Contents

[Analysis 2](#_Toc60827913)

[Introduction 2](#_Toc60827914)

[Computational Methods 2](#_Toc60827915)

[Decomposition 2](#_Toc60827916)

[Plans for Development: 2](#_Toc60827917)

[Abstraction 3](#_Toc60827918)

[Requirements 4](#_Toc60827919)

[Software 4](#_Toc60827920)

[Hardware 4](#_Toc60827921)

[Research 5](#_Toc60827922)

[Product: E-chalk, video poker (Echalk, n.d.) 5](#_Toc60827923)

[Game rules research: 6](#_Toc60827924)

[Essential Features 6](#_Toc60827925)

[Limitations 6](#_Toc60827926)

[Success Criteria 7](#_Toc60827927)

[Design 8](#_Toc60827928)

[User Interface 8](#_Toc60827929)

[User Input 8](#_Toc60827930)

[Decomposition 8](#_Toc60827931)

[Version 1 – Creation of ruleset and object interaction 9](#_Toc60827932)

[User Interface 9](#_Toc60827933)

[Mechanics 10](#_Toc60827934)

[Objects 11](#_Toc60827935)

[Classes, Functions and Variables 12](#_Toc60827936)

[Classes 12](#_Toc60827937)

[Methods 12](#_Toc60827938)

[Variables 15](#_Toc60827939)

[Validation 16](#_Toc60827940)

[Test Plan 17](#_Toc60827941)

[Post Development 18](#_Toc60827942)

[Bibliography 19](#_Toc60827943)

# Analysis

## Introduction

Poker is an immensely popular card game that has been around for years. From its early beginnings in the 1800s – in underground gambling dens in New Orleans- to its explosive growth in the early 1900s. Due to its age, this game has become intimidatingly complex and has many different rulesets – some of which have become synonymous to the world of gambling (namely four card poker and Caribbean Stud). Due to this complication, for this project I have decided to recreate the variant of poker known as “5 card draw” or “Draw Poker”. This game, despite inheriting the complicated hand values of its equals, is more simplistic in its ruleset thus making it the perfect candidate for a computing project.

## Computational Methods

### Decomposition

To achieve a fully functioning game of poker, I would have to use computational methods. The main one of which is decomposition. This method is useful because it takes one complex problem (for example a game of poker) and it splits it into smaller more specialised sectors – card objects, hand objects, the main ruleset and user interface. The use of this method makes the problem more achievable as it allows me (the main developer) to focus on each section of the problem one by one which will lead to a more polished product – compared to a product which has been put together in a rudimentary fashion to narrowly fit the brief. This overall would allow me to make a game of poker in an organised fashion which would deliver each of the required elements.

### Plans for Development:

1. Create basic objects that will work with each other to play the game
2. Link the objects together by cross referencing objects and creating super classes – for example, the “Round” object.
3. Create a simple betting system – which gives the player a set balance when the game is started- that allows the player to bet fake money.
4. Create front User Interface – which will handle all the user interaction like the “hold or draw” and “bet” functions.
5. Create an AI that will play against the user thus making the game more interesting.]
6. Create a graphical user interface that will allow the user to have something visual (more visual than IDLE shell) to interact with, implementing the basic features of the game like “hold” or “draw”.

### Abstraction

As well as breaking the problem down into smaller sections, I will also need to remove certain features – as trying to program all the rules for many different variants of poker would be unattainable. An example of a feature that I have decided to remove from my version of “Draw Poker” is the ability to gamble real money. This has been done not only to reduce the risk of underage users losing money (and potential lawsuits from such customers) but to reduce the development time. This is useful because it means the developer would not have to use certain APIs in the program to allow it to work, which would result in a less complicated program. As well as that if the program did handle sensitive data – like bank information- the quality of the program would have to be considerably better to accommodate for the handling, storing and processing of user data in a way that complies with the data protection act (Government, 2018). This would over complicate the program possibly resulting in an incomplete project as so much thought had to go into simple processing of data thus compromising the quality of the rest of the program.

