

MINISTERUL EDUCAȚIEI NAȚIONALE



---

**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

---

NORTH UNIVERSITY CENTER IN BAIA MARE

FACULTY OF SCIENCES

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

COMPUTER SCIENCE SPECIALIZATION

*Poker Reader App*  
**BACHELOR'S THESIS**

**GRADUATE COORDINATOR**

**Lecturer HAJDU-MĂCELARU MARA**

**GRADUATE**

**UJLAKI TAMAS**

**2022**

# Table of contents

Introduction .....	1
Contents of the paper .....	2
Chapter 1 – Project Theme.....	3
1.1 The main objectives .....	3
1.2 Secondary objectives .....	4
1.3 Categorization of the application on the market.....	5
Chapter 2 - Bibliographic study .....	6
Chapter 3 - Technologies used and development environment .....	8
3.1 Technologies used.....	8
3.1.1 Programming language: Python .....	8
3.1.2 Important libraries used.....	8
3.2 Development environment.....	9
3.2.1 PyCharm.....	9
3.2.1 Tabnine.....	9
Chapter 4 - Design and implementation .....	10
4. 1 Development principles .....	10
4. 1.1 Object-oriented programming .....	10
4. 1. 2 Clean code .....	11
4.2 Diagram of the use case .....	11
4.3 Application classes and diagram.....	13
4.4 Activity diagram .....	13
4.5 Functional logic.....	15
4.5.1 Tesseract.....	15
4.5.2 User interface .....	18
4.5.3 Find objects .....	20
4.5.4 Continuous verification .....	24
Chapter 5 – Application testing and debugging.....	26
5.1 Live testing.....	26
5.2 Performance testing.....	27

5.2.1 CPU testing .....	27
5.2.2 RAM test .....	28
5.2.3 Time Testing .....	28
Chapter 6 — Installation and user manual .....	29
6.1 Installation and running .....	29
6.1.1 Requirements.....	29
6.1.2 Satisfaction of requirements .....	29
6.1.3 Running the application .....	31
6.2 User manual .....	32
6.2.1 Main instructions for use.....	32
6.2.2 UI components .....	35
Chapter 7 – Conclusions .....	39
7.1 Achievement of objectives.....	39
7.2 Further developments.....	39
Bibliography.....	41

## **Introduction**

The project called Poker Reader App is a desktop application that determines the status of a poker game opened in another application on the screen. It can determine the player's cards, the cards on the table, the names of the other players, their money amounts, the player holding the dealer button, the player who is "big blind" and "small blind", the total money present in that round, and based on this information, Poker Reader App determines the action of each player in the game. This app is a tool for reading poker game activity but can also be used as a working prototype for developers eager to add new features to it.

The Poker Reader App works based on image processing, specifically it takes screenshots, processes pixels, and after accumulating information, the app determines the game's actions using the visual data readings during the round.

The app is written entirely in Python, has an intuitive and friendly interface, and has a color palette representative of a card game. The Poker Reader App can currently only be used to determine data from the PokerStars desktop application.

## **Project background**

Currently, making an application where a bot is used to take over a player's activities and play competitively with the other participants in the poker round is a huge challenge in the field of computer science and mathematics, the only poker bot that can beat more than 2 professional players at one time in the Texas Hold'em version of the game is the Pluribus bot made by Facebook [1], and on the way to this challenge the first stop is reading the data and then processing it, thus achieving an optimal poker bot.

On many online poker sites and apps you can't use these types of bots to win money, but the process of writing such a bot brings new knowledge and valuable experience to the developer.

## **Contents of the paper**

This document is structured as follows:

- Up to this point the present project has been briefly introduced, described where the inspiration came from
- Chapter 1 is the theme of the project
- Chapter 2 is the bibliographic study of the project
- Chapter 3 presents the technologies used and the development environment
- Chapter 4 shows the diagrams of the project and its logic of operation
- Chapter 5 presents the method and results of the application testing
- Chapter 6 shows us how to use the application
- In Chapter 7, which is the last one, we look at what our app has achieved and how we can develop it further.

## Chapter 1 – Project Theme

In the first chapter we will look at the goals our application aims to achieve and how we can classify them.

In terms of the goals, we can look at them from two perspectives. There are the goals that describe the functionalities of the application and the goals that show how they are achieved. We call the first category Main objectives and the second category Secondary objectives.

### 1.1 The main objectives

First, for our project to be a poker reading tool, it must be able to read and process all the elements of a poker game, and then display the processed data, so that after this process we have all the necessary information to calculate the most favorable action that we can take. These are our first main objectives.

Secondly, since our app is a poker reading tool made for use by individual users, it must have certain actions that users can do.

This is also a main objective, and these actions will be presented in the table below where we describe every action that a user could do in any situation, in any state of an application of this type: in the first column we have the status of the application, in the second column we have the actions that are possible, and in the third column we have the additional details for each action.

State	Possible actions	Details
The app is open	Start Reading	
	Close application	
The application tries to read, but without success	Stop reading	The app enters its original state
	Waiting for the read to be successful	This action depends on the user's connection with the game of poker

	Opening the "Error Logs" tab	
	Close the "Error Logs" tab	
The app reads successfully	Stop reading	The app enters its original state
	Modification of data	Player data
		Total amount of money
		User's cards
		Community cards
	Opening the "Error Logs" tab	
	Close the "Error Logs" tab	
Modification of data	Data entry	
	Save the change	
	Undo the change	

*Table 1.1 – Actions of a user*

Finally, since we want our application to be able to be improved by other developers, we must have the source code in editable form.

## 1.2 Secondary objectives

Now that we know what exactly our application needs to do, we can talk about how the main objectives should be achieved.

Because a game of poker is constantly changing, the application requires reading and data processing quickly so that it can display the results in real time.

Another important secondary objective is that the data read is not erroneous, as they will affect the final calculation.

When we talk about writing the results, it is necessary for the application to have an interface that looks good, modern, so that users can read the data easily. At the same time, the UI must be intuitive, so that anyone who is familiar with poker can use the app from the very first second.

Our application must also be efficient, not use the computer resources it does not necessarily need. This aspect is very important because of the different processing power in any user base, but also because our application tries to be also a working prototype on which additional functions can be built, and if this base is poorly optimized, then any final product derived from it will also be poorly optimized.

Editable source code must be clearly written and documented so that anyone who develops it later can easily understand it.

The app must work reliably, which means that we will not encounter any errors or crashes during the game. It must also work without an internet connection.

Its operation on several operating systems is a benefit.

It would be useful for further development if the application is written in a single programming language and could be edited in a single code editor.

### **1.3 Categorization of the application on the market**

This project can be classified in two different ways: it can be considered either as an elementary poker reading application for users, or as a functional prototype written in the Python language for developers eager to improve it and make useful tools with it.

	<b>App categorization</b>	
	As a prototype	As an app for users
Intended	Python Developers	People who play poker
What they want	Developing this application further	Using a reader software
With the aim of	To acquire the knowledge, and to obtain beneficial new applications	Reading the information of a game in several ways
Comparison with other solutions	Unlike other similar apps, the Poker Reader App gives developers the ability to edit the app as they want	In the current version, the application does not offer special functions compared to other solutions

*Table 1. 3 – Categorization of the application*



## **Chapter 2 - Bibliographic study**

In the present chapter we look at the main themes of the application: we describe poker and image processing and specify their role for the purpose of the project.

### **2.1 Poker**

Poker is a card game that first appeared in America in the fourteenth century, since then it has evolved until the end of the twentieth century, and during this time many versions have been developed for this game, each with different regulations and the number of cards [2], but the essence has remained the same, that is, the winner is who collects the best combination between his cards and/or the community cards on the table.

#### *2.1.1 Texas Hold'em*

After the game spread to Texas, Texas Hold'em also ended up in Las Vegas, where for a few years only in the Golden Nugget casino could this game be found, and until 1980 it remained relatively unpopular. In 1988, in the case of "*Tibbetts v. Van De Kamp*", it was declared that Texas Hold'em was a game of skill, and the card rooms in all the United States began to adopt it. [3]

In recent decades, the Texas Hold'em version has begun to dominate the gaming scenes, and its television has contributed enormously to the rise in popularity between 2003 and 2006. Since then it has become one of the most popular gambling form that millions of people play every day as a recreational activity, and it is the only game of chance is considered a sports activity. [4]

In our app, when working with the objects of the game, the operating logic is built on the rules of Texas Hold'em, as this is the only type of game that the Poker Reader App wants to read.

### **2.2 Image Processing**

Digital image processing is the process of putting a digital image through an algorithm on a computer. Digital image processing is much more flexible than analog image processing because with the help of a computer we can modify each pixel separately, and so we get an incomparably better

flexibility than the process of modifying analog images in which we have to edit the entire image at once. [5]

Digital processing analyzes images as multidimensional matrices, where usually each pixel has 3 values. Specifically a red value, a green value and a blue value, and that's where "RGB" (red, green, blue) comes from. Based on this, an image with the size of 1000x1000 pixels can be represented as a three-dimensional matrix, 1000x1000 small matrices where each small matrix has 3 values.

This changes at black and white images, where the matrix will be 1000x1000x1, since each pixel will have a single value between 0 and 256, with 0 representing completely black and 256 representing all white.

If the Poker Reader App can be divided into two parts, and one side is Texas Hold'em, then the other part is image processing. We use this to get the objects with which we are going to work.

## **Chapter 3 - Technologies used and development environment**

This chapter introduces the technologies used to build the Poker Reader App. We will describe the language used, the important libraries applied to the project and the development environment.

### **3.1 Technologies used**

#### *3.1.1 Programming language: Python*

Python is one of the most popular programming languages, it is high-level, interpreted, and general-purpose. It was created by Guido van Rossum in the late 80's. Its design philosophy emphasizes the legibility of the code by using significant indentation. Many of python's features allow for object-oriented programming.

The memory management system in Python is based on a garbage collector with cycle detection with dynamic printing. The binding of method and variable names occurs during the execution of the program (late binding). [6]

#### *3.1.2 Important libraries used*

##### **- PySimpleGUI:**

A Python module called PySimpleGUI allows programmers to design graphical interfaces. A "layout" that includes items that are used is to design the GUI window.

