DE LA SALLE UNIVERSITY - MANILA

BLACKJACK

A Term Project

Presented to Mr. Ruiz Stephen L. Ruiz

In Partial Fulfillment of the

Requirements for the Programming Logic and Design Laboratory (LBYCPA1)

by

MACARANAS, Percival III Quinto [illegible]

NARVAJA, Shawn Karlsten Atacador [illegible]

ONG, Franco Miguel Carlos [illegible]

EQ4

LBYCPA1, Monday & Thursday; 0730-1030

April 21, 2023

RUBRIC FOR TERM PROJECT

CRITERIA	EXEMPLARY (90-100)	SATISFACTORY(80-89)	DEVELOPING (70-79)	BEGINNING (below 70)	WEIGHT
Experimental Plan (Flowchart/ Algorithm) ***(SO-PI: B1)***	Experimental plan has supporting details and diagram/algorithm hm that is stated and well explained	Experimental plan has supporting details and diagram/algorithm m that is stated but not explained	Experimental plan is vague or brief. It has supporting details and does not have diagram/ algorithm	No experimental plan presented	20%
Codes/Data/Program	Data is well utilized in the program. Program code are easy to read. Program output has no error. Questions are answered completely and correctly	Data is somewhat utilized in the program. Program code are easy to read. Program output has an output but logically incorrect. Some questions are answered completely and correctly	Data is not utilized in the program. It has a missing significant code/syntax in the program	No program presented	30%
	Appropriate	Appropriate	Appropriate	Appropriate	10%

TABLE OF CONTENTS

LIST OF FIGURES

Figures

LIST OF TABLES

I. Introduction

This chapter introduced the basic knowledge of Blackjack and the programming language Python. It also noted the purpose and significance of the study.

A. Background of the Study

Blackjack is a popular card game played in numerous situations, from casinos to home games. It is a card game played with the standard 52-card deck and, to put it simply, the goal of the game is to acquire cards which accumulate to a value as close to 21 without going higher (BicycleCards, n.d.). The proponents of the paper are freshman undergraduates of De La Salle University taking the course LBYCPA1; though some may have some background in programming, none have any experience in using the Python programming language.

Considering the above mentioned factors, a project involving programming a blackjack game is a suitable way for the proponents to apply learnings in their course of LBYCPA1 as it is simple enough conceptually for the project to not be so convoluted, but is contains enough mechanics where most to all learnings in the course are applied.

The project aims to create a blackjack game using the Python programming language. However, the blackjack game will be non-traditional in the sense that the limit can be adjusted to determine its difficulty.

B. Problem Statement

The project is focused on the development of a program that can accurately simulate a game of blackjack, including dealing cards, calculating scores, handling bets, and determining the winner.

C. Objectives

C.1 General Objective

Develop a blackjack game using the Python programming language.

C.2 Specific Objective

1. Implement a card deck that can shuffle and deal cards.
2. Develop a system to calculate scores based on the values of the cards dealt.
3. Develop a system to calculate the total money the player has across multiple games.
4. Develop a system to determine the winner of the game based on the scores of both player and dealer.
5. Develop a system to determine what difficulty the player wants to set the blackjack to be.

D. Significance of the Project

The project would benefit programmers by helping them to understand the fundamental basics and principles of programming in Python and other possible programming languages. Similarly, individuals would experience recreation and entertainment by simply running the programming language through interpretation, reading, and coding. Improving their programming skills in Python would result in higher proficiency.

II. Review of Related Literature

The studies tackled within this section will benefit the project with the necessary knowledge about Blackjack, Python, and PyQt of the program.

Python and PyQt5

Python is a programming language that is widely used for various purposes, including web development, data analysis, and machine learning. According to Python (n.d.), Python is a programming language that is object-oriented, high-level, and interpreted, with dynamic semantics. It is very appealing for Rapid Application Development because of its high-level data structures, dynamic typing, and dynamic binding, which allow developers to easily connect existing components. Python's syntax is easy to learn and emphasizes readability, which reduces the cost of program maintenance. Python supports modules and packages, promoting code reuse and program modularity. The Python interpreter and standard library are available for all major platforms in source or binary form at no cost and can be distributed freely. It is an easy-to-learn language that is ideal for beginners, but it is also powerful enough to be used for complex applications. Python has many libraries that make it easy to develop games, including Pygame and PyQt. According to Costa (2023), PyQt and Python are widely utilized in creating graphical user interfaces, and are well-liked by game developers because of their user-friendly nature and ease of use.

PyQt, which is a collection of Python bindings for the Qt application framework, is frequently used in the development of cross-platform desktop applications. PyQt is instrumental in building graphical user interfaces (GUIs) for Python applications, which is crucial in the creation of a blackjack game that has a visual interface.

Blackjack

Blackjack, also known as 21, is a popular card game that has been implemented in various programming languages. According to BicycleCards (n.d), Blackjack is a casino game that has become very popular because of its favorable odds for players. It originated in France in the 1760s and is known as Vingt-et-Un, which means 21 in French. It is now played in every American casino and also commonly played as a home game with slightly different rules. In the casino version, the dealer is the house and remains standing while the players are seated. The dealer manages the game and handles all bets. In the home game, players take turns being the dealer. In recent years, there has been a growing interest in coding and building Blackjack games using the Python programming language and creating user interfaces using PyQt.

III. Methodology

The project will be written in the Python programming language, this will be the primary foundation and basis of this project.. For its User Interface & User Experience (UI/UX), presentation and accessibility overall, the proponents will use the PyQt library for creating those elements. For the 52 playing cards art assets, we will use the "Open Source Vector Playing Cards" for the blackjack game overall, and the favicon for the program is the “Playing Card” icon by Freepik.

The process of developing a game requires a well-defined workflow that includes various stages, starting from requirements gathering to the final delivery of the program. The following section will discuss each of these stages in detail, outlining their importance and the tasks involved in each of them.

Requirements Gathering

The first stage of the development process is requirements gathering, where the development team assesses the purpose and goals of the program. The team collects information about the material requirements and game mechanics. The team also analyzes the resources available for the project and determines a timeline for the game's completion.

Design UX

The design UX stage involves creating the game's user experience (UX) elements. This includes designing the game's user interface (UI), graphics, sound effects, and music. The design team creates mockups and prototypes, which they use to test and refine the game's UX. During this stage, the team also develops a style guide to ensure that all design elements are consistent throughout the game (Pinto, 2018).

Design Game Logic

Once the UX is designed, the next stage involves designing the game logic. This includes developing the game's rules, objectives, and mechanics. The game design team creates a game design document (GDD), which outlines the game's concept, features, and mechanics. The GDD serves as a reference for the rest of the development team and helps them understand how the game should work (Muldoon, 2021).

Implement UX

Once the game logic is implemented, the team moves onto the implementation of UX. This involves integrating the design elements into the game, including the UI, graphics, sound effects, and music. The team also ensures that the game is responsive and runs smoothly across different platforms.

Testing and Debugging

Once the game is fully implemented, the team begins testing and debugging the game. This involves identifying and fixing any bugs, glitches, or other issues that may affect the game's performance or playability. The team also conducts user testing to ensure that the game is intuitive and enjoyable to play.

Finalize Deliverables

After the testing and debugging stage, the team finalizes the game's deliverables. The team also ensures that the program meets all assigned parameters given by the instructor.

A. Conceptual Framework – IPO Chart (Input-Process-Output-Chart)

This section tackles the conceptual framework mainly the input, process, output of the project.

Input

The input is the data entered into the program by the user.

Player's Hand (Hit or Stand)

The decision the user must make during their turn in Blackjack. If the user chooses to "hit," they are requesting an additional card from the dealer. If they choose to "stand," they are satisfied with their current hand and do not want any additional cards.

Player's Bet

The amount of money the user is willing to wager on a hand of blackjack. The bet determines the potential payout if the user wins the hand. The user has five options for betting starting from $1, $5, $10, $50, and a specified number the user chooses.

Player's Amount of Money

The amount of money the user currently has available to bet with. It's important to keep track of this amount throughout the game, as users should never bet more than they can afford to lose. Additionally, if a user runs out of money, they will no longer be able to participate in the game. The user can change the amount of money they have within the program.

Player's Chosen Difficulty

The level of challenge the user has selected for the game. Choosing the appropriate difficulty level can be crucial for users to ensure they have an enjoyable and fair gaming experience. The user can change the difficulty level within the program from Easy (30), Standard (21), Intermediate (16), and Hard (12).

Process

The process is associated with resources and information that are necessary for the program to function properly.

Setting the Difficulty

The user can choose to adjust the level of difficulty of the game by setting a custom limit for the game. By selecting Easy (30), Intermediate (16) and Hard (12), the user can modify the limit of the game, which is traditionally set at Standard (21). This adjustment can make the game harder or easier, depending on the user's preference.

Betting on the Game

The user can wager money on the outcome of the game. If the user wins the game, their corresponding bet will be added to their current amount of money, and they will receive a payout based on the size of their wager. However, if the user loses the game, they will forfeit their bet and lose that amount of money.

Playing the Game

The user must decide whether to "hit" or "stand" based on their current hand and their evaluation of the likelihood of winning the game. The user will be evaluated based on the corresponding difficulty. The Standard difficulty (21) will be evaluated. If the user chooses to "hit," the dealer will deal another card to their hand, which will increase the total value of their hand. If the user chooses to "stand," the dealer will reveal their cards, and the user’s hand will be compared to the dealer's hand. If the user’s hand value is greater than the dealer's hand value and less than or equal to 21, the user wins the game. However, if the dealer's hand value is greater than the user’s hand value and less than or equal to 21, the user loses the game. If the user’s hand value exceeds 21, the user busts and automatically loses the game, regardless of the dealer's hand value. These rules apply to the other difficulties Easy (30), Intermediate (16), and Hard (12).

Output

The output is the results the program produces.

Player's Total Hand Value

The total value of the player's hand at the end of the game. The total hand value is calculated by adding up the point values of each card in the player's hand. The value of each card is determined by its rank, with face cards (Jacks, Queens, and Kings) being worth 10 points and Aces being worth either 1 or 11 points, depending on the player's preference. This output is important because it determines whether the player has won, lost, or tied with the dealer.

Game Evaluation

The outcome of the game for the player. If the player's total hand value is greater than the dealer's total hand value and less than or equal to 21, the player wins. If the dealer's total hand value is greater than the player's total hand value and less than or equal to 21, the player loses. If both the player and the dealer have the same total hand value, the game ends in a tie or push. This output is crucial because it determines whether the player will receive a payout or lose their bet.

Player's Total Money

The total amount of money the player has left after the game is over. If the player wins the game, their total money will increase by the amount of the payout they receive. If the player loses the game, their total money will decrease by the amount of their bet. This output is essential because it determines whether the player can continue playing future games or if they have run out of money.