Stakeholders

This product is mainly targeting the 18 – 30-year-old demographic. This is because this age group is the most likely to be familiar with the rules and concepts of poker, therefore it will appeal to them more than a young child. Although this program is mainly targeted towards the 19-30-year-old demographic, it may also appeal to people who are outside of this demographic. This has been accounted for as features that are only available for older people – the ability to bet/gamble real money- have been removed to make it safe for people who do not fit the target demographic.

## Requirements

### Software

Although other operating systems are viable, it is recommended that this program is to be run on a windows 10 operating system. This is because it has been conceptualised, designed and developed on this operating system which means it has been optimised for this operating system specifically. As well as this, some libraries might be harder to obtain on other operating systems compared to windows. Asides from operating systems, the user will also need a version of python (preferably the latest version of it) to be installed on their machine. This will allow the program to run on the user’s machine.

### Hardware

To run this program, the user will need a computer which can handle its operating system with at least 1 Gigabyte of RAM left after handling the background operations of that Operating System – which will be required to run python and its interpreter. As well as RAM, the computer will also need to have at least a processor with a 1 Giga Hertz clock speed and two cores (preferably four) to handle both the operating system and this program at the same time. Secondary storage will also be required, although it will not be as important. This is because the program is light weight, which will result in less space being required for the program. Due to this, it is recommended that the user has enough storage space for their operating system and at least 10 giga bytes extra.

## Research

### Product: E-chalk, video poker (Echalk, n.d.)



This product displays the game in its simplest form. Some key features from this product are the cards being displayed in the centre of the screen (as the main attraction); the ability to “Hold/Draw” cards for each of the cards and the ability to bet virtual money.

#### Pros

1. This version is straight to the point and keeps true to the rules of 5 card poker.
2. It has sprites for each of the cards and well thought out plan for the graphical user interface.
3. It is clear what you can and cannot do.

#### Cons

1. The rules and valuing of the user’s hand is poorly telegraphed which leads to confusion.
2. The betting system is very basic which results in very two-dimensional games (you either double your money or you do not).
3. There seems to be an abstraction of the rules, namely the tier system of hands which is lost due to the game being single player).
4. There is no-one to play against, which leads to a boring play session with little to no consequences for losing.

### Game rules research:

From research the following rules are required for the game of “5 Card Poker”: Hands rank from 1 to 10 (10 being the worst , 1 being the best) ; the player has the option to either hold or draw for each of the five cards which are drawn after a bet is placed ; the player can have a maximum bet of $20 per round (from their staring $100) and the player cannot see the AI’s hand until they have their final hand (after the drawing or holding stage).

## Essential Features

After the research I have concluded that the following features are essential: the ability to bet (virtual money) at the start of a round before the cards are played; the ability to hold or draw a new card – out of the five that are given when the round is started. After these basic mechanics, the hand value must be determined (using the rules of five card poker) and the player is either rewarded for their hand or they leave with nothing.

## Limitations

|  |  |  |
| --- | --- | --- |
| 1. | AI opponent | Due to single player nature of this program, you will only be able to play against an Artificial opponent as if two players play on the same client, the integrity of the experience will be forfeit. |
| 2. | Rules | Because of the time frame, some rules will have to be abstracted to ensure that the project is finished in time and to a high enough quality to be worth the development time. This also limits this version of poker to one ruleset (5 card poker). |
| 3. | Difficulty | An unintended side effect of an AI opponent is the fixation of difficulty. This means that the game will only be able to be played at one difficulty as the AI will be set to a certain ruleset whereas normal human opponents will function on their own ruleset, resulting in a variation in difficulty for the user. This variation could be simulated using a tick system (higher tick, better ruleset) but this will lead to an unbalanced experience which could hurt the user’s experience. |
| 4. | GUI | Due to the limitations of python and the tkinter module, the GUI that will be produced for this program will not only be affected by the development time but also the limitations of tkinter. This will result in a less polished result. |
| 5. | Controls | The user input of this program will be limited to the user’s peripherals. This means if the user is using a laptop, they will have to use the trackpad which is built into the computer. This will limit the flow of the game and could affect the user’s experience with the program. |