PySimpleGUI uses these 4 frameworks - Tkinter, Qt, WxPython and Remi - to display and interact with the window. [7]

##### **- OpenCV:**

The Open-Source Computer Vision Library, often known as the "OpenCV", is a free and open-source software library for computer vision and machine learning, developed by Intel in 1999. OpenCV was created to provide a standard framework for computer vision applications. [8] It contains over 2500 optimized algorithms, including a wide range of traditional and cutting-edge computer vision and machine learning techniques. These algorithms can be used to track camera movements, detect and recognize faces, identify objects, classify human actions in movies, and more. [9]

- PyTesseract:

Google's Tesseract-OCR Engine optical character recognition tool (OCR) has a Python pack called Python-tesseract, also known as PyTesseract.

OCR is a type of program in which an image is inserted, and it recognizes all the alphabetical and numerical characters on it. [10]

## **3.2 Development environment**

### *3.2.1 PyCharm*

The Python programming language is supported by PyCharm, a cross-platform integrated development environment (IDE) used in software development. JetBrains (formerly IntelliJ), a Czech firm, created it in 2010. In addition to many other capabilities, PyCharm offers code analysis, a graphical debugger, a built-in drive tester, integration with version control systems (VCS), and actively supports the use of plugins. [11]

### *3.2.1 Tabnine*

Tabnine is an artificial intelligence-based code completion software that works with 9 programming languages and 12 different IDEs.

It has a free plan and a paid one, and in this project the free level was used.

We used it as a plugin, which, as we can read above, is an easy thing to do in PyCharm.

## **Chapter 4 - Design and implementation**

The purpose of this chapter is to help us understand the logic of the operation of the Poker Reader App, the description of the written ui code and its explanation.

### **4. 1 Development principles**

#### *4. 1.1 Object-oriented programming*

Object-Oriented Programming (OOP) is a programming model that organizes software design around data/objects rather than functions and logic.

An object is a data field with its own set of properties and behavior. [12]

This type of programming is ideal for large, complicated and frequently updated or maintained projects. It comprises of manufacturing and design software, as well as mobile applications; for example, OOP can be used to simulate manufacturing systems.

The structure of an object-oriented program also makes it useful for collaborative development, in which developers are organized into groups. Code reuse, scalability, and efficiency are also advantages of OOP.

The first step in OOP is data modeling, which involves collecting all the objects that a programmer wants to manage and determining how they connect with each other.

Physical entities, such as a human being with characteristics such as name and address, to simple software such as widgets, are examples of objects.

Once an object is identified, this is assigned to a class of objects that describes the type of data it contains and any manipulable logical patterns. The term "method" refers to each unique logical sequence.

The following are some of the components or building elements of object-oriented programming:

- Individual objects, attributes and methods are created using classes, which are user-defined data types.

- Objects are instances of a class that have been generated using data that has been precisely defined. Objects can be an abstract concept or a real-world thing. The description is the only element that is defined for the first time when a class is created.

- The methods are described as internal functions of the class that explain the behavior of an object. Each method observed in class statements begins with a reference to an instance object. Instance methods are another name for the routines that make up an object. Programmers use techniques to make code reusable or maintain the functionality contained in one object at a time.

The state of an object is represented by attributes, which are defined in the class template. The attribute field of the objects will contain data. The characteristics of the class are the properties of the class.

### *4. 1. 2 Clean code*

A development method called "clean code" puts the reader first and results in software that is simple to build, read and maintain. When the program works as planned, the developer could often be tempted to consider the effort put in as finished. However, we do not create code just for use on the computer. [13]

Realizing that in most projects, the code will be read and modified in the future by other people means that most developers have to write them in this way.

Since the last main objective of the Poker Reader App is to have an easy-to-read and edit code, the principle of clean coding is of the utmost importance.

## **4.2 Diagram of the use case**

The Unified Modeling Language (UML) is a general-purpose modeling and development language in the field of software engineering, created by Rational Software in 1994-1995. Its purpose is to provide a coherent way of representing the architecture of a system. Structure diagrams, behavior charts, and interaction diagrams are the most commonly used. [14]

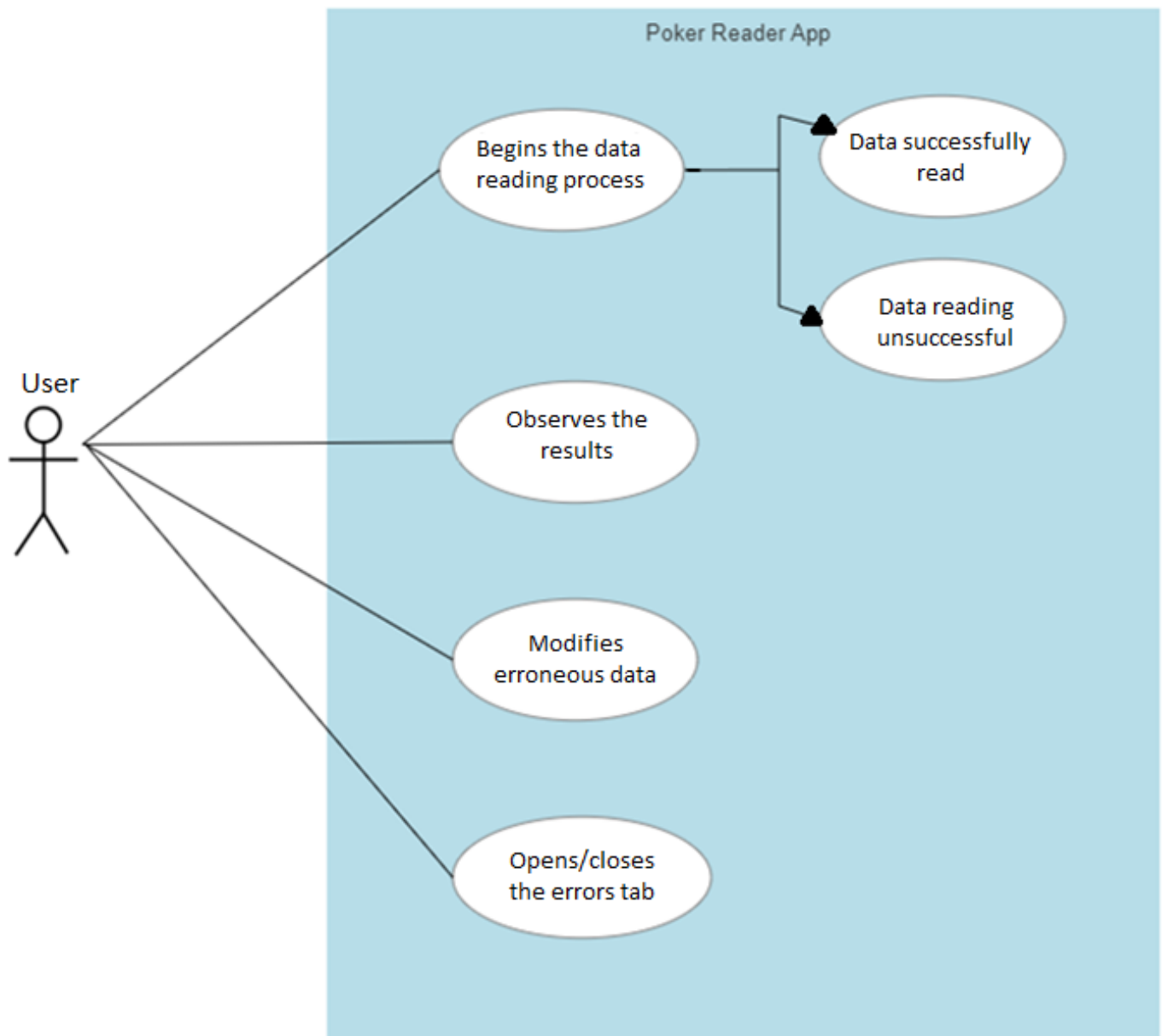
Use case diagrams are used in UML to represent the behavior of a system. They describe the high-level functions and scope of a system. The interactions between the system and its users are also described in these diagrams. In the use case diagrams, the functions of the UI system and how the participants interact with it are defined, but not how the system acts inside. [15]

## *Poker Reader App*

A user is a real-world entity that interacts with the system through a use case. Customers, merchants, operators and others can play the role of users. External systems may also be participants of the use case. [16]

First, we have developed the system as a personal application, and the participants are the user and the PokerStars app from which we obtain the desired data.

Because the PokerStars app is considered a separate entity that does the same actions regardless of the Poker Reader App system, it does not appear in the diagram below.



*Image 4.2 – Diagram of the use case*

### 4.3 Application classes and diagram

Classes are one of the main components of object-oriented programming. We use them when we have several objects of a certain type, if each object will have the same types of attributes and the same types of functions associated with them.

For example, in my application we will have several instances of a Player object, and each will have an attribute "id", "name", "money" and functions to modify them.

We can also put in the class definitions all the functions that are associated with them, and this makes the code easier to read and avoids duplicating the code if we have to call that function in several places.

The application contains two classes used for object types that own multiple instances, these can be found in the UML class diagram in the image below.

Class diagrams can be used to model the objects that make up the system, to describe the relationships between them, and to define the execution of these objects and services provided. [17]

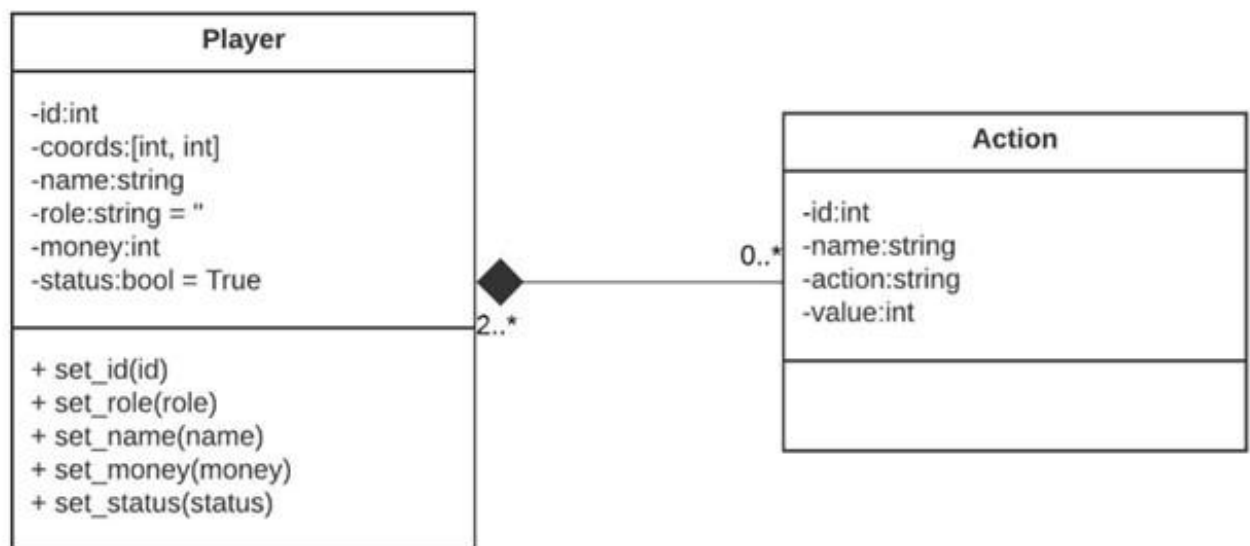


Image 4.3 – Class diagram

### 4.4 Activity diagram

By representing the sequence of actions in a process, an activity diagram provides a projection of the behavior of a system. [18] In the image below I presented the activity diagram of my application. Activities can be decomposed even further, but important fragments will be presented.



