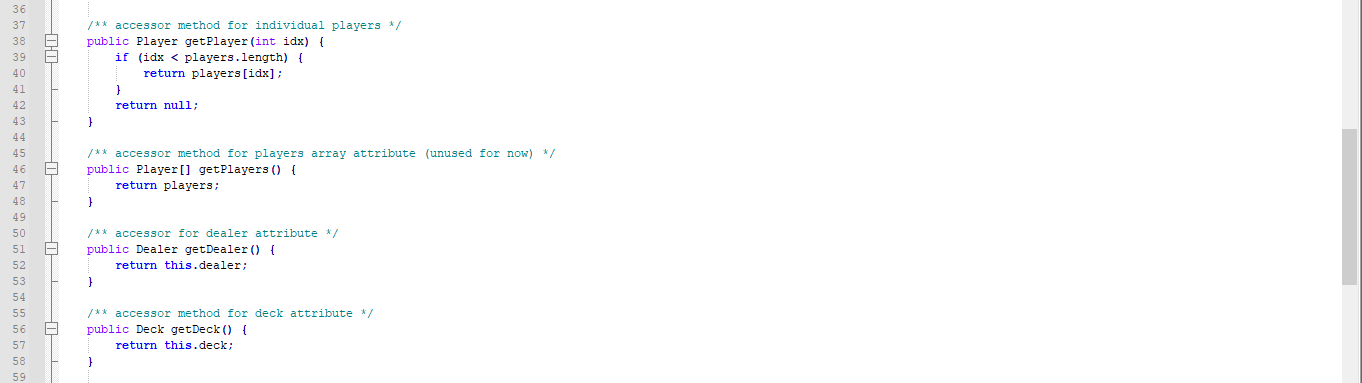
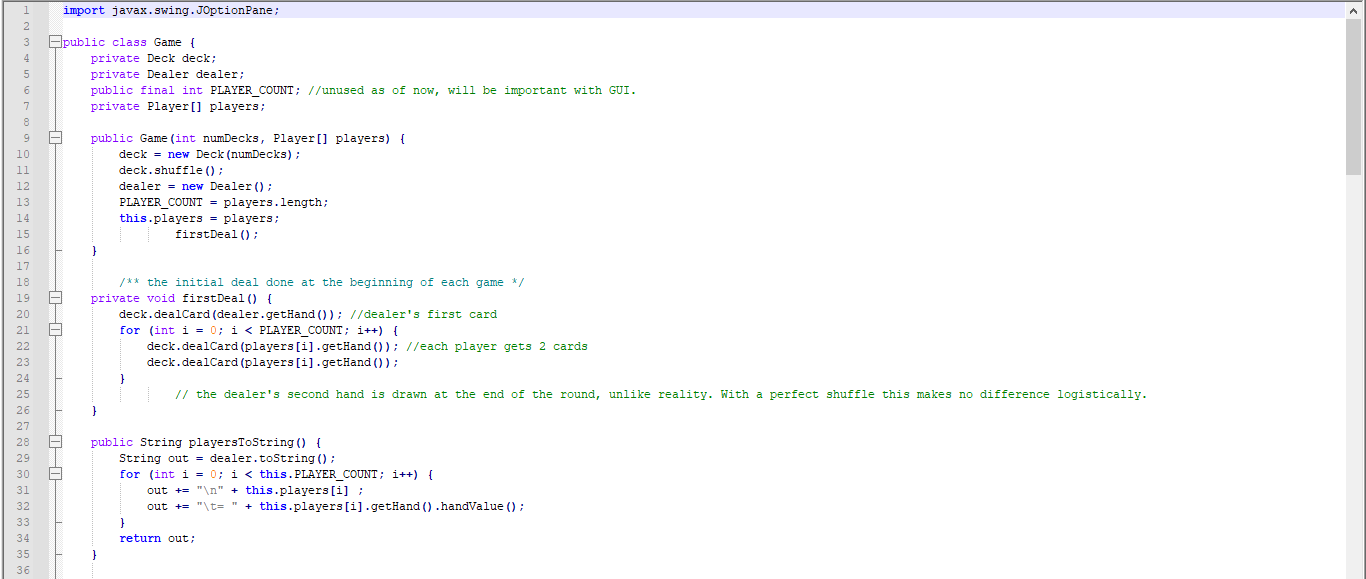