# Success Criteria

|  |  |  |
| --- | --- | --- |
| 1. | A functioning Card, Hand and Deck object | Simple class setup of the objects including the attributes, methods and cross referencing required for a card game to function. |
| 2. | A functioning super class Round object | The creation of the Round super class which will contain the main setup and user inputs for the game. This will need to reference the other objects in order to work as intended – for example, a local deck must be created for each round which is instantiated. |
| 3. | Simple rules – four of a kind, two of a kind etc | Using the attributes of the cards, the round class can work out the quantity of the cards in the player’s hand. From this, a value can be assigned, and the hand can be ranked – only using the basic rules for now |
| 4. | Combination rules – full house, double two of a kind | After the success of the basic rule implementation, more advanced rules can be implemented using the same techniques – e.g., if there are two sets of two, it is a double two of a kind. |
| 5. | Advanced rules – flush, straight flush, royal flush and straight. | New rules are implemented to accommodate for the advanced features. Some of these include finding the suit of each card I the user’s hand and comparing them to each other. |
| 6. | Implementation of a second player | Now that a single player can run through the game flawlessly, a second player can be implemented. This will require some extra logic to determine the winner of the round, but it should be as simple as creating a new instance of the Hand object and using a few for loops. |
| 7. | Implementation of an Artificial Intelligence | Using the successful two player approach, instead of a second human player, an AI can be programmed (possibly using inheritance) which will have a set of instructions to follow in order to be a worthy opponent for the user. |
| 8. | Implementation of a Graphical User Interface | Now that the game is working in the text editor, it is time to make something a bit more colourful. Using tkinter, a graphical user interface (GUI) will be constructed to be more user friendly and to hide the console text generated – thus making the program look cleaner and polished. |

# Design

## User Interface

Due to the time-consuming nature of GUIs, for the first versions, this program will be using the standard python console as its user input and output. I am choosing this for a few reasons: to save time and allow the core logical components of the program to be developed seamlessly. As shown below, the user will be given a few simple options at the start of each round. These options are designed to be concise and easy to interpret to allow the most natural UI experience as possible. Despite this, due to the nature of the python console window, the product will always look unpolished despite the effort that has gone into the smoothness of player inputs – that is why a full GUI will be developed at a later stage.

## User Input

To use this program, the user will need a keyboard and a mouse. These peripherals will be needed to input their desired choice and to operate the program.

At the start of each round the user (player 1) will be given an option to bet a specified amount of money. The user will then input that amount into the python window and that value will be saved. After this, the same procedure will be repeated for the other player and once they have inputted the amount of money, they are willing to bet the game will start.

## Decomposition

To ease the development process, I will be splitting my development into three main stages – which will each have a major version. These versions will include a version that contains the rules of poker, a version that has a built in AI opponent that can make decisions on its own accord (after being given a ruleset) and a version that encompasses the above and adds a graphical user interface on top to enable a more user-friendly experience.

### Version 1 – Creation of ruleset and object interaction

As seen below is my development and decomposition map for the first major stage of development – the development of the fundamental mechanics and the basic user interactions.

### User Interface

To construct a fully functioning user interface for this version of the program the problem must be broken down (decomposed) into three main sections – The user’s options (the option to draw a card or hold it), the display of the user’s inputs (acknowledgment of the user’s interaction with the program) and the creation of the player(s) hand objects. For each of the sections, a plan has been devised to aid the methodical and systematic development of each component in isolation – which at a later stage will be joined together to achieve a fully functioning user interface.

Graphical user interface, text, application

Description automatically generated

Above is the user interface that displays the user’s name, bet and (once they have placed a bet) their hand. After the user’s input, the program will then determine what rank the player has achieved (from the “determineRank” method) and it will then be displayed.

### Mechanics

Now that the basic user interface has been created, the main mechanics of the game of “five card poker” must now be tackled. Like the development of the user interface, this aspect will also be decomposed into its main few elements. These elements are the hand ranking system and the player ranking system.

The hand ranking system will be comprised of two sub elements, the hand ranking algorithm and the ruleset which will be required for algorithm to function. Below is an excerpt of the algorithm which is used to determine the player’s hand rank (from 1 to 10, 10 being the lowest and 1 the best).