Dealer's Face-up Card:

The value of the dealer's face-up card, which is the card that is visible to the player. In blackjack, the dealer is dealt one face-up card and one face-down card, and the player must try to beat the value of the dealer's visible card. The value of the dealer's face-up card can help the player make decisions about whether to hit or stand based on the likelihood of the dealer busting.

Total Number of Games

The total number of games played by the player. This output can be helpful for the player to track their progress over time.

Table 1
Input-Process-Output-Chart

Input	Process	Output
● Player's hand (Hit or Stand) ● Player's bet ● Player's amount of money ● Player's chosen difficulty	***Setting the Difficulty*** - The difficulty will be determined based on what the player wants to set the limit to be. For instance, instead of the traditional limit of 21 in blackjack, the user can input it as 16 to make it harder or 30 to make it easier.	● The player's total hand value ● Whether the player won, lost, or tied with the dealer (Game Evaluation) ● The player's total money ● Dealer's face-up card

Table 2

Input-Process–Output-Chart

Input	**Process**	**Output**
	Betting on the Game - If the user wins in blackjack, the corresponding bet they placed will be doubled. ***Playing the Game*** - The player must decide whether to "hit" (request another card) or "stand" (keep their current hand). - The player must evaluate the likelihood of busting (exceeding 21) if they hit. - The player must evaluate the likelihood of beating the dealer's hand if they stand.	● Total number of games

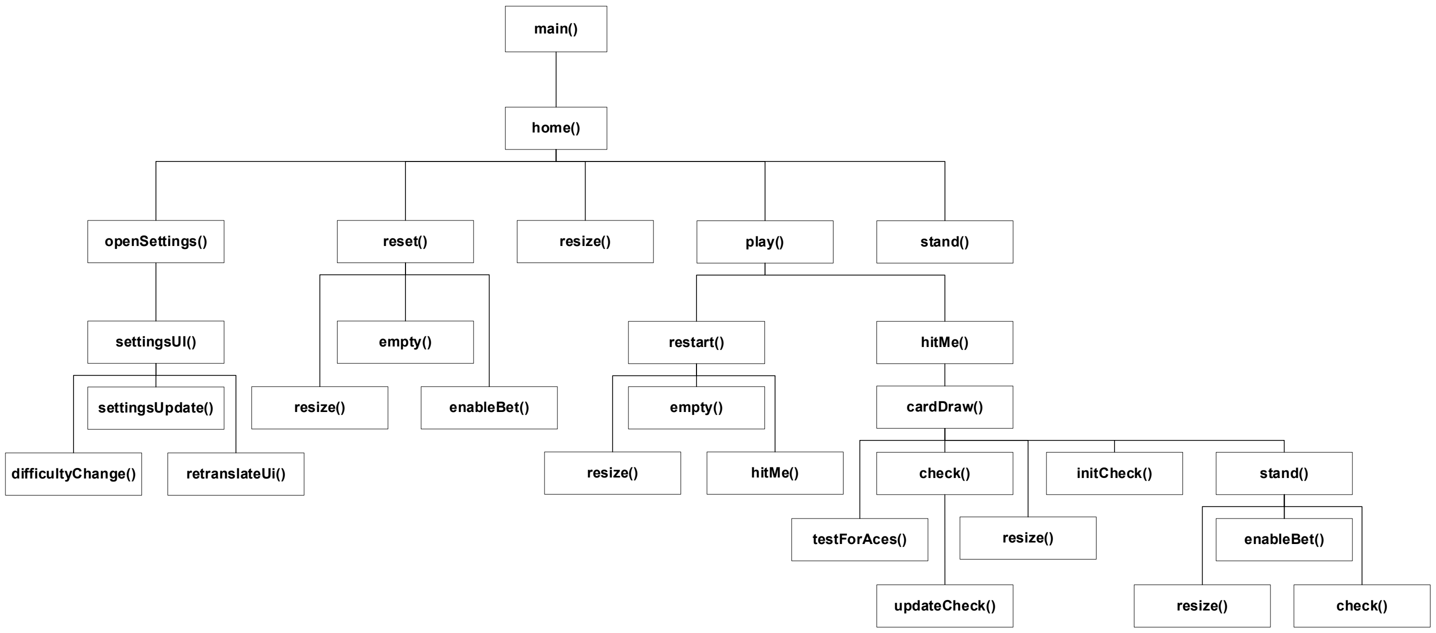
Table 3
Input-Process-Output

Input	Process	Output
	Playing the Game - If the user inputs "Hit", a card would be added to the corresponding card on the table. - If the user inputs "Stand", the game stops and the AI opponent will reveal their cards. - If the user's card total is greater than the AI's card total and less than or equal to 21, the user wins. - If the user's card total is less than the AI's card total and the AI's card total is less than or equal to 21, then user losses.	

B. Hierarchy Chart

Figure 1

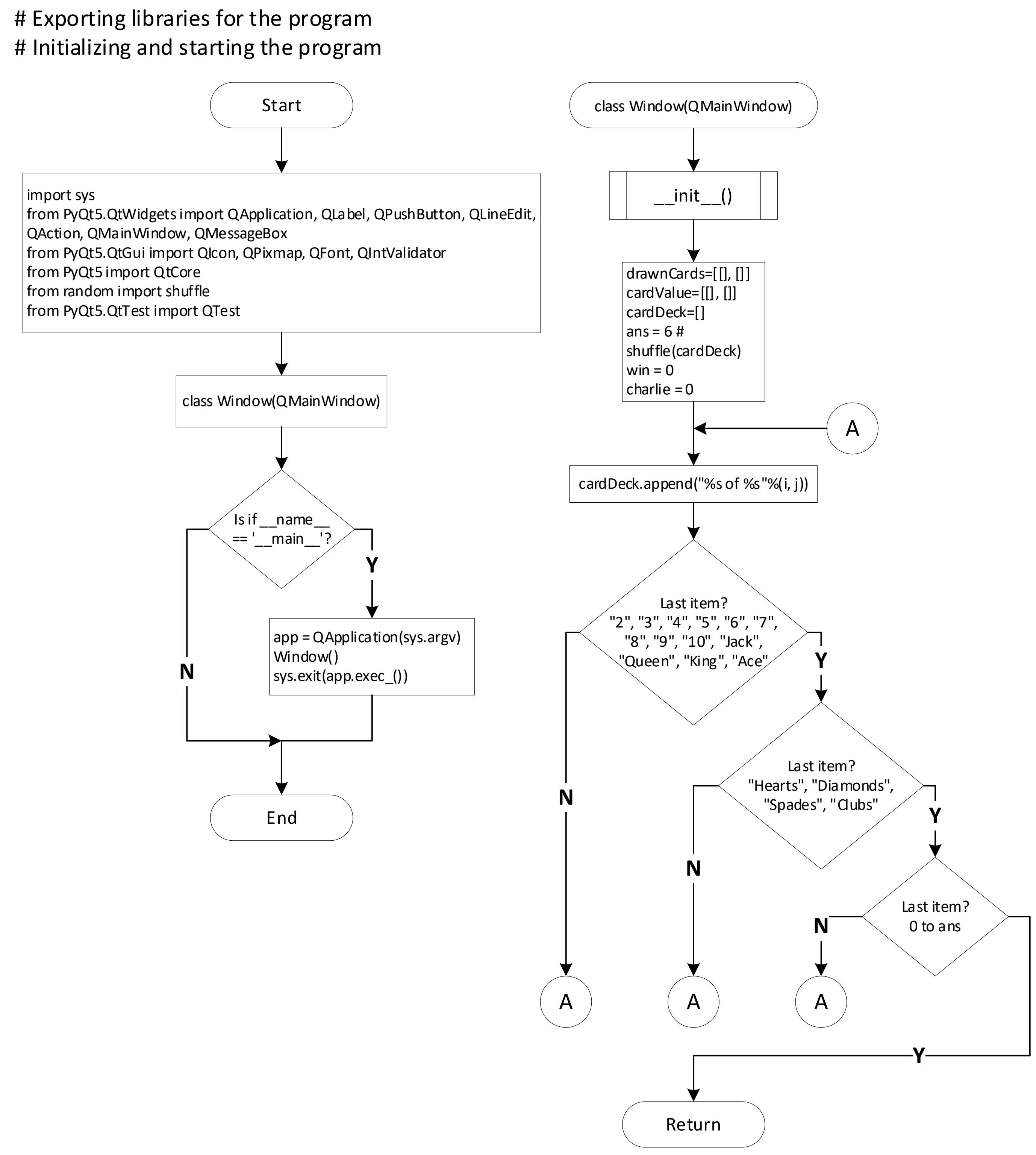
Hierarchy Chart



C. Flowchart

Figure 2

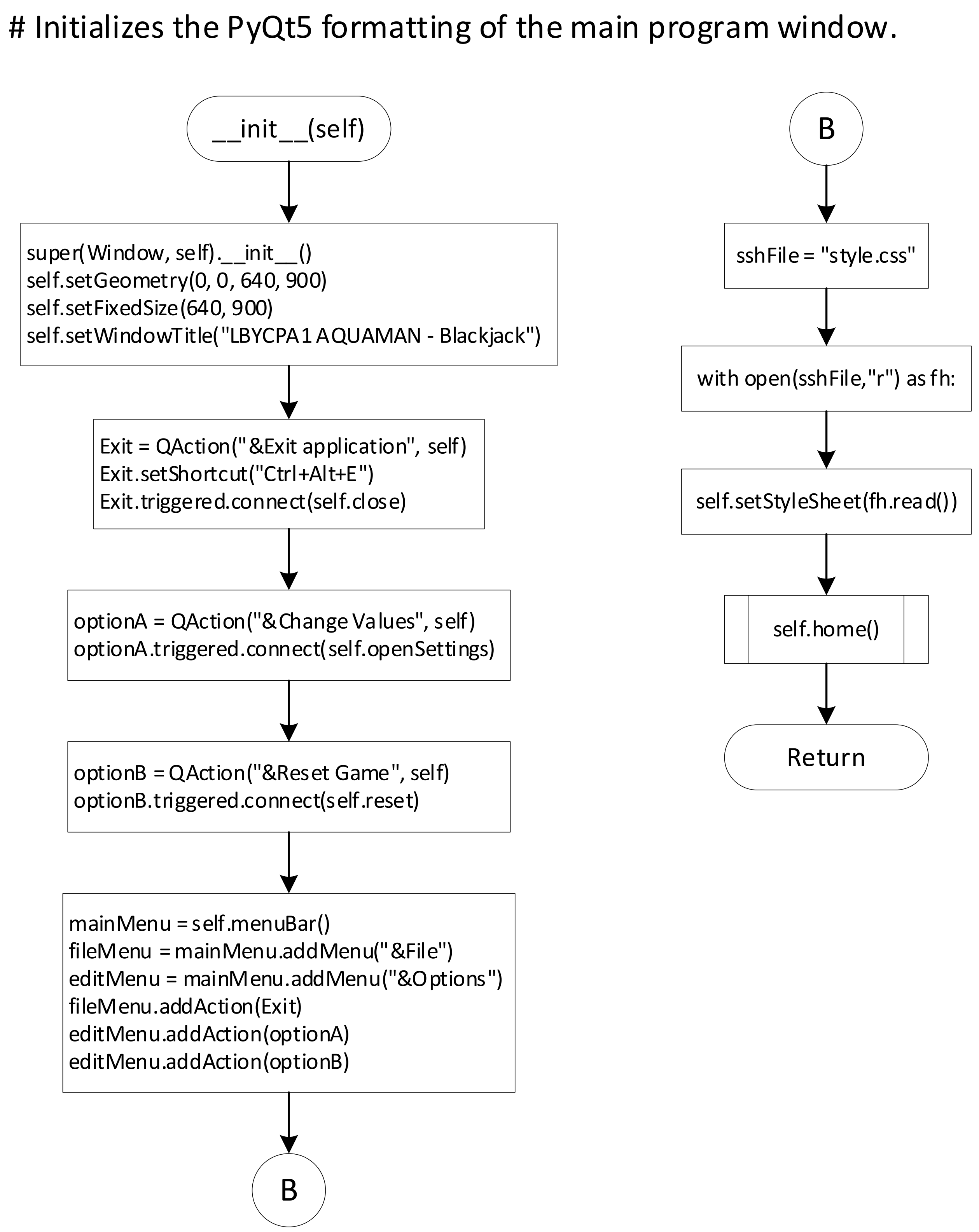
Exporting Libraries, Initializing and Starting the Program