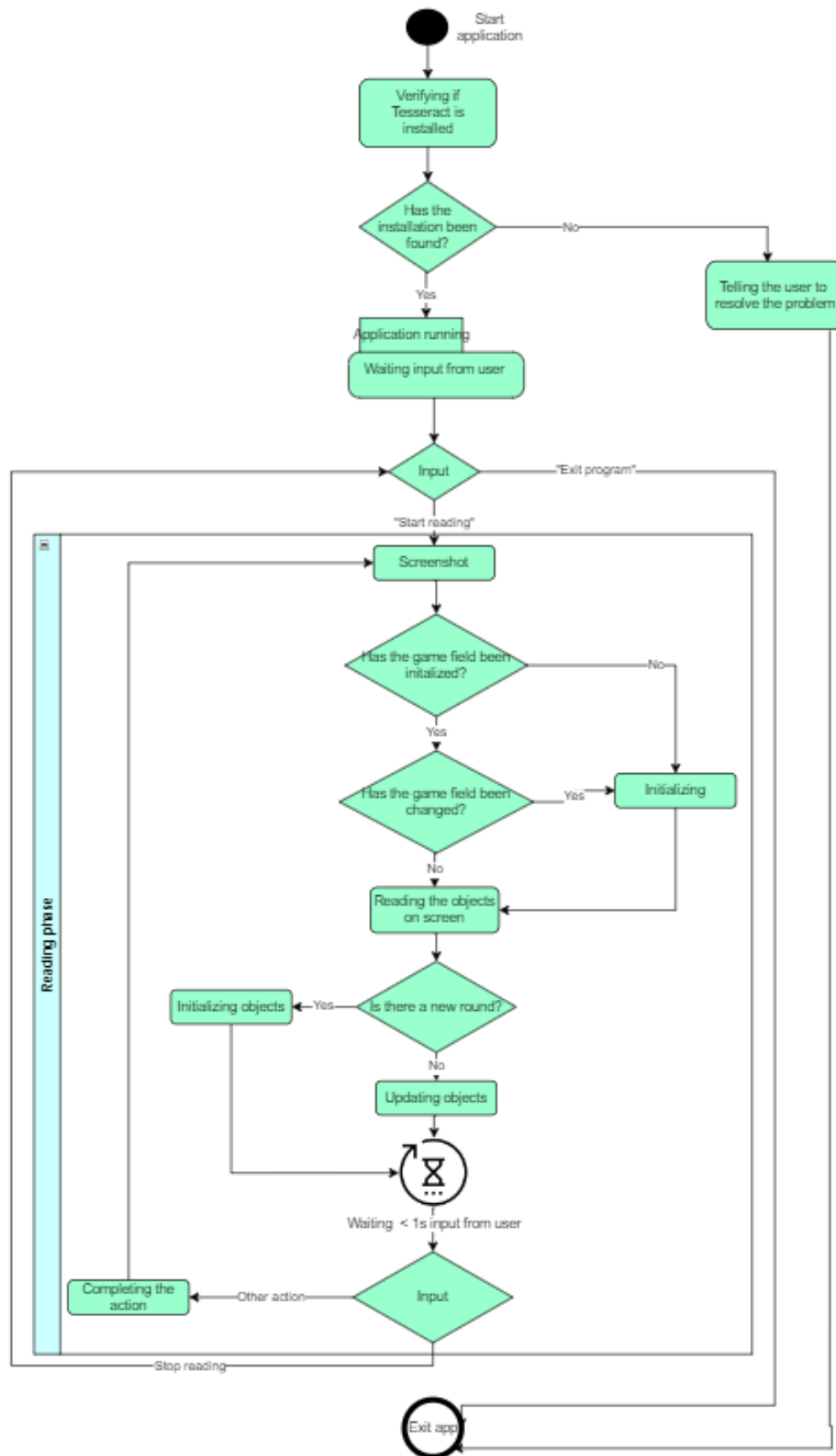


Image 4.4.1 – Activity diagram

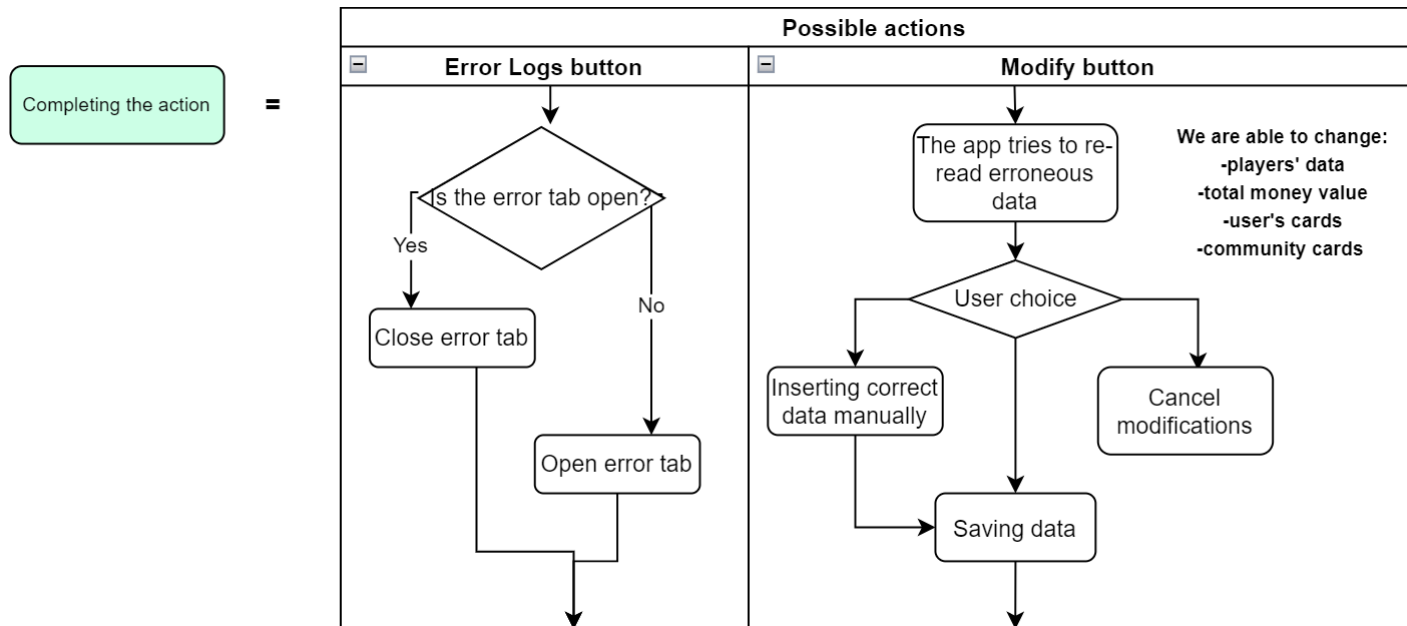


Image 4.4.2 – Completion of the action

## 4.5 Functional logic

In the previous subchapter we showed the logic on which the application is built, and here we will examine in more depth what exactly the code is for these actions mentioned.

### 4.5.1 Tesseract

For OCR (Optical Character Recognition) functions, the Poker Reader App uses an external program called Tesseract, which we must download separately (the detailed guide for this is in chapter 6).

It makes sense that the first thing our application does is check if it can locate the installation folder, and if it fails, then it warns the user about the problem and helps them with solving it.

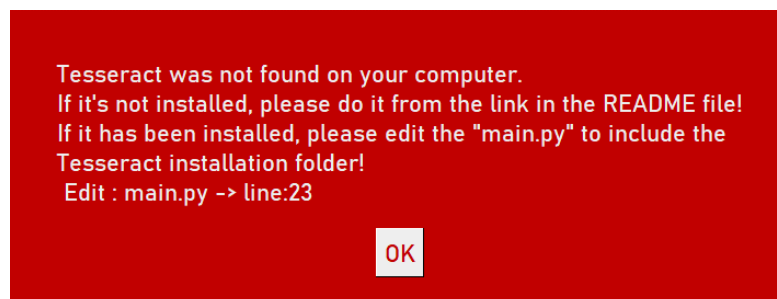


Image 4.5.1.1 – The Tesseract problem

## Poker Reader App

This red box is a boundless window created with the help of the PySimpleGUI library, which was presented in chapter 3.

Codes that execute Tesseract's searches:

```
start_time = time.time()
available_drives = ['%s:' % d for d in string.ascii_uppercase if os.path.exists('%s:' % d)]
count = 0
for drive in available_drives:
    for root, dirs, files in os.walk(drive):
        count += 1
        if 'Tesseract-OCR' in dirs:
            tess_window.close()
            del tess_window
            return os.path.join(root, 'Tesseract-OCR') + '/tesseract'
# at every 10.000 iterations it reads the elapsed time
if count == 10000:
    sg.Window.refresh(tess_window)
    count = 0
# if the elapsed time is more than 5 seconds then it tells you about the error
if (time.time() - start_time) > 5:
    tess_window['TEXT'].update('Tesseract was not found on your computer.')
```

Image 4.5.1.2 — Cooking tesseract first method

```
for drive in available_drives:
    first_dirs = os.listdir(f"{drive}/")
    if 'Tesseract-OCR' in first_dirs:
        return f"{drive}/Tesseract-OCR/tesseract"
    else:
        for fdir in first_dirs:
            try:
                second_dirs = os.listdir(f"{drive}/{fdir}")
                if 'Tesseract-OCR' in second_dirs:
                    return f"{drive}/{fdir}/Tesseract-OCR/tesseract"
                else:
                    for sdir in second_dirs:
                        try:
                            third_dirs = os.listdir(f"{drive}/{fdir}/{sdir}")
                            if 'Tesseract-OCR' in third_dirs:
                                return f"{drive}/{fdir}/{sdir}/Tesseract-OCR/tesseract"
                        except:
                            pass
            except:
                pass
```

Image 4.5.1.3 — Cooking Tesseract the second method

The application begins to search for all the drives on the computer and, on each drive, searches in folders to find the folder called "Tesseract-OCR".