A picture containing text

Description automatically generated

In this example, the program has been given a hand that does not satisfy the rules for a flush or a straight. It will then use the validation deck (which has been created during the instantiation of the “PlayerHand” object) to look for a multiple of a card amount. If the “tValue” (the first number of a tuple value that contains the “value” and the amount of a card) is equal to either a single amount of four, a single amount of three, a single amount of two, a double amount of two or a single amount of two and a single amount of three the program will then assign an integer value to the “rank” variable. This variable is later returned at the end of the process and is used to rank the competing player’s hands.

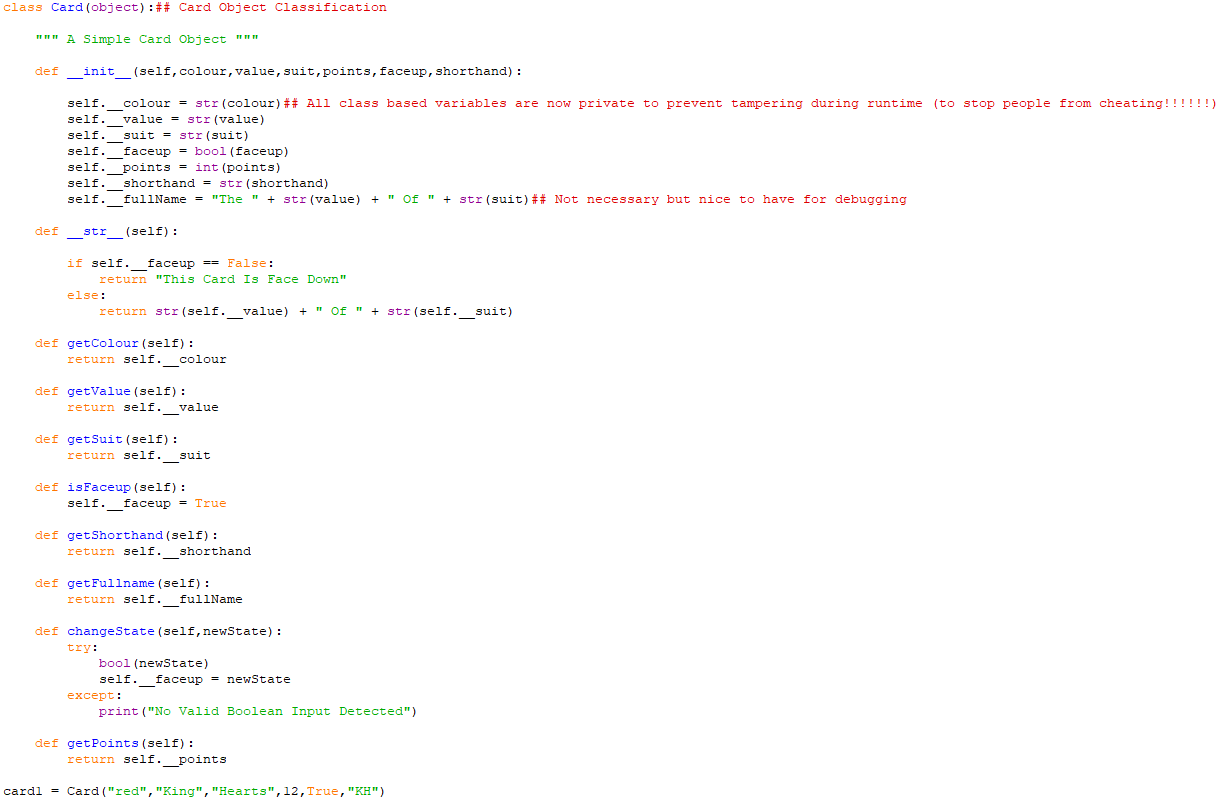
Now that a rank has been devised, the “determineWinner” function can compare the two values for “rank” and determine who won the round. Below is an excerpt of this function.