# Exporting libraries for the program
# Initializing and starting the program
Start
import sys
from PyQt5.QtWidgets import QApplication, QLabel, QPushButton, QLineEdit, QAction, QMainWindow, QMessageBox
from PyQt5.QtGui import QIcon, QPixmap, QFont, QIntValidator
from PyQt5 import QtCore
from random import shuffle
from PyQt5.QtTest import QTest
class Window(QMainWindow)
Is if __name__ == '__main__'?
Y
N
app = QApplication(sys.argv)
Window()
sys.exit(app.exec_())
End
class Window(QMainWindow)
__init__()
drawnCards=[[], []]
cardValue=[[], []]
cardDeck=[]
ans = 6 #
shuffle(cardDeck)
win = 0
charlie = 0
A
cardDeck.append("%s of %s"%(i, j))
Last item? "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"
Y
N
Last item? "Hearts", "Diamonds", "Spades", "Clubs"
Y
N
Last item? 0 to ans
N
A
A
A
Y
Return

Figure 3

Initializing PyQt5


# Initializes the PyQt5 formatting of the main program window.
__init__(self)
B
Return

```
super(Window, self).__init__()
self.setGeometry(0, 0, 640, 900)
self.setFixedSize(640, 900)
self.setWindowTitle("LBYCPA1 AQUAMAN - Blackjack")

Exit = QAction("&Exit application", self)
Exit.setShortcut("Ctrl+Alt+E")
Exit.triggered.connect(self.close)

optionA = QAction("&Change Values", self)
optionA.triggered.connect(self.openSettings)

optionB = QAction("&Reset Game", self)
optionB.triggered.connect(self.reset)

mainMenu = self.menuBar()
fileMenu = mainMenu.addMenu("&File")
editMenu = mainMenu.addMenu("&Options")
fileMenu.addAction(Exit)
editMenu.addAction(optionA)
editMenu.addAction(optionB)

sshFile = "style.css"

with open(sshFile,"r") as fh:

self.setStyleSheet(fh.read())

self.home()
```

Figure 4

home() Function

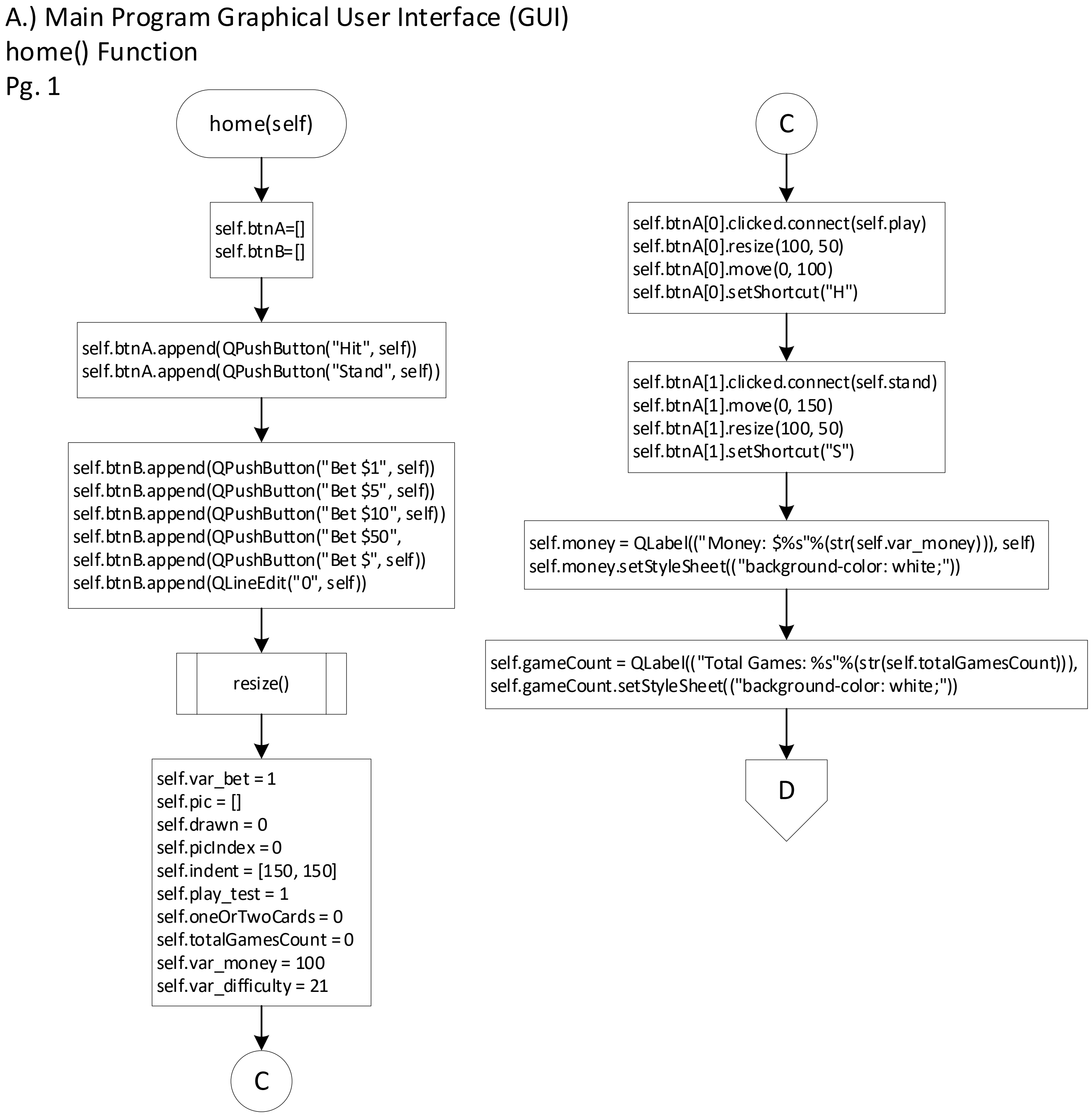
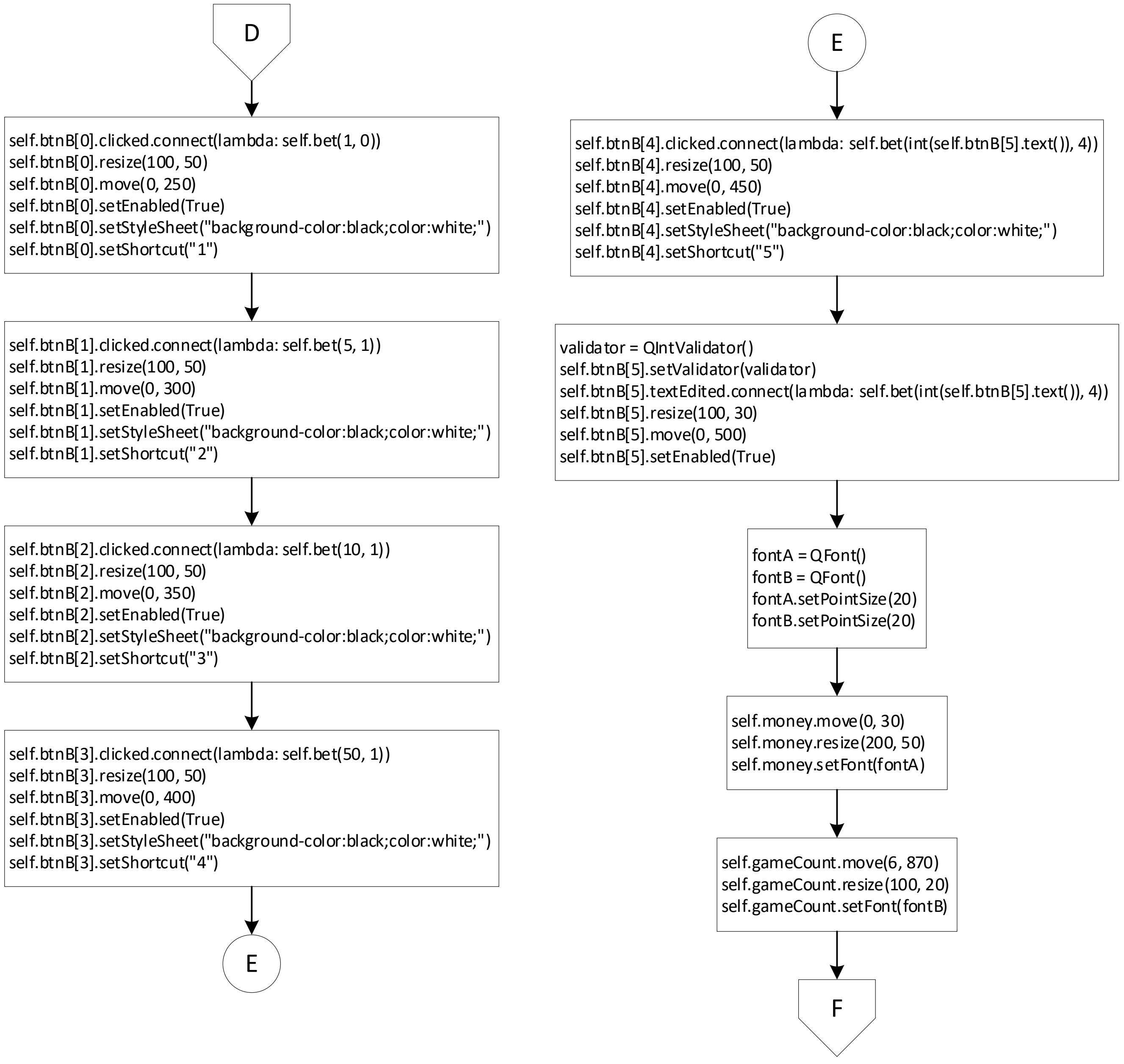