As seen in Image 4.5.1.2, at first I tried to approach this search with the `os.walk()` function, which goes through each folder and file in the specified directory, and the code of this method was more elegant,

but it took much longer than the code in *Image 4.5.1.3*. For example, on a computer with the Intel i7-8750H processor, the first method took more than 2 minutes for a False result, but with the second method only a few seconds. The search time with the second method improves even in the case of successful searches, but the difference is less visible.

Even if the second method checks fewer folders in total, and the source code is harder to understand at first glance, the decrease in execution time compared to the first method outweighs the negative aspects.

But let's see why Tesseract is so vital for the app that it doesn't start without it. We use Tesseract to read the names of the players, their money values and to read the chat.

Since our app can only read from PokerStars, it is easy to calculate from the coordinates of a player's cards where their name and the amount of their money will be on the screen.

From this point on, we just have to cut those small parts of the image, give it to Tesseract, and it says what characters appear in the given image.

Since in PokerStars, players' names and money are written in white against a black background, we reverse the colors so that Tesseract gives us more accurate results. I noticed that the percentage of error decreased when this color reversal was done.

```
picture = picture[upper:lower, left:right] # cutting out the picture so we only have the name part

# manipulating our image a bit, so that Pytesseract will be able to read it better
gray = cv.cvtColor(picture, cv.COLOR_BGR2GRAY)
gray = cv.resize(gray, (gray.shape[1] * 5, gray.shape[0] * 5))
retval, thresh = cv.threshold(gray, 130, 255, cv.THRESH_BINARY)
thresh = cv.bitwise_not(thresh)

# cv.imshow('readname', thresh)

name = pytesseract.image_to_string(thresh) # pytesseract reads the text from the little image
```

*Image 4.5.1.4 – Read names with Tesseract*

Reading a name or the cash value of players with Tesseract adds quite a lot of time to our reading loop, usually 0.2 seconds, which is about how long our loop would last without it.

When a new round begins and we read the name and money of each player, the total reading time could reach 4-5 seconds.

Reading with Tesseract is not recommended in the long term, so in the future it is desired to develop an internal character reader.

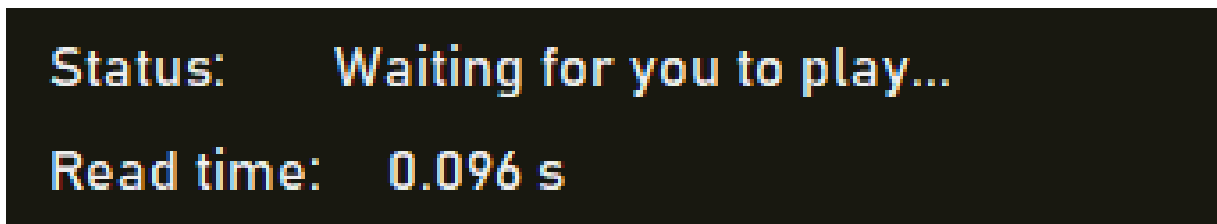
### 4.5.2 User interface

As was said in the previous chapter, for the interface part of the application I used PySimpleGUI. It is based on the logic that the necessary elements must be written down, then added to a list, and that list will become a row. Several lists are used to create the entire interface.

In *Picture 4.5.2.1* we have 4 different elements, and thanks to the presented structures, they will form 2 rows as seen in *Image 4. 5.2.2*.

```
[sg.Text('Status:   '), sg.Text('', key='STATUS')],  
[sg.Text('Read time: '), sg.Text('0.000 s', key='READ_TIME')],
```

*Image 4.5.2.1 – Example UI item code*



*Image 4.5.2.2 — Example of UI item code operation*

I've also arranged items in columns, and multiple columns in a list will form a row. For the UI in *Image 4.5.2.4* I vertically separated the surface of the application into two parts, so I made columns and put them together to form a row, as seen in *Image 4.5.2.3*.

We also have a separator, but only for stylistic reasons.

```
[gamecol1, sg.VSeparator(pad=(5, 15)), gamecol2]
```

*Image 4.5.2.3 – Example columns in a row*



vertical separator

*Image 4.5.2.4 – Example of columns in action*

Another thing worth noting about PySimpleGUI, is how the elements are updated. We call the syntax `<window['ELEMENT_KEY'].update(attribute=value)>` to change, for example, the value of a text element or any characteristic associated with that element.

The last interesting feature of PySimpleGUI is that once you have created the window, it is not possible to remove or add elements. This means that the only solution for an element that does not need to be visible in each state of the application is to hide it, later it can be made visible again.

The best example of this will be the "Error Logs" tab, where we start with it hidden *Image 4.5.2.6*, but when the user clicks a button, it appears *image 4.5.2.5*.

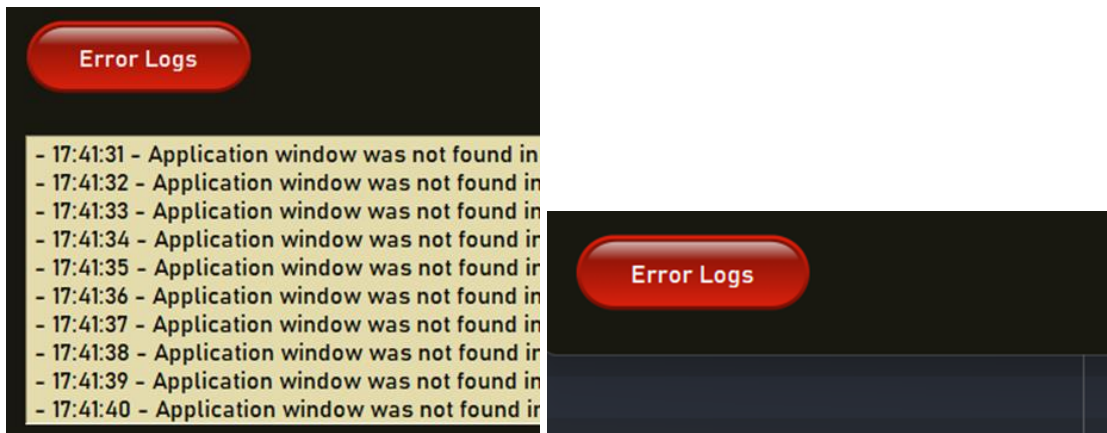


Image 4.5.2.5 – Tab of errors appears    Image 4.5.2.6 – Error tab hidden

Code (Image 4.5.2.7) The Error Log tab is written so that the user can close and open it with the same button called "Error Logs".

```
errors_open = False # the initial state of our error logs tab
if event == 'ERRORS':
    # opens and closes the error logs tab
    errors_open = not errors_open
    window['ERRORLOGS'].update(visible=errors_open)
```

Image 4.5.2.7 - Operating Code Error Logs

### 4.5.3 Find objects

When we first start the application, the dimensions of the elements in the game are not known, for example the length and width in pixels of the cards or the dealer button. We need an initialization function to get these dimensions, and we will also get initial reference points so that in subsequent reading instances we can determine if the PokerStars window has been moved or resized.

First of all, at each instance of reading, we get a screenshot of the left half of the screen. I chose this because I felt that the Poker Reader App window should be in the right half of the screen, and since the PokerStars app can't be covered anywhere, it must only use the left half. Luckily, Windows has a useful function for splitting the screen in two. Another benefit of the fact that the PokerStars app is only in the left half of the screen is that of working with fewer pixels and fewer elements on the screen.

The initialization procedure has the following logic:

- check that the PokerStars app bar is visible, then save its dimensions and coordinates (Image 4.5.3.1)

## Poker Reader App

- check if the chat tab can be found, then save its dimensions and coordinates (*Image 4.5.3.2*)
- cut the image over the application bar and under the chat, because between this will be our playing space (*Image 4.5.3.3*)
- finds the user's books and saves their dimensions (*Image 4.5.3.4*)
- finds the opposing cards and saves their dimensions
- finds the button and saves its size

```
if not app_bar_found:
    gui.error(window, "Application window was not found in the initial mapping") # error message
    return False
app_bar_pic = gray[app_bar_coords[1]: app_bar_coords[1] + app_bar_height, # the picture of our app bar
                  app_bar_coords[0]: app_bar_coords[0] + app_bar_width]
interface["app_bar_pic"] = app_bar_pic # we start writing data in our global data holder
interface["app_bar_coords"] = app_bar_coords
interface["app_bar_width"] = app_bar_width
interface["app_bar_height"] = app_bar_height
```

*Image 4.5.3.1 – get app\_bar*

```
if not chat_found:
    gui.error(window, "Chat was not found in the initial mapping") # error message
    return False
interface["chat_coords"] = chat_coords
interface["chat_width"] = chat_width
interface["chat_height"] = chat_height
```

*Image 4.5.3.2 – get chat*

```
# now we cut out the screenshot, so we only have the place between the upper app bar and the chat
# every action is going to happen here in this space
image = image[(app_bar_coords[1] + app_bar_height): (chat_coords[1]),
              (app_bar_coords[0]): (app_bar_coords[0] + app_bar_width)]
```

*Image 4.5.3.3 – image trimming*

```
if not found_a_card:
    gui.error(window, "Your cards were not found in the initial mapping")
    return False
interface["my_card_width"] = my_card_width
interface["my_card_height"] = my_card_height
```

*Image 4.5.3.4 – get my\_cards*

If any of these objects can not be found, then we will receive an error, and our application waits for a small period of time and tries this initialization process again.



As we have noticed, the data that we have saved from these objects is put in a global dictionary (*Image 4.5.3.5*). I decided to do it this way, since this type of object has only one instance, so a class would not be more beneficial, and it is global because the value of its variables does not constantly change, so it is mainly for reading purposes. Also, the application does not run multithreaded in the current version, so changing unauthorized data is not possible.