Text

Description automatically generated

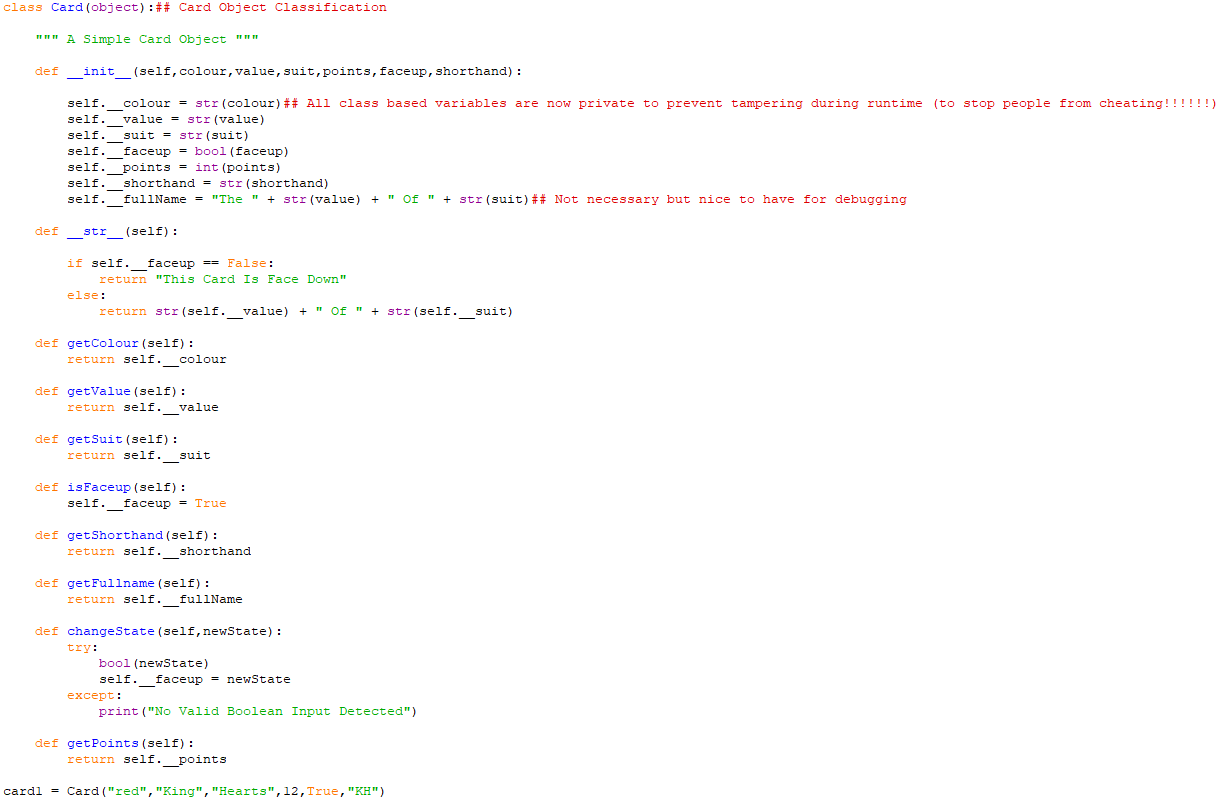
### Objects

Despite the prowess of the user interface and the functionality of the mechanics, the program will not function without the implementation and management of objects. With these objects, the separate sections of the program can interact with each other and create a program that works as intended. In the following sections, the objects will be outlined and explained in a suitable fashion which will indicate each object’s uses and how it interacts with the rest of the program.



Above is the code that is responsible for the creation of card objects . The variable “card1” has been created to test the functionality of the card object and whether it functions as intended. The object’s main methods have then been tested to confirm that they are working as intended.

Text, letter

Description automatically generated

As seen on the right, validation has been implemented in the “changeState” method. This is to prevent the accidental input of a non-Boolean response, which would result in the collapse of the main program.

Classes, Functions and Variables

To enable a program like this to function, a plethora of objects, functions, variables and external files are going to be required. Below are diagrams for each of the classes, functions and variables that are going to be required to run this program. Each of the diagrams will contain a note which describes how each of the objects, variables or functions relate to the program as a whole and will give some insight into the systematic decomposition which will take place.