A.) Main Program Graphical User Interface (GUI)
home() Function
Pg. 1
home(self)
self.btnA=[]
self.btnB=[]
self.btnA.append(QPushButton("Hit", self))
self.btnA.append(QPushButton("Stand", self))
self.btnB.append(QPushButton("Bet $1", self))
self.btnB.append(QPushButton("Bet $5", self))
self.btnB.append(QPushButton("Bet $10", self))
self.btnB.append(QPushButton("Bet $50",
self.btnB.append(QPushButton("Bet $", self))
self.btnB.append(QLineEdit("0", self))
resize()
self.var_bet = 1
self.pic = []
self.drawn = 0
self.picIndex = 0
self.indent = [150, 150]
self.play_test = 1
self.oneOrTwoCards = 0
self.totalGamesCount = 0
self.var_money = 100
self.var_difficulty = 21
C
C
self.btnA[0].clicked.connect(self.play)
self.btnA[0].resize(100, 50)
self.btnA[0].move(0, 100)
self.btnA[0].setShortcut("H")
self.btnA[1].clicked.connect(self.stand)
self.btnA[1].move(0, 150)
self.btnA[1].resize(100, 50)
self.btnA[1].setShortcut("S")
self.money = QLabel(("Money: $%s"%(str(self.var_money))), self)
self.money.setStyleSheet(("background-color: white;"))
self.gameCount = QLabel(("Total Games: %s"%(str(self.totalGamesCount))),
self.gameCount.setStyleSheet(("background-color: white;"))
D

Figure 5

home() Function

A.) Main Program Graphical User Interface (GUI)
home() Function
Pg. 2



```
self.btnB[0].clicked.connect(lambda: self.bet(1, 0))
self.btnB[0].resize(100, 50)
self.btnB[0].move(0, 250)
self.btnB[0].setEnabled(True)
self.btnB[0].setStyleSheet("background-color:black;color:white;")
self.btnB[0].setShortcut("1")
```

```
self.btnB[1].clicked.connect(lambda: self.bet(5, 1))
self.btnB[1].resize(100, 50)
self.btnB[1].move(0, 300)
self.btnB[1].setEnabled(True)
self.btnB[1].setStyleSheet("background-color:black;color:white;")
self.btnB[1].setShortcut("2")
```

```
self.btnB[2].clicked.connect(lambda: self.bet(10, 1))
self.btnB[2].resize(100, 50)
self.btnB[2].move(0, 350)
self.btnB[2].setEnabled(True)
self.btnB[2].setStyleSheet("background-color:black;color:white;")
self.btnB[2].setShortcut("3")
```

```
self.btnB[3].clicked.connect(lambda: self.bet(50, 1))
self.btnB[3].resize(100, 50)
self.btnB[3].move(0, 400)
self.btnB[3].setEnabled(True)
self.btnB[3].setStyleSheet("background-color:black;color:white;")
self.btnB[3].setShortcut("4")
```

```
self.btnB[4].clicked.connect(lambda: self.bet(int(self.btnB[5].text()), 4))
self.btnB[4].resize(100, 50)
self.btnB[4].move(0, 450)
self.btnB[4].setEnabled(True)
self.btnB[4].setStyleSheet("background-color:black;color:white;")
self.btnB[4].setShortcut("5")
```

```
validator = QIntValidator()
self.btnB[5].setValidator(validator)
self.btnB[5].textEdited.connect(lambda: self.bet(int(self.btnB[5].text()), 4))
self.btnB[5].resize(100, 30)
self.btnB[5].move(0, 500)
self.btnB[5].setEnabled(True)
```

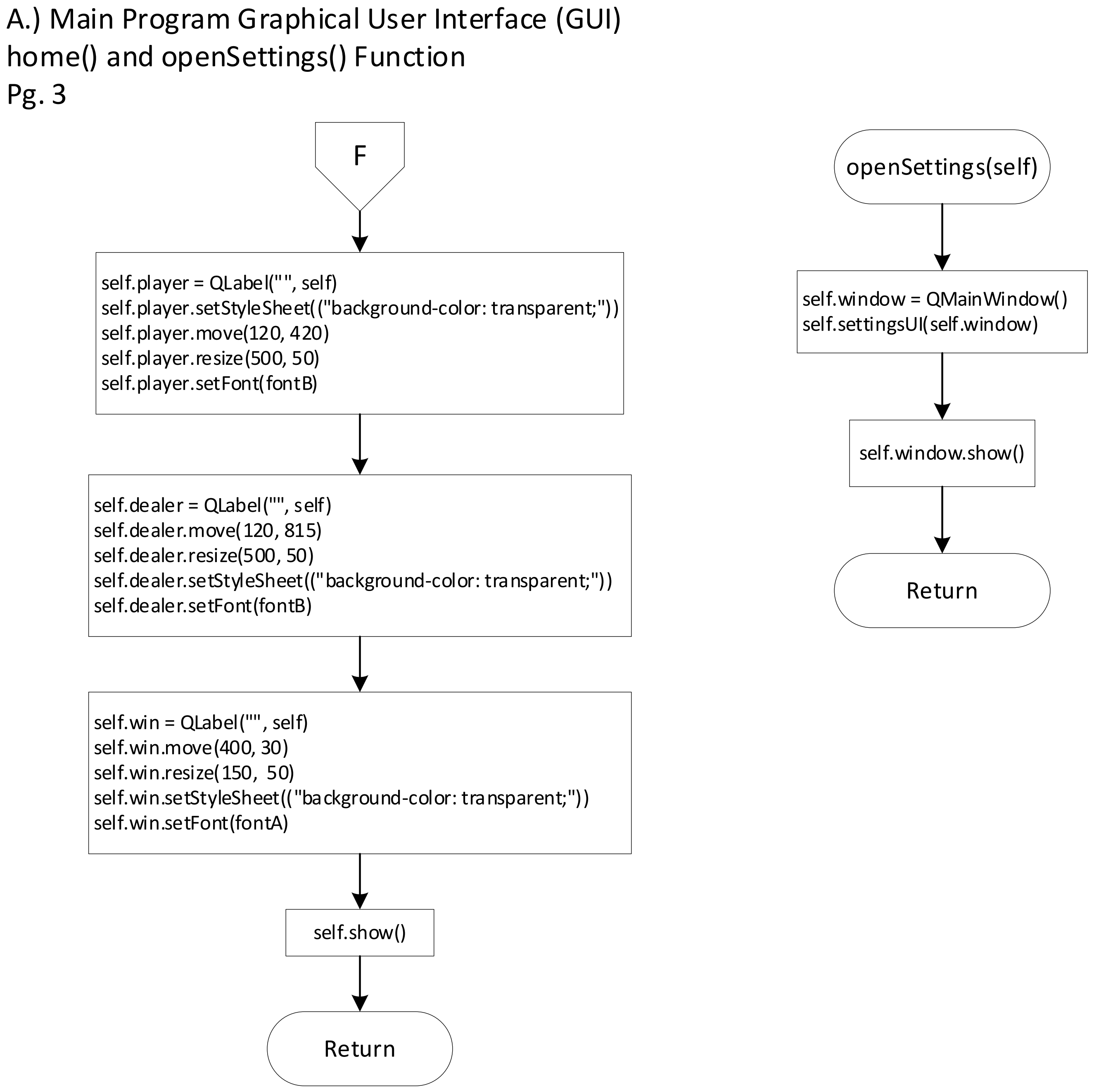
```
fontA = QFont()
fontB = QFont()
fontA.setPointSize(20)
fontB.setPointSize(20)
```

```
self.money.move(0, 30)
self.money.resize(200, 50)
self.money.setFont(fontA)
```

```
self.gameCount.move(6, 870)
self.gameCount.resize(100, 20)
self.gameCount.setFont(fontB)
```

Figure 6

home() & openSettings() Function


A.) Main Program Graphical User Interface (GUI)
home() and openSettings() Function
Pg. 3
F
Return
openSettings(self)
Return

```
self.player = QLabel("", self)
self.player.setStyleSheet(("background-color: transparent;"))
self.player.move(120, 420)
self.player.resize(500, 50)
self.player.setFont(fontB)
```

```
self.dealer = QLabel("", self)
self.dealer.move(120, 815)
self.dealer.resize(500, 50)
self.dealer.setStyleSheet(("background-color: transparent;"))
self.dealer.setFont(fontB)
```

```
self.win = QLabel("", self)
self.win.move(400, 30)
self.win.resize(150,  50)
self.win.setStyleSheet(("background-color: transparent;"))
self.win.setFont(fontA)
```