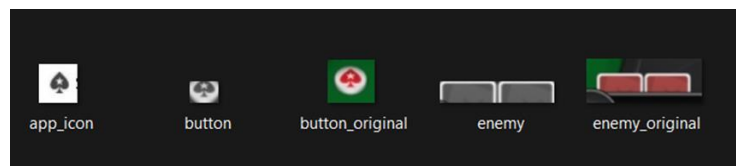
```
# this interface will be initialized once, and is needed for the readings of the game's elements
# once the game's window is resized, the values will be changed
interface = {
    'app_bar_pic': 0,
    'app_bar_coords': (0, 0),
    'app_bar_width': 0,
    'app_bar_height': 0,
    'chat_coords': (0, 0),
    'chat_width': 0,
    'chat_height': 0,
    'my_card_width': 0,
    'my_card_height': 0,
    'button_width': 0,
    'button_height': 0,
    'enemy_width': 0,
    'enemy_height': 0,
}
```

*Image 4.5.3.5 — Original data dictionarye*

Finding objects on your screen:

A more complex system could use machine learning algorithms, such as the one used to identify faces, but this is very expensive in terms of development time.

In the Poker Reader App, pre-saved images are used in the project files (*Image 4.5.3.6*), which we compare with the images currently being read to check how similar they are.

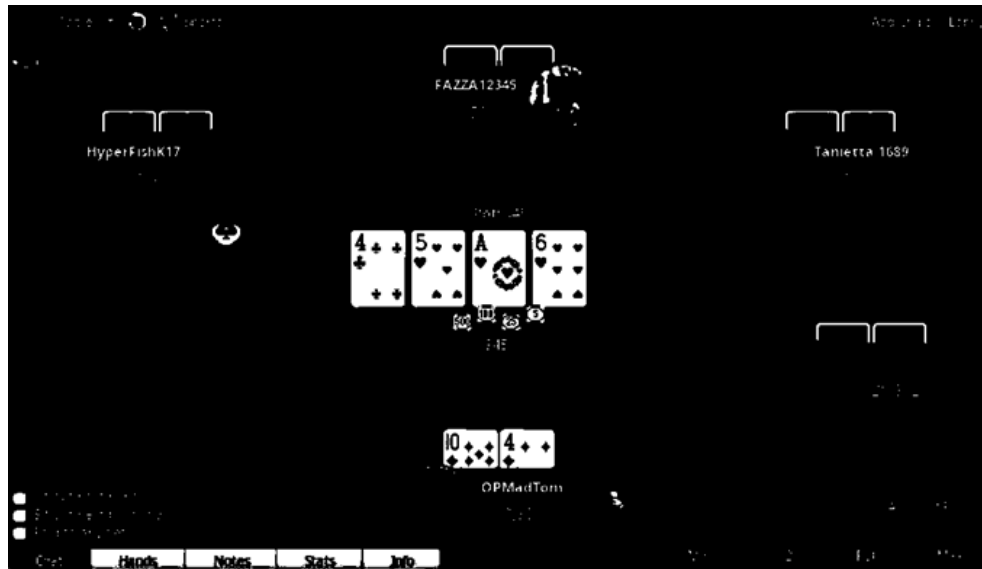


*Image 4.5.3.6 – Pre-saved images*

First, we check our screenshot for contours. This is done by turning it black and white, blurring it and applying a threshold. Thresholding means that on a black and white image, if the intensity of a particular pixel is above a certain value, then we make that pixel completely white, otherwise completely black.

## Poker Reader App

This means that the image used after the threshold will be a black background with white objects on it as in *Image 4.5.3.7*.



*Image 4.5.3.7 — Image after application of a threshold*

At this point we can check the dimensions of these white objects, and if, for example, the dimensions of an object are very similar to the calculated dimensions of the dealer button, then we check how similar the cropped part of the original image is to the predefined button image. (*Image 4.5.3.8*)

```
# gets the contours of our image, to which we are applying GaussianBlur = 1(None) and Threshold = 190
contours, gray = getcontours(picture, 1, 190)

for contour in contours:
    x, y, width, height = cv.boundingRect(contour)
    # if the contour's sizes are correct, we are going to check it against a predefined image to see if it is
    # similar enough
    if approx(height, button_height, width, button_width):
        picture = gray[y: y + height, x: x + width] # cutting out our image
        button_picture = cv.imread(globals.button_path) # predefined image
        diff = difference(button_picture, picture) # difference between them
```

*Image 4.5.3.8 – Finding the button*

Checking cards is more complex, since it is not the best idea to check 52 cards for each object. Instead, we select the parts where the color and rank are placed, and then check the outlines in that small part cropped again to have only the symbols and small numbers. During the development of the application, I realized that this last step greatly improves the accuracy. After this step, we compare the

## Poker Reader App

image of the suit with the 4 predefined suit images (*Image 4.5.3.9*), and if the difference is not too large, then we compare the rank image with the 12 predefined rank images. This way, we only perform 4-16 checks for a card object instead of 52. Finally, we select the comparisons with the smallest differences, and this will decide which suit and rank has the card (*Image 4.5.3.10*).

```
# calculating to which predefined image is ours most similar to
i = 0
for file in os.listdir(globals.suits_path):
    if file.endswith('.jpg'):
        suit = cv.imread(f'{globals.suits_path}/{file}')
        diff = difference(suit, card)
        suits[i] = diff
        i += 1
```

*Image 4.5.3.9 - Finding the colour*

```
card_rank = ranksnames[np.argmin(ranks)]
card_suit = suitsnames[np.argmin(suits)]

return f'{card_rank}_of_{card_suit}'
```

*Image 4.5.3.10 – Determination of the card*

### 4.5.4 Continuous verification

Each reading loop in the Poker Reader App starts with taking a screenshot and then reading the objects' data in each capture. However, we cannot search for items unless we are sure that since the initialization of the game the PokerStars app has not been moved, resized or disappeared. This can be checked by cropping the current screenshot in the same place where the app bar was found when initialized and seeing the difference between them (*Image 4.5.4.1*).

```
# gets half a screenshot, and returns if the poker application has been moved or resized
i = globals.interface
current_app_bar = half_screen[(i['app_bar_coords'][1]): (i['app_bar_coords'][1] + i['app_bar_height']),
                               (i['app_bar_coords'][0]): (i['app_bar_coords'][0] + i['app_bar_width'])]
# [upper : lower, left : right]

# checking the difference between our initialized Pokerstars application bar, and the same part of our screenshot
if difference(current_app_bar, i["app_bar_pic"]) == 0:
    # cuts out the game's part of the screenshot and the chat part of the screenshot, and returns them
    game_pic = half_screen[(i['app_bar_coords'][1] + i['app_bar_height']): (i['chat_coords'][1]),
                           (i['app_bar_coords'][0]): (i['app_bar_coords'][0] + i['app_bar_width'])]

    chat_pic = half_screen[(i['chat_coords'][1]): (i['chat_coords'][1] + i['chat_height']),
                           (i['chat_coords'][0]): (i['chat_coords'][0] + i['chat_width'])]
    return game_pic, chat_pic
```

*Image 4.5.4.1 – Initial verification of each image*

If these two do not match it means that the PokerStars app has been moved, resized or disappeared, so we will have to try again the initialization process described in the last subchapter.

If they fit, then we can proceed with reading the objects.

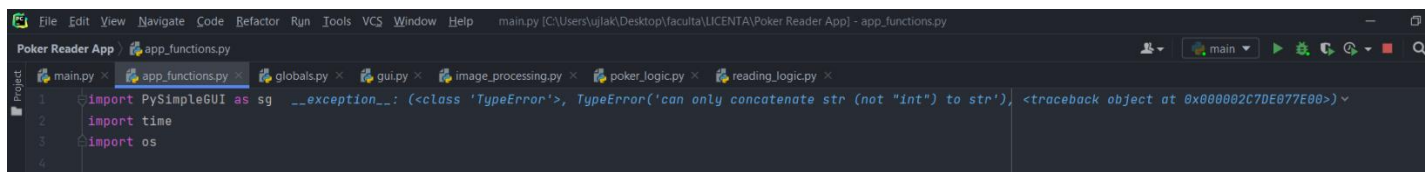
## Chapter 5 – Application testing and debugging

This chapter presents the testing of the application and how the encountered bugs were fixed.

### 5.1 Live testing

In Pycharm there is a "debugging" mode when we choose to run our application, and when it encounters an error, it stops at the line of code where the error was produced. When it stops in this mode, it gives us the name of the error and all the variables and values present at that moment so that we can examine them and make corrections in the code so that that problem is avoided in the future.

An example of an error is shown in *Image 5.1.1* and *Image 5.1.2*.



*Image 5.1.1 – Example error 1*



*Image 5.1.2 – Example error 2*

In this example from *Image 5.1.1* we can see that Pycharm's debugging mode tells us that we have a value type error with a string and an integer.

In *Picture 5.1.2* we can see that the variable `count` contains the string "count", so when the application reaches line 23, which is `count += 1`, it tries to add 1 to the value of `count`.

Since it is not possible to perform an addition between a string of characters and an integer, we get the error shown above.

To correct this problem, we can go to the declaration of the `count` variable to turn it into an integer (*Image 5.1.4*) instead of a string (*Image 5.1.3*).

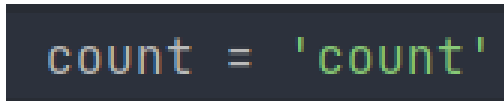


Image 5.1.3 – Cause of error



Image 5.1.4 - Correction of error

The key for the debugging method to be a good test choice is running the application for extended periods of time and putting it in different scenarios in which a user would put them. We can consider that we have an application that works correctly if after all these scenarios the application works as expected.

## 5.2 Performance testing

I have tested the resource requirements of the application on my personal computer. I tested how much of the RAM and CPU the application uses and the read time of each reading loop. For RAM and CPU values we use Task Manager, and for reading the time the application has a built-in timer that is constantly updated.

Tests are carried out on a laptop with an Intel i7-8750H processor and 8gb RAM on the Windows 11 operating system.