### Classes

A picture containing timeline

Description automatically generated

### Methods

Below is the method table for the object “Card”.

|  |  |  |
| --- | --- | --- |
| Method number: | Method name | Brief description |
| 1. | getColour () | Returns the value for the variable “colour” |
| 2. | getValue () | Returns the value for the variable “value” |
| 3. | getSuit () | Returns the value for the variable “suit” |
| 4. | isFaceup () | Sets the variable “faceup” to True |
| 5. | getShorthand () | Returns the value for the variable “shorthand” |
| 6. | getFullname () | Returns the value for the variable “fullName” |
| 7. | changeState () | Changes the “faceup” variable to an input parameter |
| 8 | getPoints () | Returns the value for the “points” variable |

Below is the method table for the object “Deck”

|  |  |  |
| --- | --- | --- |
| Method number: | Method name | Brief description |
| 1. | shuffle () | Randomises the card order |
| 2. | getDeckBackup () | Returns the value of the variable “deckbackup” |
| 3. | getContents () | Returns the value of the variable “contents” |
| 4. | displayCardnames () | Displays the card names of each card in the deck |
| 5. | drawCard () | Returns a “Card” object from the deck and removes it from the deck. |
| 6. | replaceCard () | Adds a card to the deck |
| 7. | resetPlay () | Resets all the deck components to default (e.g., deck contents and card order) |

Below is the method table for the “Round” object (super class)

|  |  |  |
| --- | --- | --- |
| Method number: | Method name | Brief description |
| 1. | playRound () | This method is responsible for running the entire round. Inside of this method the program: instantiates an instance of a deck object – which is assigned to the variable “deck”. Once the “Deck” object has been instantiated, the players then have the option to place bets against each other which will then be saved in the “PlayerHand” variable “bet”. After this has occurred, the game of “five card poker” will commence. |

Below is the method table for the object “PlayerHand”

|  |  |  |
| --- | --- | --- |
| Method number: | Method name | Brief description |
| 1. | getLength () | Returns the length of the variable “contents” |
| 2. | resetContents () | Sets the “contents” variable to empty |
| 3. | getContentsShort () | Returns all the values of the variable “shorthand” for all of the “contents” |
| 4. | getContentsFull () | Returns the “contents” variable |
| 5. | playerDraw () | Adds a new card to the “contents” variable |
| 6. | playCard () | Removes a card from the “contents” variable. |
| 7. | placeBet () | Takes a user input and then sets the variable “bet” equal to it. |
| 8. | getHandType () | Returns the value for the “handType” variable. |
| 9. | winBet () | An input is taken and is added to the “balance” variable. |
| 10. | loseBet () | The “bet” variable is set to 0 and the value of it is returned. |
| 11. | replaceCard () | A card is removed from the “contents” variable and a new card is added in its place. |
| 12. | sortHand () | Sorts the hand in order of the “sortCode” variable – which is tied to the “Card” object. |
| 13. | displayContents () | The position in the “contents” list, “shorthand” and “fullname” are displayed. |
| 14. | getRank () | Returns the value for “rank” |
| 15. | getHighestCard () | Returns the value for the “highestCard” variable. |
| 16. | determineRank () | Using the ruleset, a ranking for the hand is derived and the variable “rank” is set to a value between 1 and 10. |

### Variables

Here are the variables that are required to enable the “Card” object to function.

|  |  |  |
| --- | --- | --- |
| Variable Name | Data Type | Description |
| Colour | String | The colour of the card |
| Value | String | The title of the card |
| Suit | String | The suit of the card |
| Faceup | Boolean | Whether the card is visible or not |
| Points | Integer | The amount of points the card is worth |
| Shorthand | String | The first Letter of the “value” and “suit” |
| Fullname | String | A concatenation of the “value” and “suit” of the card |

Here are the variables that are required to enable the “PlayerHand” object to function

|  |  |  |
| --- | --- | --- |
| Variable Name | Data Type | Description |
| Contents | List <Card> | A list of “Card” objects |
| Name | String | The name of the player |
| Validation | List<Card.Value> | A separate validation step setup to enable the ruleset to work |
| Rank | Integer (when assigned) | The strength of the hand |
| HandType | String | The formal name of the hand’s strength |
| NewCard | Card | The latest card to be added to “contents” |
| Bet | Integer | The amount being played for |
| Balance | Integer | The amount the player has to play with |
| HighestCard | Card | The highest scoring card in “contents” |