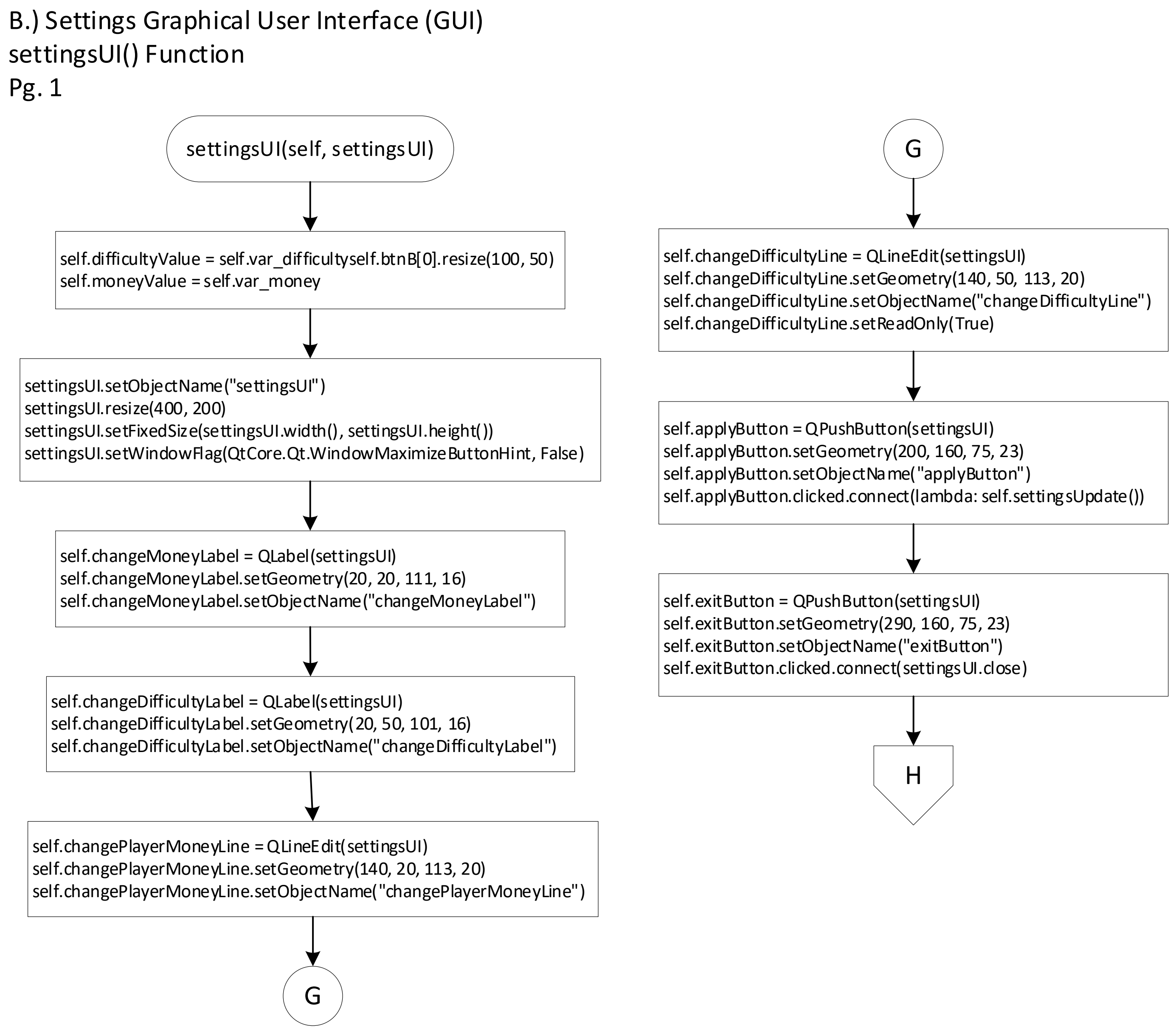
```
self.show()
```

```
self.window = QMainWindow()
self.settingsUI(self.window)
```

```
self.window.show()
```

Figure 7

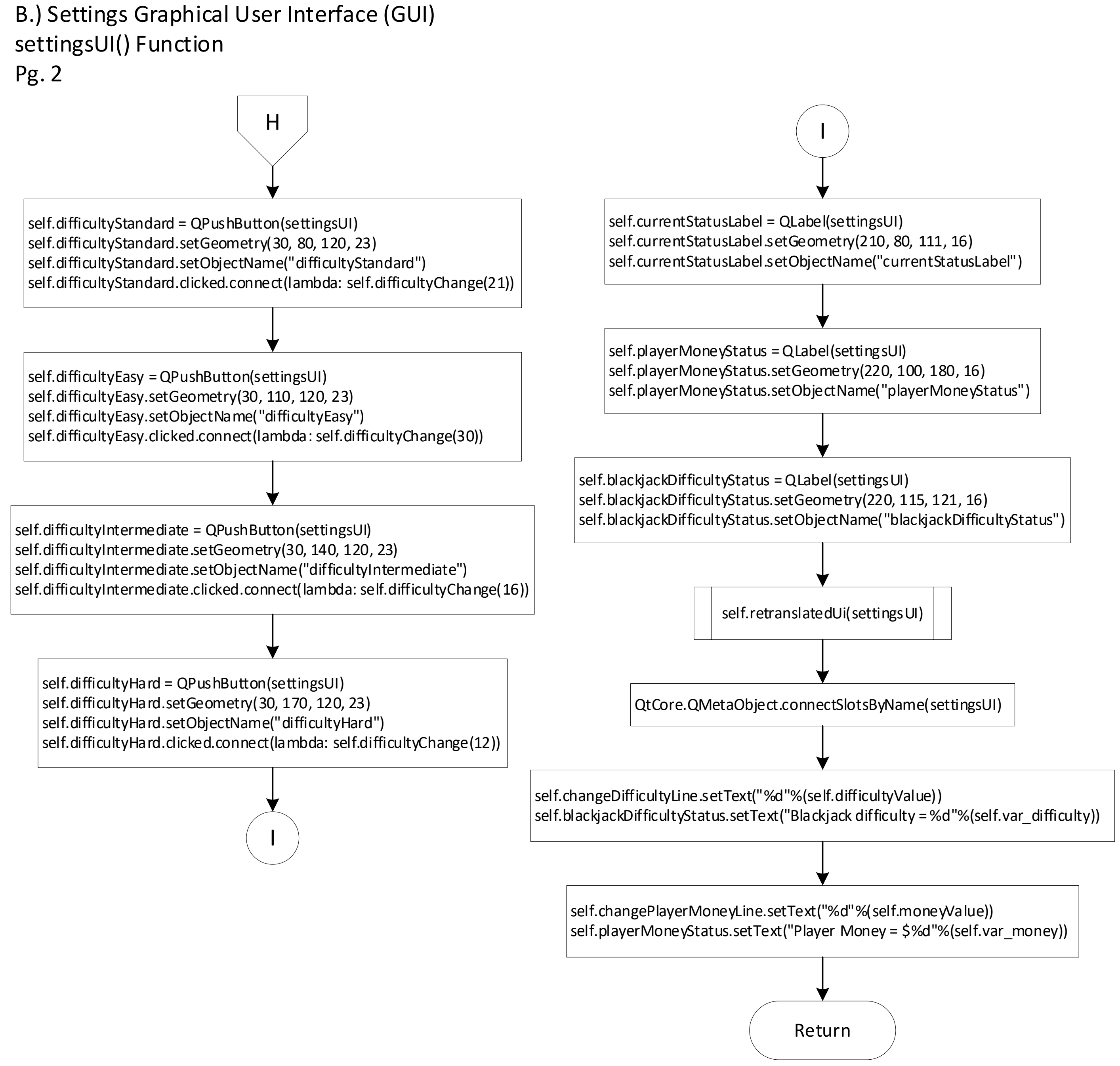
settingsUI() Function


B.) Settings Graphical User Interface (GUI)
settingsUI() Function
Pg. 1
settingsUI(self, settingsUI)
G
G
H

```
self.difficultyValue = self.var_difficultyself.btnB[0].resize(100, 50)
self.moneyValue = self.var_money

settingsUI.setObjectName("settingsUI")
settingsUI.resize(400, 200)
settingsUI.setFixedSize(settingsUI.width(), settingsUI.height())
settingsUI.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint, False)

self.changeMoneyLabel = QLabel(settingsUI)
self.changeMoneyLabel.setGeometry(20, 20, 111, 16)
self.changeMoneyLabel.setObjectName("changeMoneyLabel")

self.changeDifficultyLabel = QLabel(settingsUI)
self.changeDifficultyLabel.setGeometry(20, 50, 101, 16)
self.changeDifficultyLabel.setObjectName("changeDifficultyLabel")

self.changePlayerMoneyLine = QLineEdit(settingsUI)
self.changePlayerMoneyLine.setGeometry(140, 20, 113, 20)
self.changePlayerMoneyLine.setObjectName("changePlayerMoneyLine")

self.changeDifficultyLine = QLineEdit(settingsUI)
self.changeDifficultyLine.setGeometry(140, 50, 113, 20)
self.changeDifficultyLine.setObjectName("changeDifficultyLine")
self.changeDifficultyLine.setReadOnly(True)

self.applyButton = QPushButton(settingsUI)
self.applyButton.setGeometry(200, 160, 75, 23)
self.applyButton.setObjectName("applyButton")
self.applyButton.clicked.connect(lambda: self.settingsUpdate())

self.exitButton = QPushButton(settingsUI)
self.exitButton.setGeometry(290, 160, 75, 23)
self.exitButton.setObjectName("exitButton")
self.exitButton.clicked.connect(settingsUI.close)
```

Figure 8

settingsUI() Function



```
self.difficultyStandard = QPushButton(settingsUI)
self.difficultyStandard.setGeometry(30, 80, 120, 23)
self.difficultyStandard.setObjectName("difficultyStandard")
self.difficultyStandard.clicked.connect(lambda: self.difficultyChange(21))

self.difficultyEasy = QPushButton(settingsUI)
self.difficultyEasy.setGeometry(30, 110, 120, 23)
self.difficultyEasy.setObjectName("difficultyEasy")
self.difficultyEasy.clicked.connect(lambda: self.difficultyChange(30))

self.difficultyIntermediate = QPushButton(settingsUI)
self.difficultyIntermediate.setGeometry(30, 140, 120, 23)
self.difficultyIntermediate.setObjectName("difficultyIntermediate")
self.difficultyIntermediate.clicked.connect(lambda: self.difficultyChange(16))

self.difficultyHard = QPushButton(settingsUI)
self.difficultyHard.setGeometry(30, 170, 120, 23)
self.difficultyHard.setObjectName("difficultyHard")
self.difficultyHard.clicked.connect(lambda: self.difficultyChange(12))
```

```
self.currentStatusLabel = QLabel(settingsUI)
self.currentStatusLabel.setGeometry(210, 80, 111, 16)
self.currentStatusLabel.setObjectName("currentStatusLabel")

self.playerMoneyStatus = QLabel(settingsUI)
self.playerMoneyStatus.setGeometry(220, 100, 180, 16)
self.playerMoneyStatus.setObjectName("playerMoneyStatus")

self.blackjackDifficultyStatus = QLabel(settingsUI)
self.blackjackDifficultyStatus.setGeometry(220, 115, 121, 16)
self.blackjackDifficultyStatus.setObjectName("blackjackDifficultyStatus")

self.retranslatedUi(settingsUI)

QtCore.QMetaObject.connectSlotsByName(settingsUI)

self.changeDifficultyLine.setText("%d"%(self.difficultyValue))
self.blackjackDifficultyStatus.setText("Blackjack difficulty = %d"%(self.var_difficulty))

self.changePlayerMoneyLine.setText("%d"%(self.moneyValue))
self.playerMoneyStatus.setText("Player Money = $%d"%(self.var_money))
```