### 5.2.1 CPU testing

Application status	CPU used
Open app	0 - 0.1 %
Trying to find the PokerStars app	0.7 - 2.4 %
With PokerStars found	0.9 - 2.9 %
With the P.R.A. actively reading a game of poker	1.3 - 6.4 %

Table 5.2.1 - CPU testing

(P.R.A. = Poker Reader App)

We can note that the processor is not used significantly , even when the application actively performs image and data processing during a round of poker, and this aspect can be explained by two reasons:

- The application is designed to be easy and efficient
- The application uses only one of the processor cores

## *Poker Reader App*

Most likely, any processor that can run Windows 10, 11 or Mac OS operating systems can also run this application without problems.

### *5.2.2 RAM test*

<b>Application status</b>	<b>RAM used</b>
Open app	76.9 MB
Trying to find the PokerStars app	79.1 - 96.3 MB
With PokerStars found	85.0 - 91.1 MB
With the P.R.A. actively reading a game of poker	82.0 - 114.5 MB

*Table 5.2.2 - RAM testing*

We can also observe that the use of RAM does not vary much between the time of stagnation and the time when each process is run.

### *5.2.3 Time Testing*

<b>Application status</b>	<b>Reading time</b>
Opened app	-
Trying to find the PokerStars app	0.110 – 0.151 seconds
With PokerStars found	0.098 - 0.124 seconds
With the P.R.A. actively reading the game of poker	0.155 – 0.303 seconds
When the P.R.A. reads a new round	3,582 – 4,530 seconds

*Table 5.2.3 – Time Testing*

Finally, we can establish that in almost all cases the operation of the application is very fast, a fact that is also consistent with the CPU usage figures. The only time the application needs more than 0.5 seconds to perform a reading and processing loop is when a new round is initiated and read, which usually happens once every minute.

## **Chapter 6 — Installation and user manual**

In this chapter we present each step that a user must take to successfully start the application, after which a short user manual will be presented.

### **6.1 Installation and running**

#### *6.1.1 Requirements*

For running the application the hardware requirements are:

- a monitor with resolution of 1920x1080p, or higher

The software requirements are:

- Eligible operating systems: Windows 10 or 11, Mac OS
- Python version minimum 3.10.1
- Pip minimum version 22.0.4
- PySimpleGUI minimum version v4.60.1
- NumPy version minimum 1.22.4
- Pyautogui minimum version 0.9.53
- OpenCV minimum version 4.5.5
- PyTesseract minimum version 0.3.9
- Tesseract minimum version 5.1.0
- PokerStars desktop app downloaded, installed
- Functional PokerStars account

#### *6.1.2 Satisfaction of requirements*

##### Hardware:

- The development of this application was made on a display of 1920x1080p. In the case of using the application on displays with different resolutions, it is recommended to change the resolution of the screen.

Resolution change guide for Windows: [https://support.microsoft.com/en-us/windows/view-display-settings-in-windows-37f0e05e-98a9-474c-317a-e85422daa8bb#WindowsVersion=Windows\\_11](https://support.microsoft.com/en-us/windows/view-display-settings-in-windows-37f0e05e-98a9-474c-317a-e85422daa8bb#WindowsVersion=Windows_11)



Guide to Mac OS: <https://support.apple.com/guide/mac-help/change-your-displays-resolution-mchl86d72b76/mac>

### Software:

- The Python language must be installed first. The link to download it we can find on the official website <https://www.python.org/downloads/>, or if we are on Windows, we can also download it from the Microsoft Store.

If we download it from the official website, once we have selected our operating system and pressed the version that we want to download, the download should begin. Once this is over, we must open the downloaded file, press "Install Now" and follow the instructions.

We have to be careful to check the box "Add Python to PATH", so the installer automatically makes it necessary to set the environment variables, in order to be able to run Python programs on our computer.

If we want to download it from the Microsoft Store, then we will write "Python" in the search bar of the application, press the version we want, and download it. In this way the installation and setting of the environmental variables are done automatically.

- Next we will restart the computer, and if we are on Windows, we click on the Start button, we look for "cmd" and pressing on it we open the terminal. If we are on Mac OS, we must search for and start the "Terminal" application.

- In the opened terminal we write "python --version", and if it writes us the version of Python that we have installed, then it means that we have done the right thing. If errors occur, repeat the steps from the beginning.

- In case we have succeeded, we will look at the version of pip with the command "pip --version", and if we do not have a corresponding version, we can download the newest version with "pip install --upgrade pip".

- We install the packages "pysimplegui", "numpy", "pyautogui", "pytesseract", "opencv-python" in the terminal with the help of the command "pip install <the name of the package>".

- For the PyTesseract package to work, we must install the Tesseract application, which is done from the <https://github.com/UB-Mannheim/tesseract/wiki> website where we choose and download the version we want.

## Poker Reader App

After downloading we click on the downloaded file, this will open the installer, and we follow the instructions until the end.

- The next step will be to download the PokerStars app, open and validate a new account, and the instructions for this step are well described on their official website <https://www.PokerStars.uk/en/poker/download/>.

- The last step will be to unzip the downloaded folder with the name "Poker Reader App.zip".

This on Windows is done with right click on the folder -> Extract All -> Extract.

On Mac OS the procedure consists of pressing Double Click on the folder, and the whole folder will unzip automatically.

### 6.1.3 Running the application

If so far we have done the steps successfully, we navigate to the unzipped folder where we find the files of the downloaded application, and we copy the absolute path.

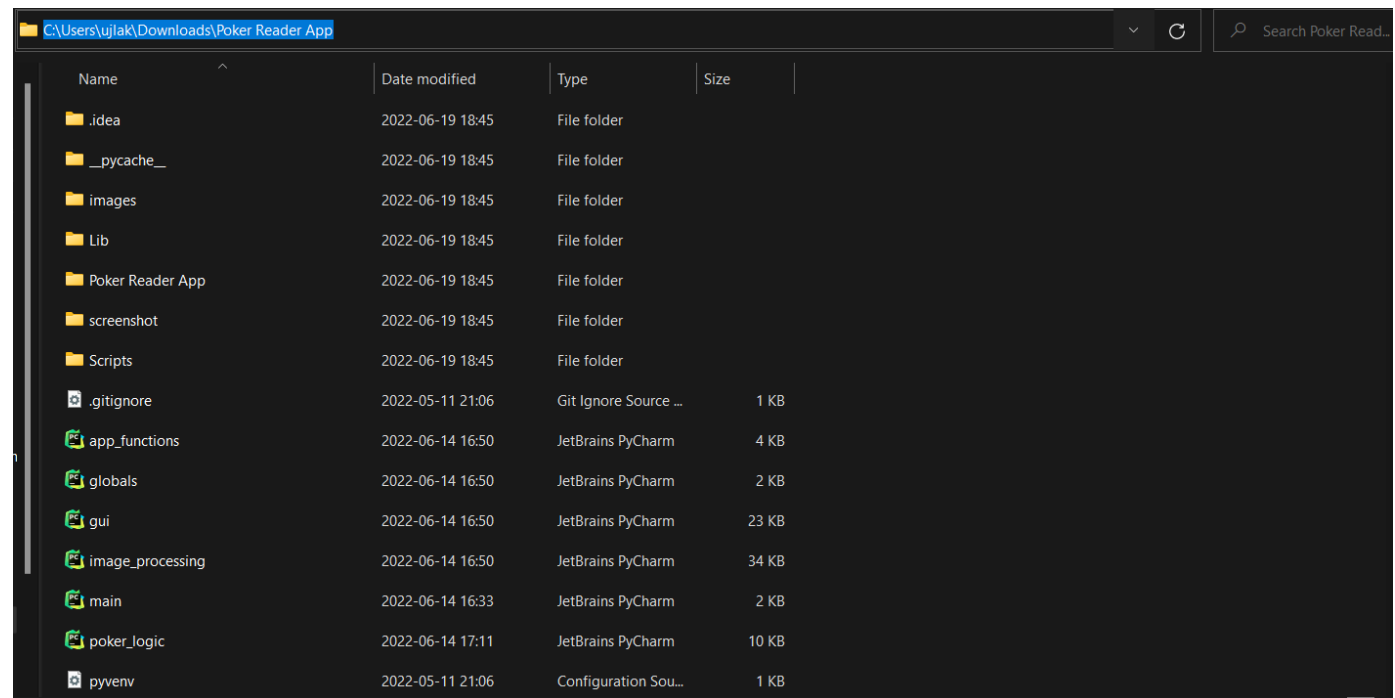


Image 6.1.3.1 – Application files

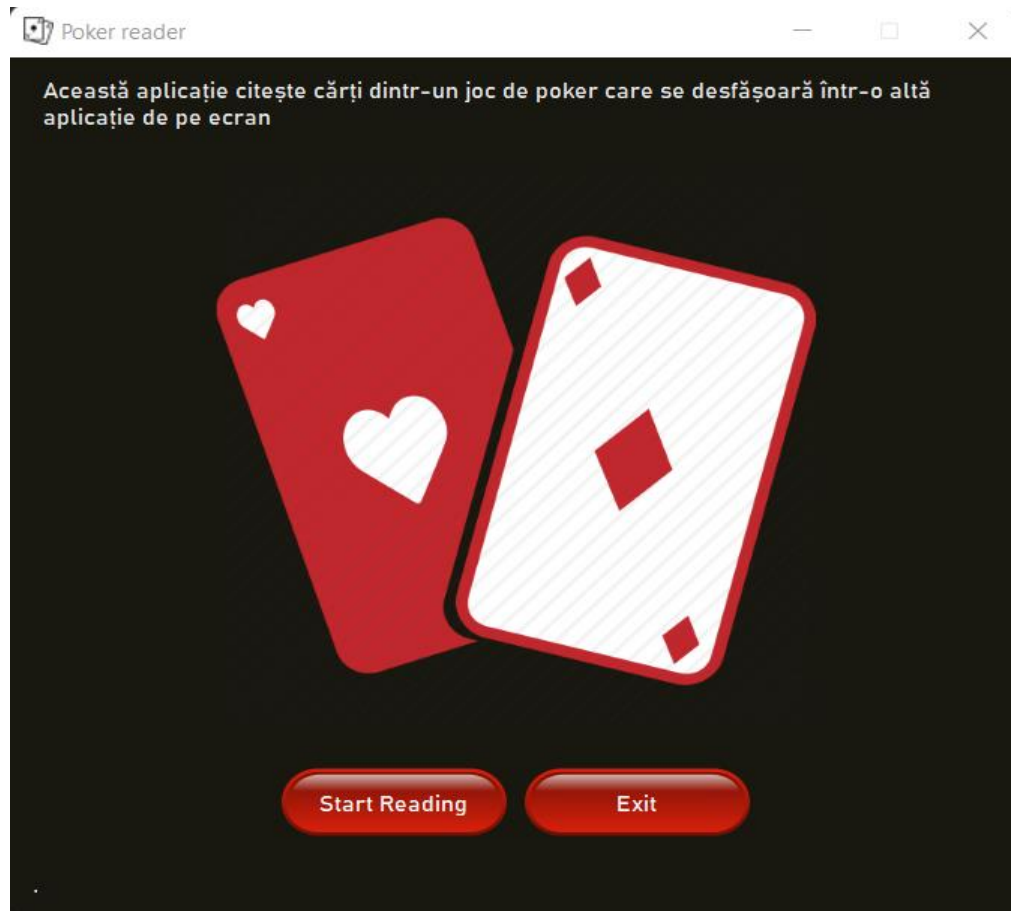
For example in this case the absolute path copied will be "C:\Users\ujlak\Downloads\Poker Reader App".