Here are the variables that are required to enable the “Deck” object to function

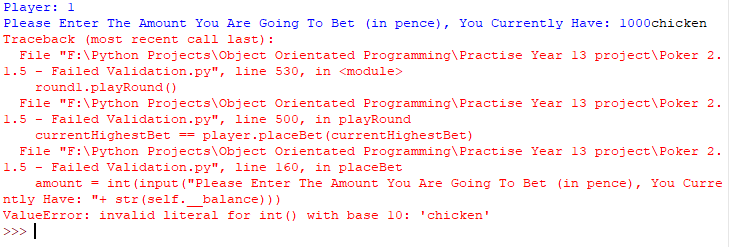
|  |  |  |
| --- | --- | --- |
| Variable Name | Data Type | Description |
| Contents | List<Card> | The main contents of the “Deck” object |
| deckBackup | List<Card> | The backup contents of the “Deck” object – this is also referenced when the “validation” is created in the “PlayerHand” object. |

Here are the variables that are required to enable the “Round” object to function

|  |  |  |
| --- | --- | --- |
| Variable Name | Data Type | Description |
| PlayerList | List<PlayerHand> | A list which contains the players |
| Deck | Deck | An instance of a “Deck” object which is responsible for the distribution of the cards in play |

## Validation

Due to the complex nature of Object Orientated Programming, validation is required to prevent the program from crashing during runtime. Some precautions that are used to prevent this from occurring include: preventing the user from inputting the wrong type of data (for example, a string instead of an integer) and preventing the user from tampering with the program mid runtime.



Above is an excerpt from an earlier version of the program which did not have validation of the “placeBet” method of the “PlayerHand” object. This meant inputting a string value instead of an integer value would result in the collapse of the program. To combat this, the validation below was implemented (namely the “try” and “except” statements) to help guide the user and to prevent a catastrophic melt down of the main program.

Graphical user interface, text, application

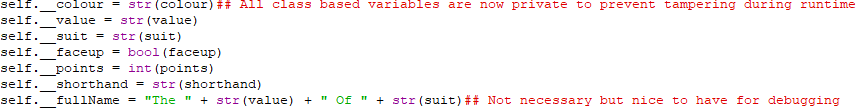
Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Now due to the validation that has been put in place, the program will now return the phrase “Invalid Input” whenever a casting error occurs thus preventing the program from breaking during runtime.

Due to the nature of the python Shell user interface, the user can modify the fundamental code during the runtime. While this is useful for development and debugging, this is not useful when the program is available to the public. To counter this issue, most of the values inside of the program have been set to private. This allows me to control how and when the values are distributed thus preventing tampering during runtime (for example, the user cannot change their entire hand to benefit them). This not only results in a cleaner program, but also results in an almost unbreakable program.



In this example, the fundamental attributes of the “Card” class have been made private.

## Test Plan

Now that most of the basic development is completed, a usable product has now been constructed. This product now must undergo some simple tests which will determine whether it is ready to be distributed to the client(s). Below is a table which contains the steps that will be carried out to test the program and how each one is going to be executed.

|  |  |  |
| --- | --- | --- |
| Test Number | Test | Description |
| 1. | Does the program function basically? | Run the program and check for simple errors |
| 2. | Does the program’s validation work? | Run the program and input the wrong data type for the input boxes -for example, a string for an integer value. |
| 3. | Does the betting system work as intended? | Try and bet against a second player, does the logic function as intended? |
| 4. | Does the ruleset work as intended? | Are all the rules functional? Using specialised files, test whether the program can detect all the hand types (not just the most common ones) |
| 5. | Can you modify the card attributes mid program? | Try and change the card attributes to obtain a better hand score, thus cheating the game. |
| 6. | Are there any aesthetic issues? | Are there any problems with the user interface? To much text to enter or read? |

## Post Development

After the first wave of development has ceased, work on the second version must commence. This includes the implementation of an Artificial opponent to play against the user and a graphical user interface to aid the use of the program. As well as adding to the program, a few patches must be implemented when bugs have been detected. This will not only potentially optimise the program to run on a wider range of machines, but it will also increase the lifetime of the product.

# Bibliography

*Echalk*. (n.d.). Retrieved from Echalk.

Government, U. K. (2018). *Data Protection Act 2018.* legislation.gov.uk.