Figure 9

difficultyChange() & settingsUpdate() Function

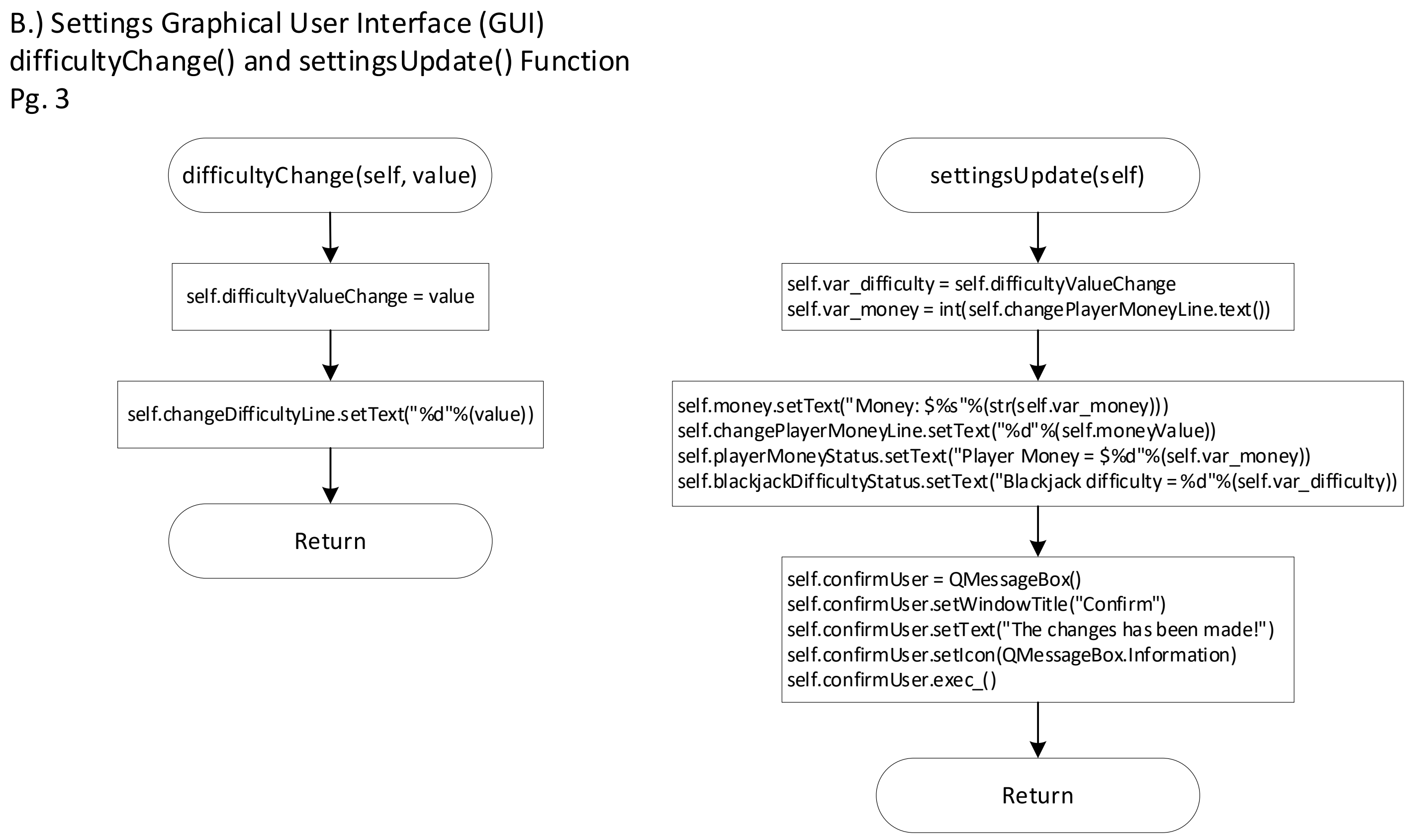
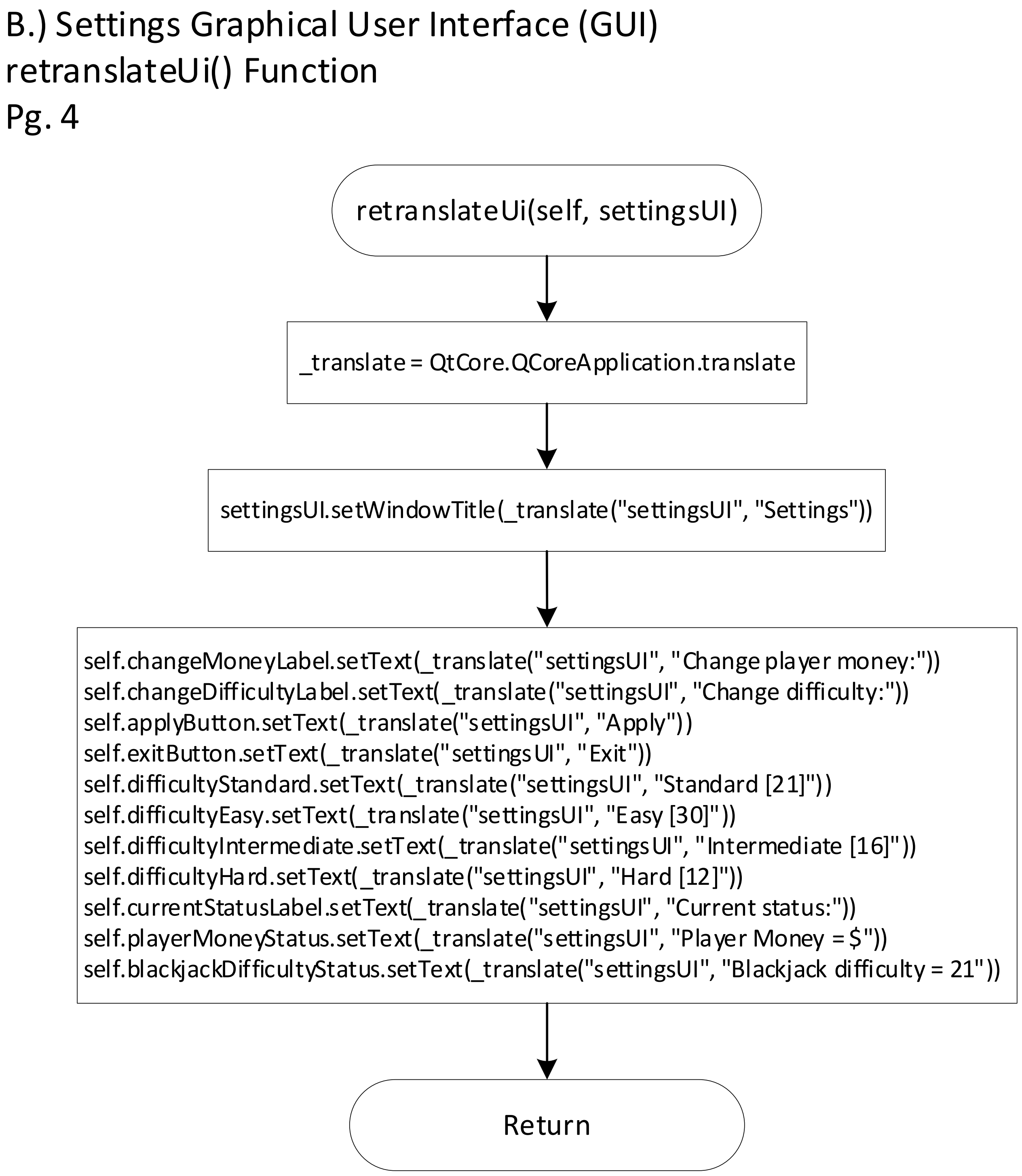

B.) Settings Graphical User Interface (GUI)
difficultyChange() and settingsUpdate() Function
Pg. 3
difficultyChange(self, value)
self.difficultyValueChange = value
self.changeDifficultyLine.setText("%d"%(value))
Return
settingsUpdate(self)
self.var_difficulty = self.difficultyValueChange
self.var_money = int(self.changePlayerMoneyLine.text())
self.money.setText("Money: $%s"%(str(self.var_money)))
self.changePlayerMoneyLine.setText("%d"%(self.moneyValue))
self.playerMoneyStatus.setText("Player Money = $%d"%(self.var_money))
self.blackjackDifficultyStatus.setText("Blackjack difficulty = %d"%(self.var_difficulty))
self.confirmUser = QMessageBox()
self.confirmUser.setWindowTitle("Confirm")
self.confirmUser.setText("The changes has been made!")
self.confirmUser.setIcon(QMessageBox.Information)
self.confirmUser.exec_()
Return

Figure 10

retranslateUI() Function


B.) Settings Graphical User Interface (GUI)
retranslateUi() Function
Pg. 4
retranslateUi(self, settingsUI)
Return

```
_translate = QtCore.QCoreApplication.translate
```

```
settingsUI.setWindowTitle(_translate("settingsUI", "Settings"))
```

```
self.changeMoneyLabel.setText(_translate("settingsUI", "Change player money:"))
self.changeDifficultyLabel.setText(_translate("settingsUI", "Change difficulty:"))
self.applyButton.setText(_translate("settingsUI", "Apply"))
self.exitButton.setText(_translate("settingsUI", "Exit"))
self.difficultyStandard.setText(_translate("settingsUI", "Standard [21]"))
self.difficultyEasy.setText(_translate("settingsUI", "Easy [30]"))
self.difficultyIntermediate.setText(_translate("settingsUI", "Intermediate [16]"))
self.difficultyHard.setText(_translate("settingsUI", "Hard [12]"))
self.currentStatusLabel.setText(_translate("settingsUI", "Current status:"))
self.playerMoneyStatus.setText(_translate("settingsUI", "Player Money = $"))
self.blackjackDifficultyStatus.setText(_translate("settingsUI", "Blackjack difficulty = 21"))
```