We open the Terminal and we will write "cd", we press the paste command (Ctrl+V on Windows), and when we press Enter, the command line should change to the path that we copied.

## *Poker Reader App*

The last step will be to write "python main.py", and press Enter.

If we have done everything correctly, the application opens (*Image 6.1.3.2*).



*Image 6.1.3.2 – Open app*

## **6.2 User manual**

### *6.2.1 Main instructions for use*

When starting the application we can read on the interface a brief description of what the application does, and we can press 2 different buttons.

- When pressing the "Exit" button the application closes, and if we want to start it again then we must repeat the instructions from [6.1.3](#).
- If you press the "Start Reading" button then the app starts reading the left side of the screen to find the PokerStars app where a game is being played, and the user interface changes to what is shown in

*Image 6.2.1.1.*

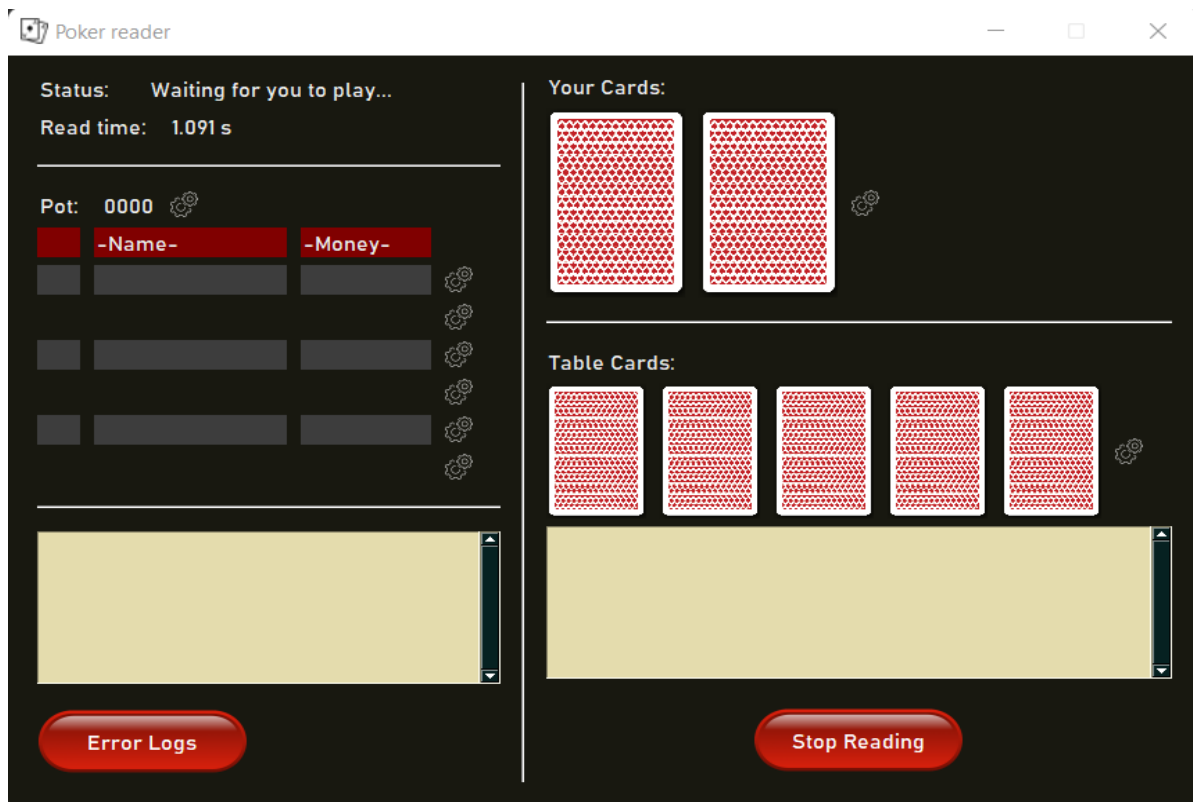


Image 6.2.1.1 – Application in read mode

In this state the application cannot read anything, which is what is signaled with the text "Waiting for you to play".

At this point we can press 2 buttons:

- "Stop Reading" will cause the application to finish the reading process and will return to the original state shown in *Image 6.1.3.2*.
- Pressing the "Error Logs" button will open a new part of the UI in which the reasons why the application is not able to read the data will be further displayed (*Image 6.2.1.2*).

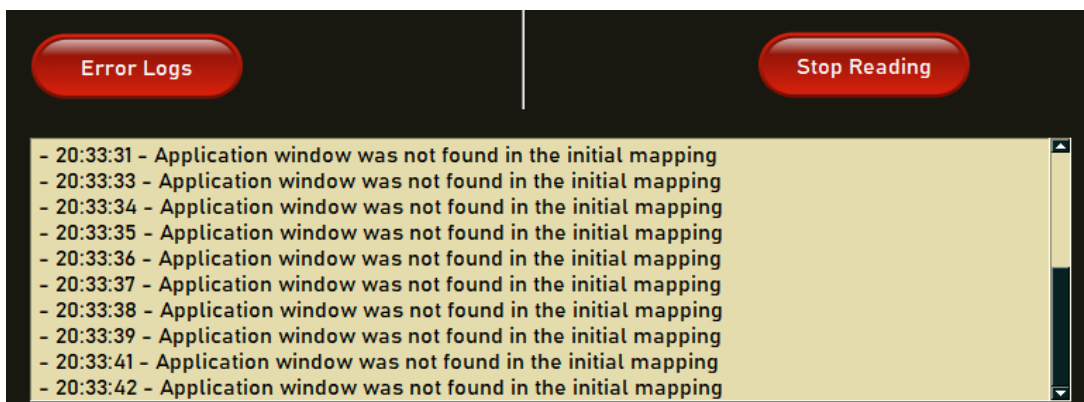


Image 6.2.1.2 – Error Logs

## Poker Reader App

We can at any time close the Error Logs tab with the press of the button with which we have opened it.

In Error Logs the errors appear in the following way: "<current hour, minute, second> <text of the error>".

On *Image 6.2.1.2* we can read "Application window was not found in the initial mapping" which signals that the PokerStars application cannot be found on the left side of the screen.

To rectify the problem, we open PokerStars and log in (*Image 6.2.1.3*).

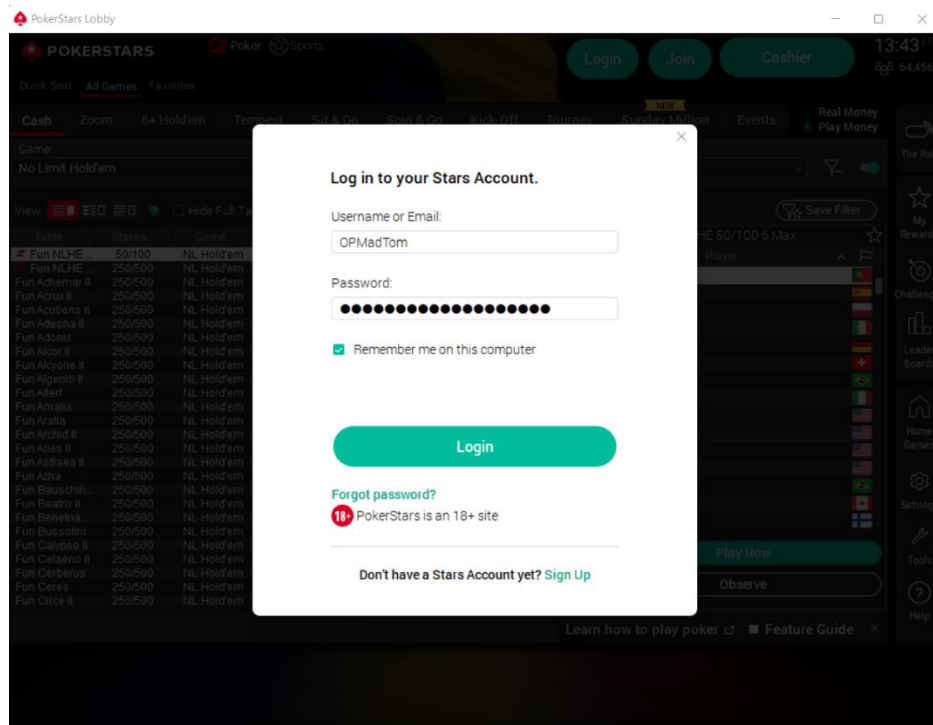


Image 6.2.1.3 – PokerStars Login

Through the list of active playing rooms, we choose one, and we press the "Play Now" button (*Image 6.2.1.4*).

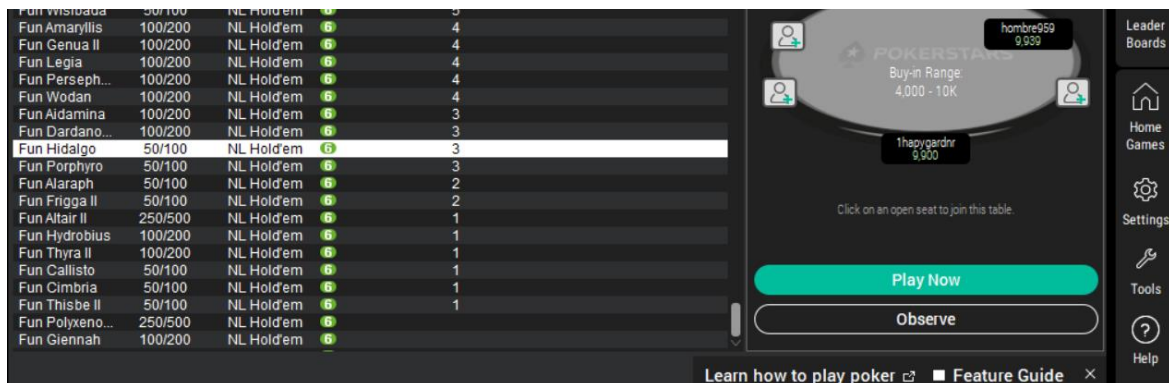


Image 6.2.1.4 – Room list

## Poker Reader App

When a game is in progress, you absolutely must position the PokerStars app to be entirely on the left side of the screen!