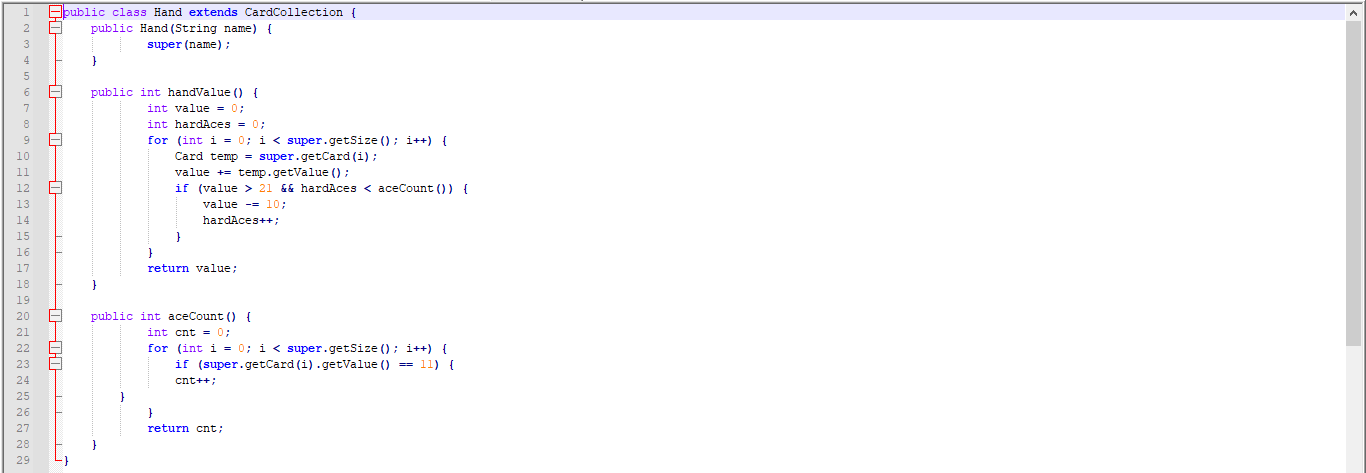
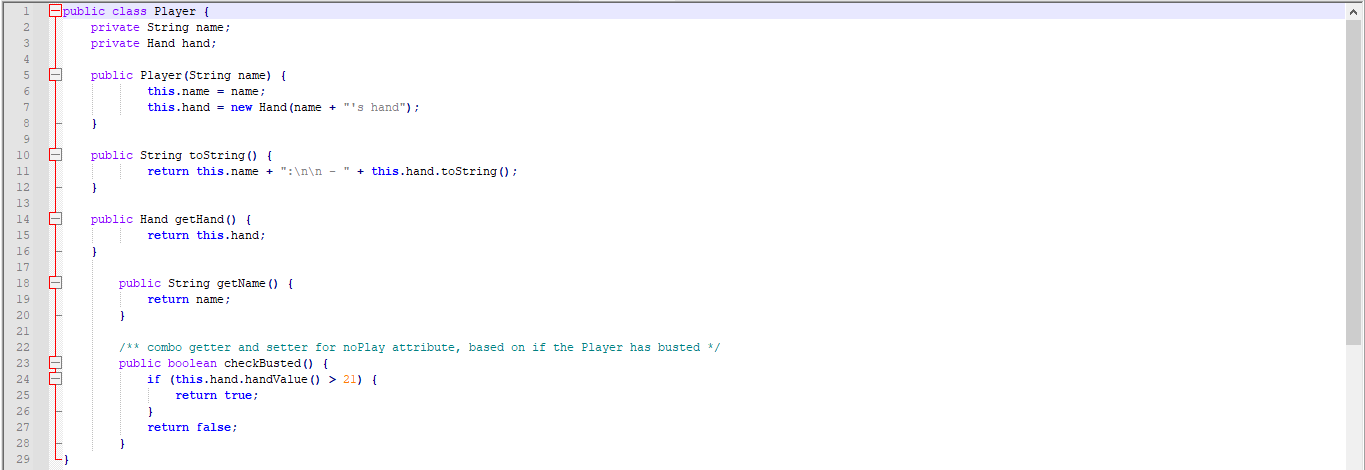
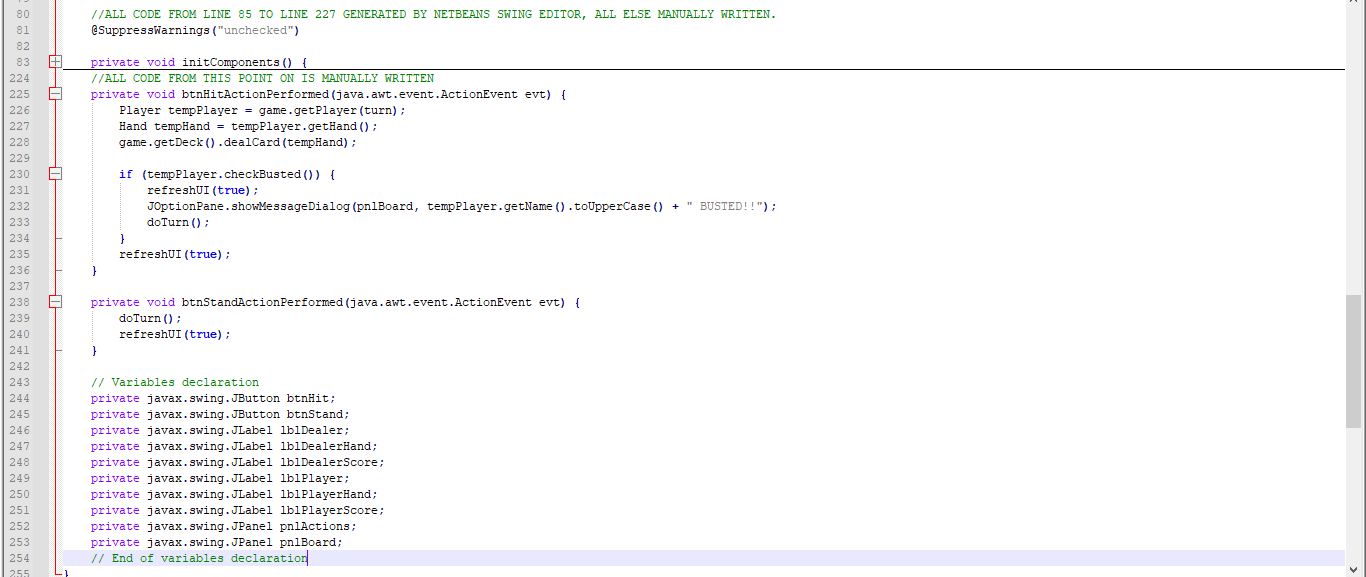
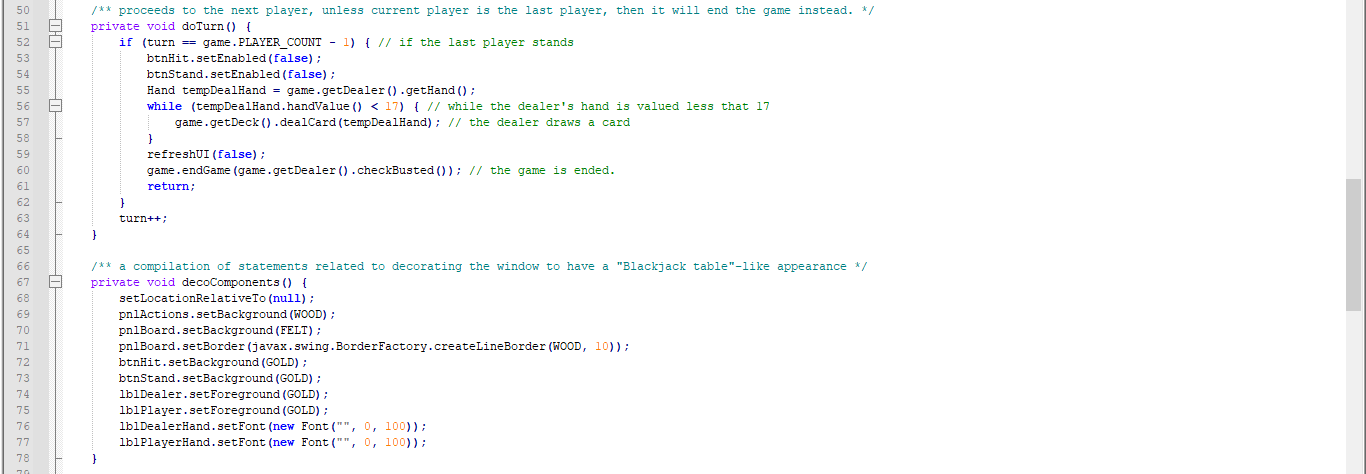
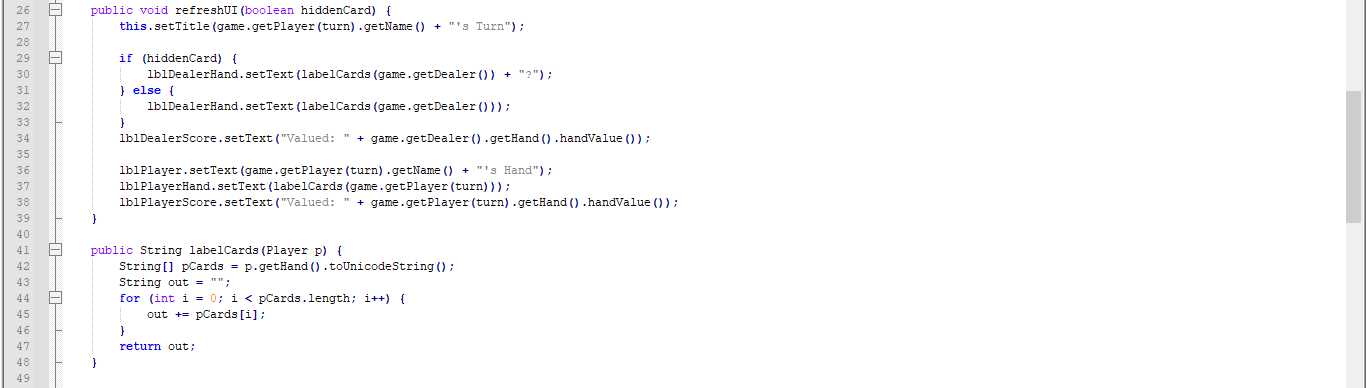
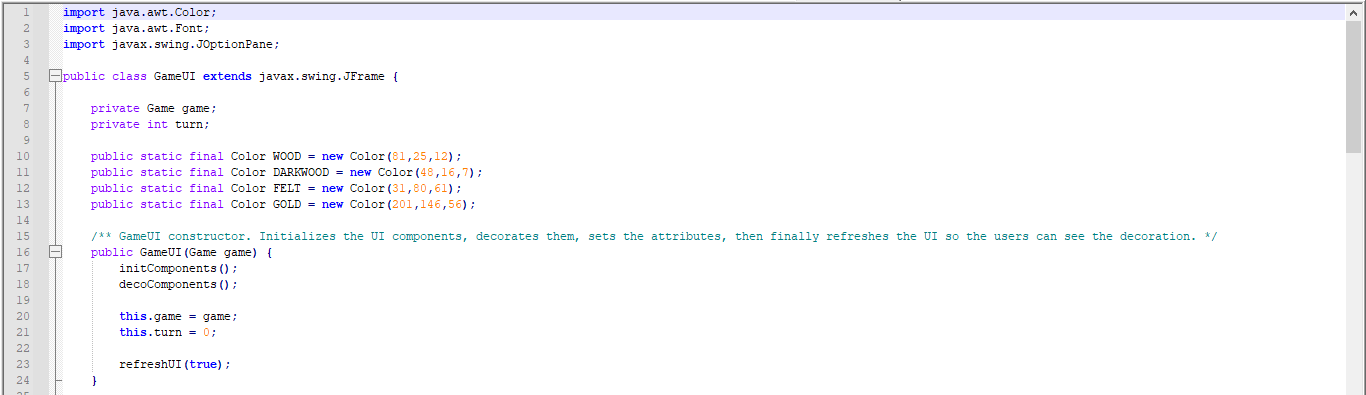












# List of References

Bennis, W. (2004). Blackjack playing strategies and beliefs: A view from the field. *Journal of Gambling Issues, 10*. 10.4309/jgi.2004.10.1.

Blanchard, H. (2019). Blackjack: the math behind the cards. *Mathematics Senior Capstone Papers.* 4.https://digitalcommons.latech.edu/mathematics-senior-capstone-papers/4

Depaulis, T. (2010). Dawson's Game: Blackjack and the Klondike. In P. Endebrock (ed.). *The Playing-Card.* Journal of the International Playing-Card Society. *38*(4). The International Playing-Card Society.

Kendall, G., & Smith, C. (2003). The evolution of blackjack strategies. *The 2003 Congress on Evolutionary Computation, CEC ’03, 4,* 2474-2481. doi: 10.1109/CEC.2003.1299399.

Parlett, D. (1990). *A History of Card Games*. OUP, Oxford, 78.