Figure 11

play() & empty() Function

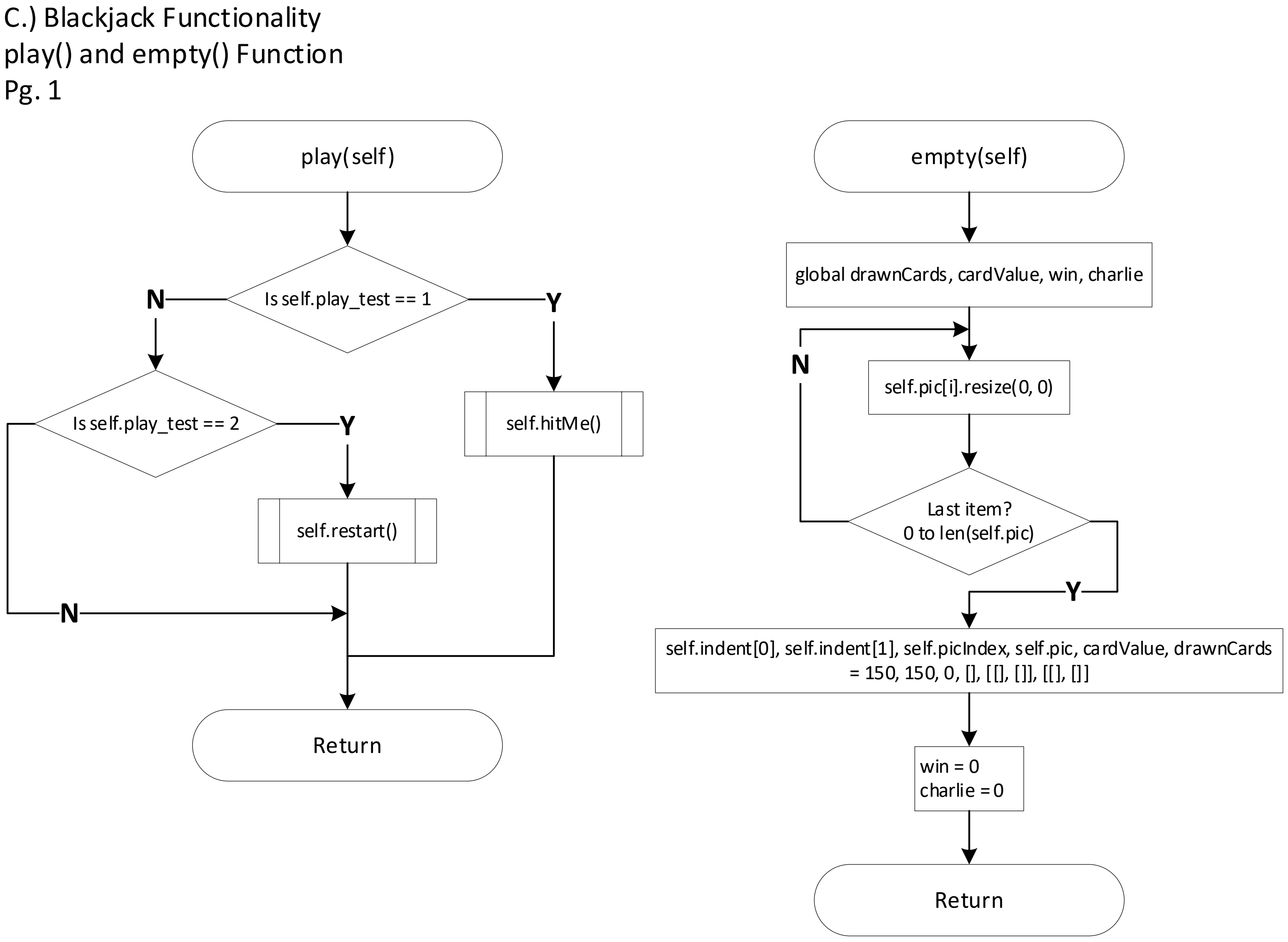

C.) Blackjack Functionality
play() and empty() Function
Pg. 1
play(self)
Is self.play_test == 1
N
Y
Is self.play_test == 2
Y
N
self.hitMe()
self.restart()
Return
empty(self)
global drawnCards, cardValue, win, charlie
self.pic[i].resize(0, 0)
Last item?
0 to len(self.pic)
N
Y
self.indent[0], self.indent[1], self.picIndex, self.pic, cardValue, drawnCards = 150, 150, 0, [], [[], []], [[], []]
win = 0
charlie = 0
Return

Figure 12

resize() & bet() Function

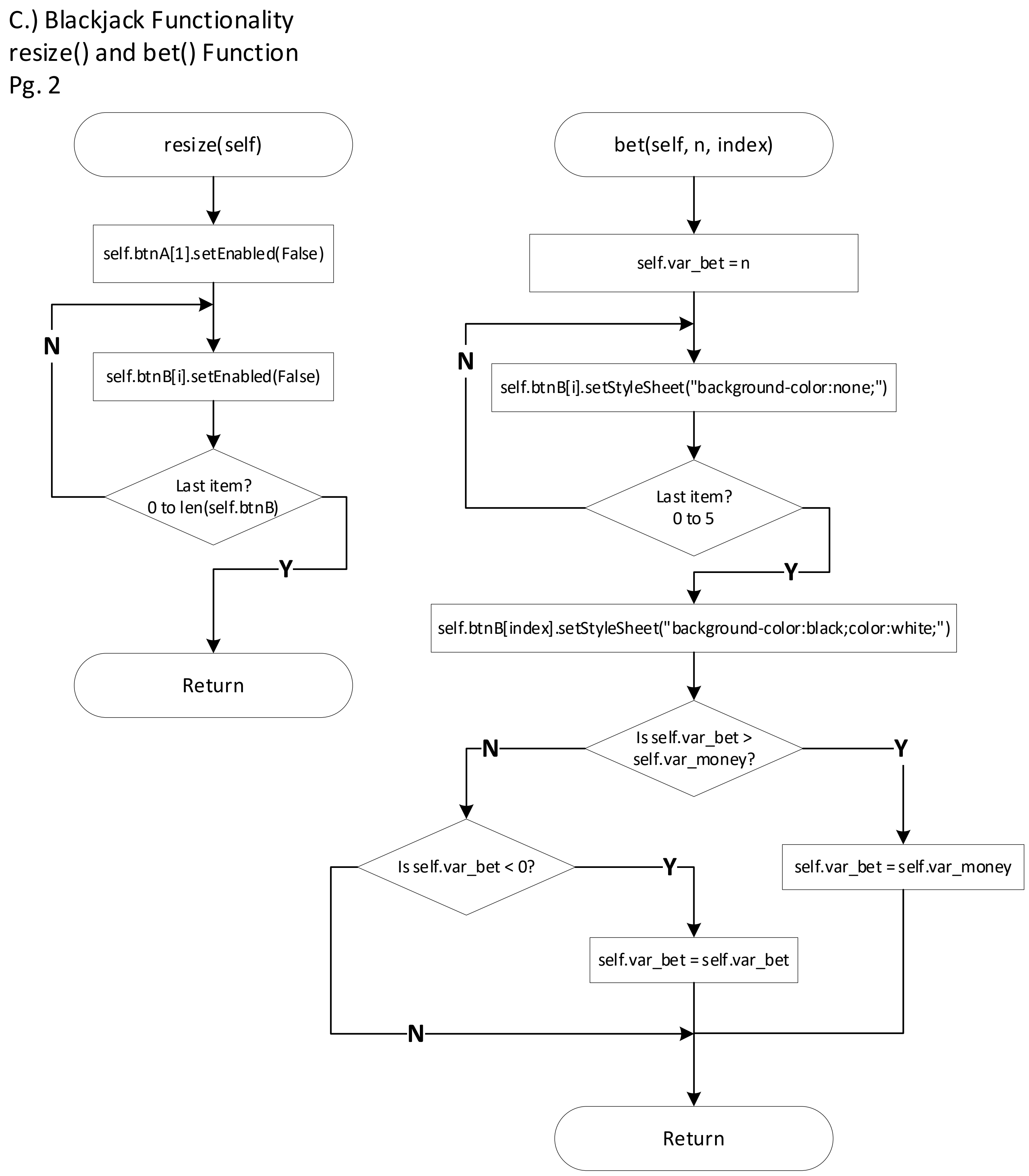

C.) Blackjack Functionality
resize() and bet() Function
Pg. 2
resize(self)
self.btnA[1].setEnabled(False)
N
self.btnB[i].setEnabled(False)
Last item?
0 to len(self.btnB)
Y
Return
bet(self, n, index)
self.var_bet = n
N
self.btnB[i].setStyleSheet("background-color:none;")
Last item?
0 to 5
Y
self.btnB[index].setStyleSheet("background-color:black;color:white;")
Is self.var_bet >
self.var_money?
N
Y
Is self.var_bet < 0?
Y
self.var_bet = self.var_money
self.var_bet = self.var_bet
N
Return

Figure 13

hitMe() Function

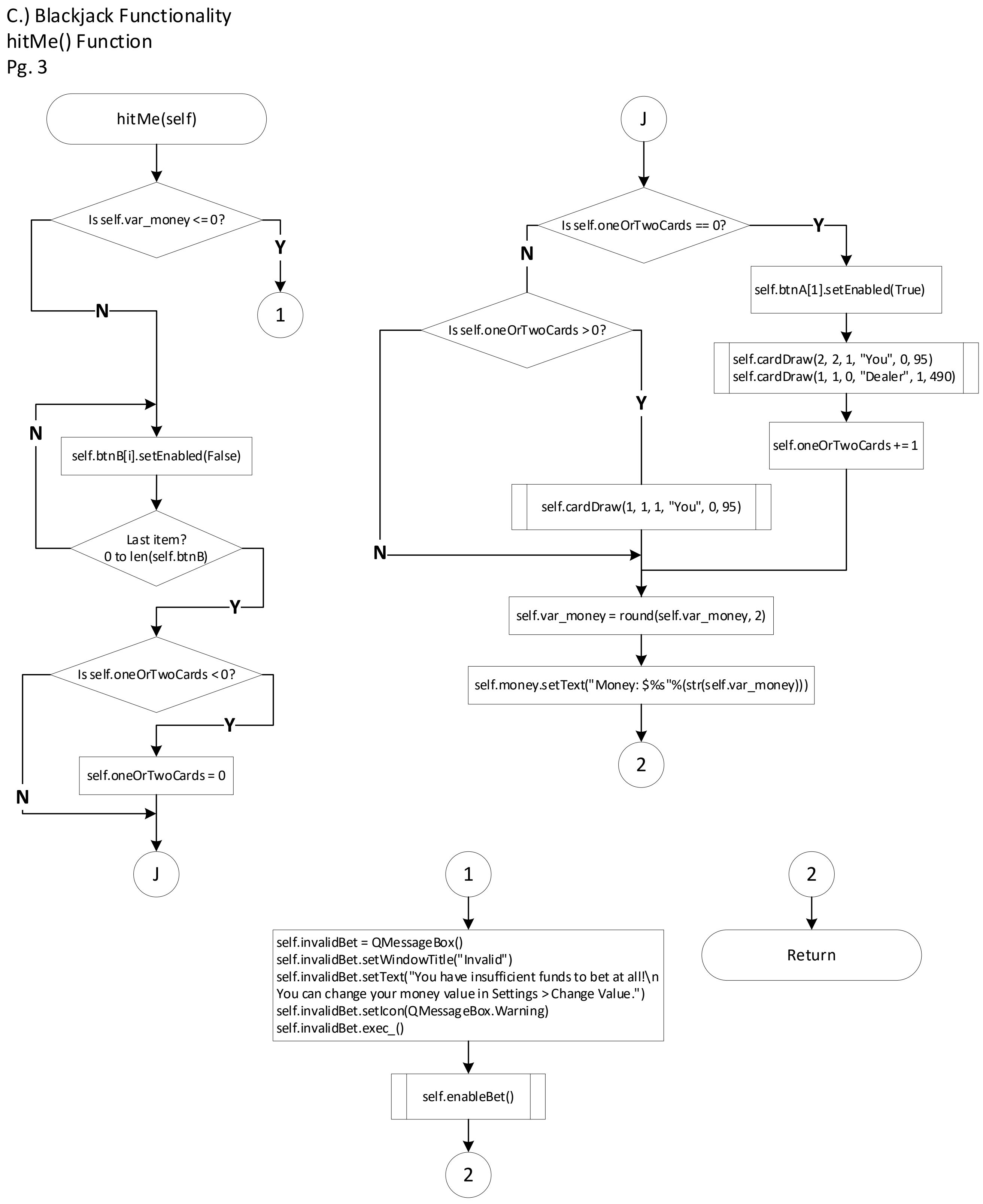

C.) Blackjack Functionality
hitMe() Function
Pg. 3
hitMe(self)
Is self.var_money <= 0?
Y
1
N
N
self.btnB[i].setEnabled(False)
Last item?
0 to len(self.btnB)
Y
Is self.oneOrTwoCards < 0?
Y
self.oneOrTwoCards = 0
N
J
J
Is self.oneOrTwoCards == 0?
Y
N
self.btnA[1].setEnabled(True)
self.cardDraw(2, 2, 1, "You", 0, 95)
self.cardDraw(1, 1, 0, "Dealer", 1, 490)
self.oneOrTwoCards += 1
Is self.oneOrTwoCards > 0?
Y
self.cardDraw(1, 1, 1, "You", 0, 95)
N
self.var_money = round(self.var_money, 2)
self.money.setText("Money: $%s"%(str(self.var_money)))
2
1
self.invalidBet = QMessageBox()
self.invalidBet.setWindowTitle("Invalid")
self.invalidBet.setText("You have insufficient funds to bet at all!\n You can change your money value in Settings > Change Value.")
self.invalidBet.setIcon(QMessageBox.Warning)
self.invalidBet.exec_()
self.enableBet()
2
2
Return