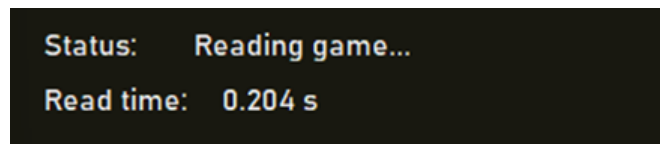
An example of the correct positioning can be seen in *Image 6.2.1.5*.



*Image 6.2.1.5 – Example of correct positioning*

### 6.2.2 UI components

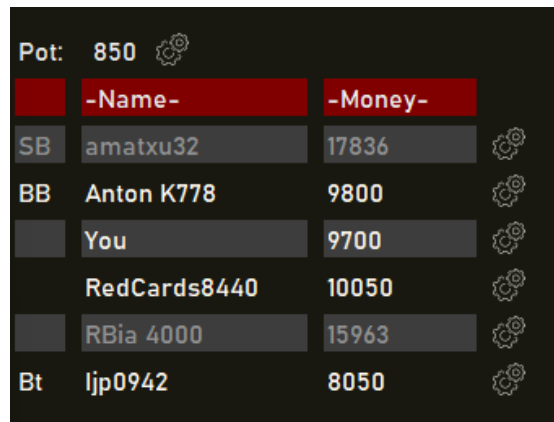
The UI of the application can be divided into 6+1 components.



*Image 6.2.2.1 – Status component*

-First component is the status in *Image 6.2.2.1*. This tells us what the application is doing at the moment, and much time has taken the last iteration of reading and processing.

In the present example, the status says "Reading game" which means that the reading of the game is proceeding as expected. We can also have here the text "Waiting for you to play" which means that the application has not yet found all the components whose presence indicates that the user is playing.

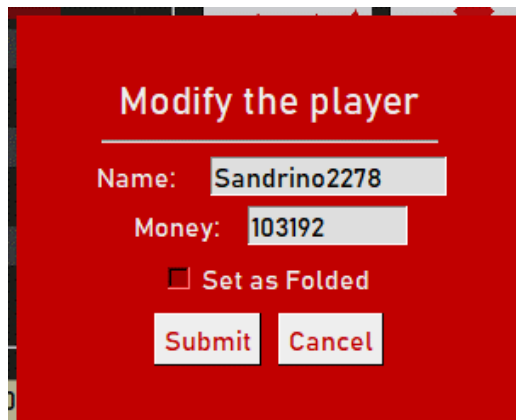


Pot: 850

	-Name-	-Money-	
SB	amatxu32	17836	
BB	Anton K778	9800	
	You	9700	
	RedCards8440	10050	
	RBia 4000	15963	
Bt	ljp0942	8050	

Image 6.2.2.2 – Player component

- The second component is a table that has the data of the players present, and on top we can see the total amount of money raised in that round (pot).
- In the table, players written in white are active in the round, and players written in gray have already folded their cards.
- On the first column are displayed the roles of the players: "SB" = small blind, "BB" = big blind, "Bt" = button.
- On the second column we can read the names of the players. The user of the application will receive the standard name "You" regardless of what his name is in the game.
- On the third column we can read the amount of money for each player.
- On the right side next to each row with the player we notice a small icon. This is the change button whose press opens a small window for us (Image 6.2.2.3) where we can modify the respective player.
- A similar window will appear if we press the change button of the pot, the user's cards, or the table cards.



Modify the player

Name: Sandrino2278

Money: 103192

☐ Set as Folded

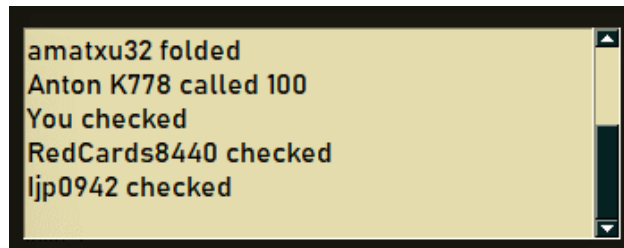
Submit Cancel

Image 6.2.2.3 - Change window



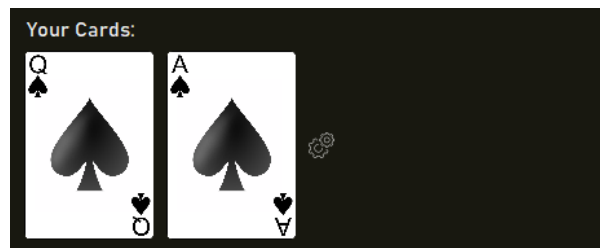
## *Poker Reader App*

An important aspect of the change buttons is that when we press them, the application tries to reread the specific information that we want to modify so that it has a chance to correct itself, and we can save the new information.



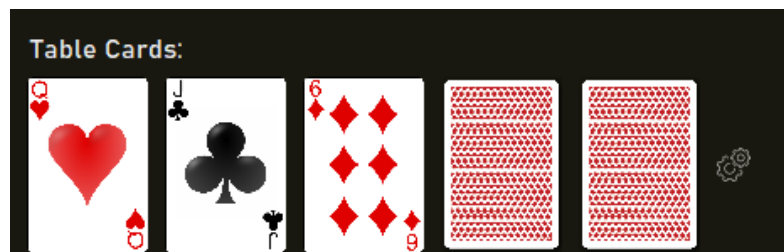
*Image 6.2.2.4 - Share component*

-The third component is the action field where the reading application will write what actions the players in this round have taken (*Image 6.2.2.4*).



*Image 6.2.2.5 – User's cards component*

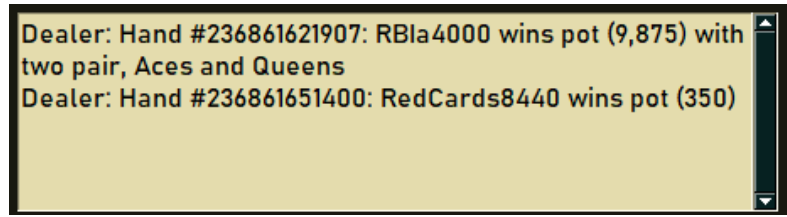
-The fourth component shows what cards the user has in the present round. We can see the change button on the right side of the cards (*Image 6.2.2.5*).



*Image 6.2.2.6 - Component with the table cards*

-The fifth component shows which community cards are on the table in the current round. We can see the change button on the right side of the cards (*Image 6.2.2.6*).





*Image 6.2.2.7 – Chat component*

-The sixth component is the chat field where the app writes the text it read from the PokerStars app's chat (*Image 6.2.2.7*).

-I consider an extra component the hideable Error Logs tab seen in *Image 6.2.1.2*.

## **Chapter 7 – Conclusions**

In this last chapter, I will analyze what objectives the application has achieved and possible further developments.

### **7.1 Achievement of objectives**

The Poker Reader App manages to achieve all the main goals that were set at the beginning. It successfully reads and processes the elements of a poker game and has all the features that a user may want. It also manages to be in an easy-to-edit form for the benefit of future improvements by any developer.

In terms of secondary objectives, Poker Reader App manages to complete a reading loop under 0.5 seconds in most cases, so it is a fast application, it has a modern and intuitive user interface, it uses very small portions of the computer's processing power, the code is edited, commented in detail. Problems during operation are few in the current version, it can be used on Windows 10, 11 and Mac OS, and is written exclusively in Python.

### **7.2 Further developments**

Although the Poker Reader App is a working app, it can be improved in many ways.

The first and most important way is to improve the process that determines the actions of players based on the data read from the screen.

An important function that would be very beneficial is editing the app so that it can read from several poker game apps, not just PokerStars. This would mean that the app must make an initialization for each poker game application, and create and save the parameters from them. With this improvement, there would be only one initialization for each different application.

Another improvement could be to make Poker Reader App work on multiple resolutions. This means changing the user interface and reading functions so that they adapt to the size of the screen.

It is also important to create an internal system for OCR functions, and to avoid the use of Tesseract.

## *Poker Reader App*

Once these improvements are made and we have a perfectly functional and versatile on-screen poker reading app, we can customize it according to our needs. It can turn into a poker game robot, or there can be added "text to speech" to help visually impaired people play poker, or one could also build features that analyze the user's game and offer advice.

## **Bibliography**

- [1] [https://en.wikipedia.org/wiki/Pluribus\\_\(poker\\_bot\)](https://en.wikipedia.org/wiki/Pluribus_(poker_bot))
- [2] <https://en.wikipedia.org/wiki/Poker>
- [3] [https://en.wikipedia.org/wiki/Texas\\_hold\\_%27em](https://en.wikipedia.org/wiki/Texas_hold_%27em)
- [4] <https://upswingpoker.com/is-poker-a-sport-or-a-game/>
- [5] [https://en.wikipedia.org/wiki/Digital\\_image\\_processing](https://en.wikipedia.org/wiki/Digital_image_processing)
- [6] [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [7] <https://pypi.org/project/PySimpleGUI/>
- [8] <https://en.wikipedia.org/wiki/OpenCV>
- [9] <https://opencv.org/about/>
- [10] <https://pypi.org/project/pytesseract/>
- [11] <https://en.wikipedia.org/wiki/PyCharm>
- [12] <https://www.techtarget.com/searcharchitecture/definition/object-oriented-programming-OOP>
- [13] <https://www.pluralsight.com/blog/software-development/10-steps-to-clean-code>
- [14] [https://en.wikipedia.org/wiki/Unified\\_Modeling\\_Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language)
- [15] <https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-use-case>
- [16] [https://ebrary.net/73282/computer\\_science/system\\_actors](https://ebrary.net/73282/computer_science/system_actors)
- [17] <https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-class>
- [18] <https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-activity>