Figure 14

cardDraw() Function

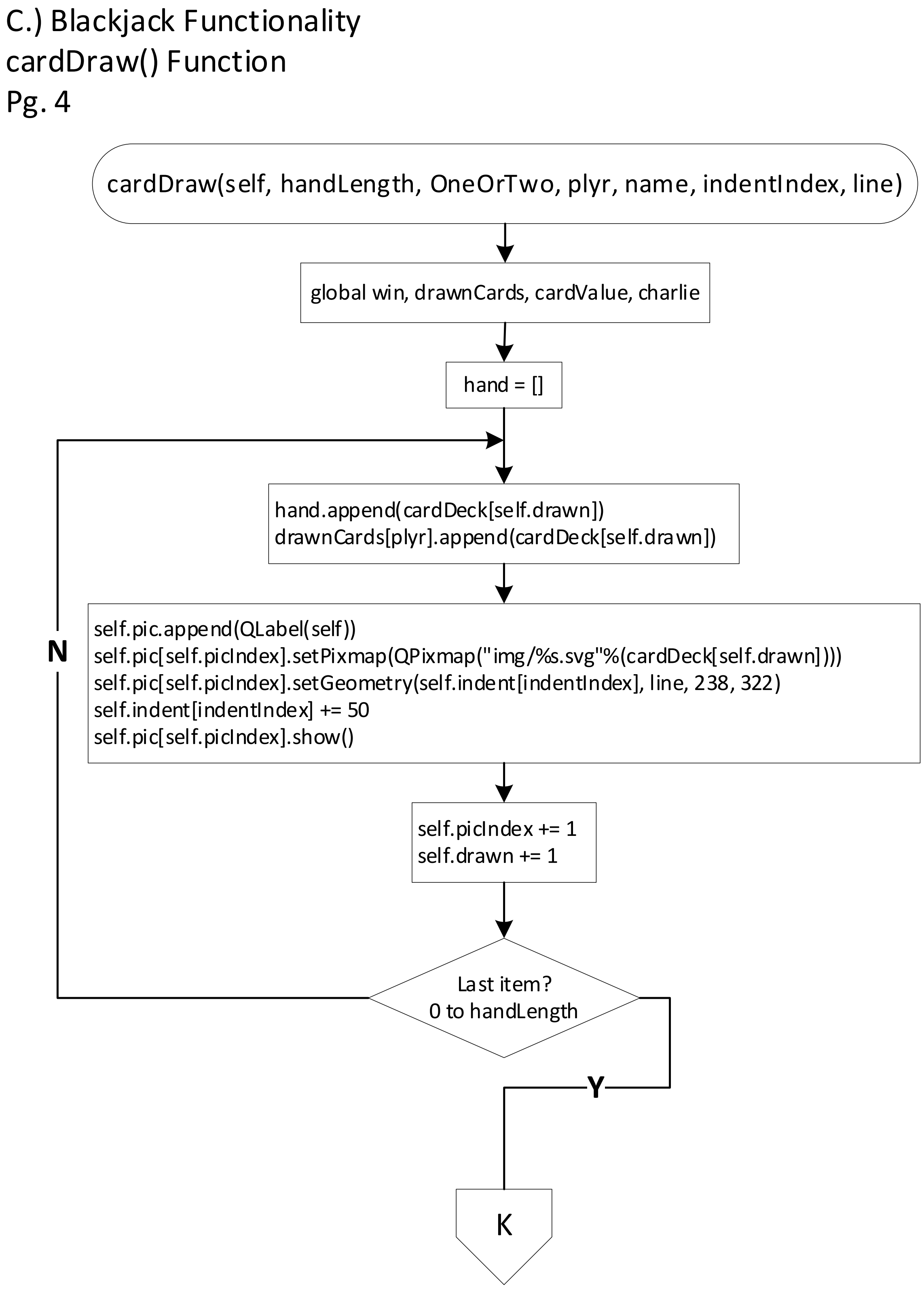

C.) Blackjack Functionality
cardDraw() Function
Pg. 4
cardDraw(self, handLength, OneOrTwo, plyr, name, indentIndex, line)
global win, drawnCards, cardValue, charlie
hand = []
hand.append(cardDeck[self.drawn])
drawnCards[plyr].append(cardDeck[self.drawn])
self.pic.append(QLabel(self))
self.pic[self.picIndex].setPixmap(QPixmap("img/%s.svg"%(cardDeck[self.drawn])))
self.pic[self.picIndex].setGeometry(self.indent[indentIndex], line, 238, 322)
self.indent[indentIndex] += 50
self.pic[self.picIndex].show()
self.picIndex += 1
self.drawn += 1
Last item?
0 to handLength
N
Y
K

Figure 15

cardDraw() Function

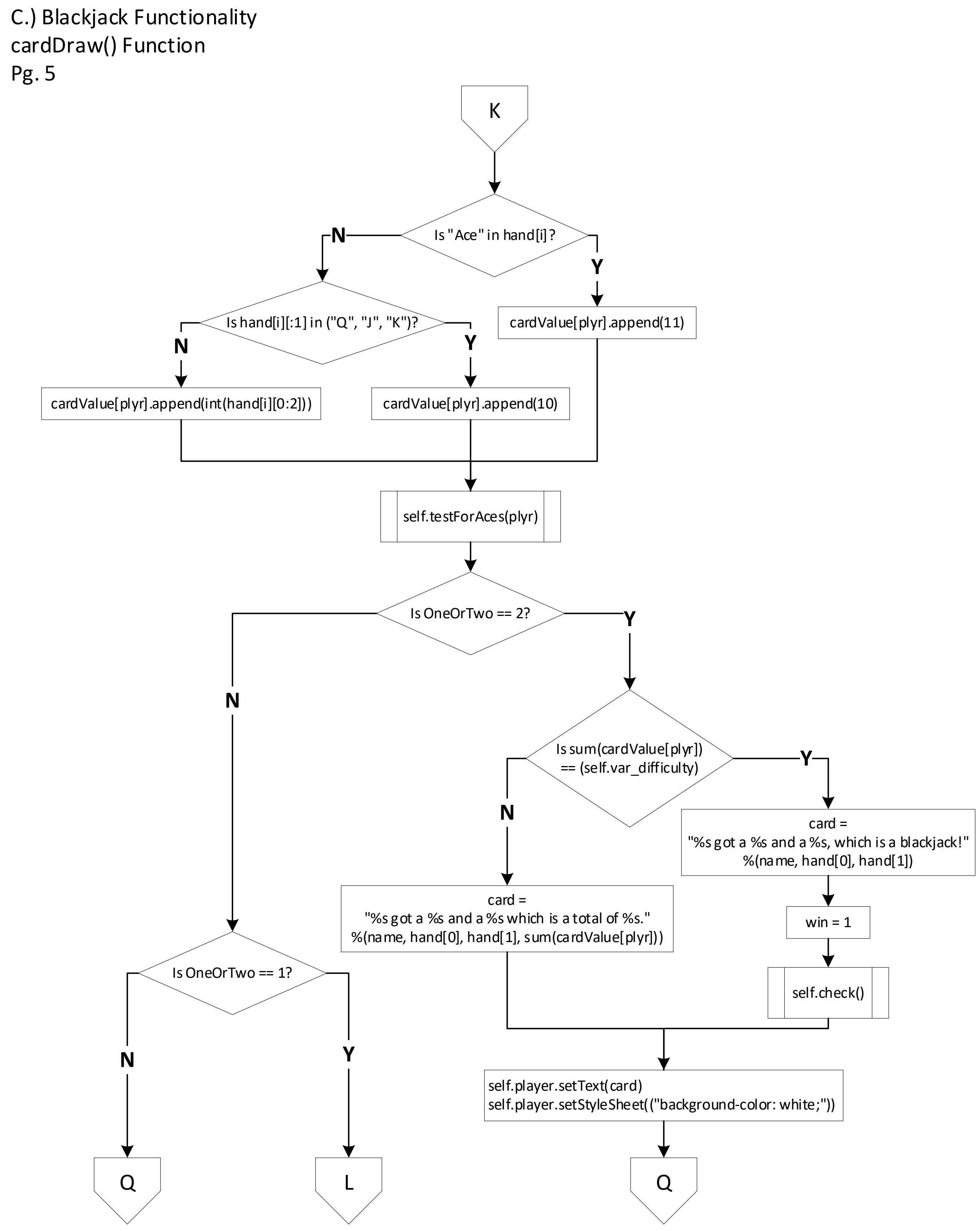

C.) Blackjack Functionality
cardDraw() Function
Pg. 5
K
Is "Ace" in hand[i]?
N
Y
Is hand[i][:1] in ("Q", "J", "K")?
N
Y
cardValue[plyr].append(11)
cardValue[plyr].append(int(hand[i][0:2]))
cardValue[plyr].append(10)
self.testForAces(plyr)
Is OneOrTwo == 2?
Y
N
Is sum(cardValue[plyr]) == (self.var_difficulty)
N
Y
card = "%s got a %s and a %s, which is a blackjack!" %(name, hand[0], hand[1])
card = "%s got a %s and a %s which is a total of %s." %(name, hand[0], hand[1], sum(cardValue[plyr]))
win = 1
self.check()
Is OneOrTwo == 1?
N
Y
self.player.setText(card)
self.player.setStyleSheet(("background-color: white;"))
Q
L
Q

Figure 16

cardDraw() Function

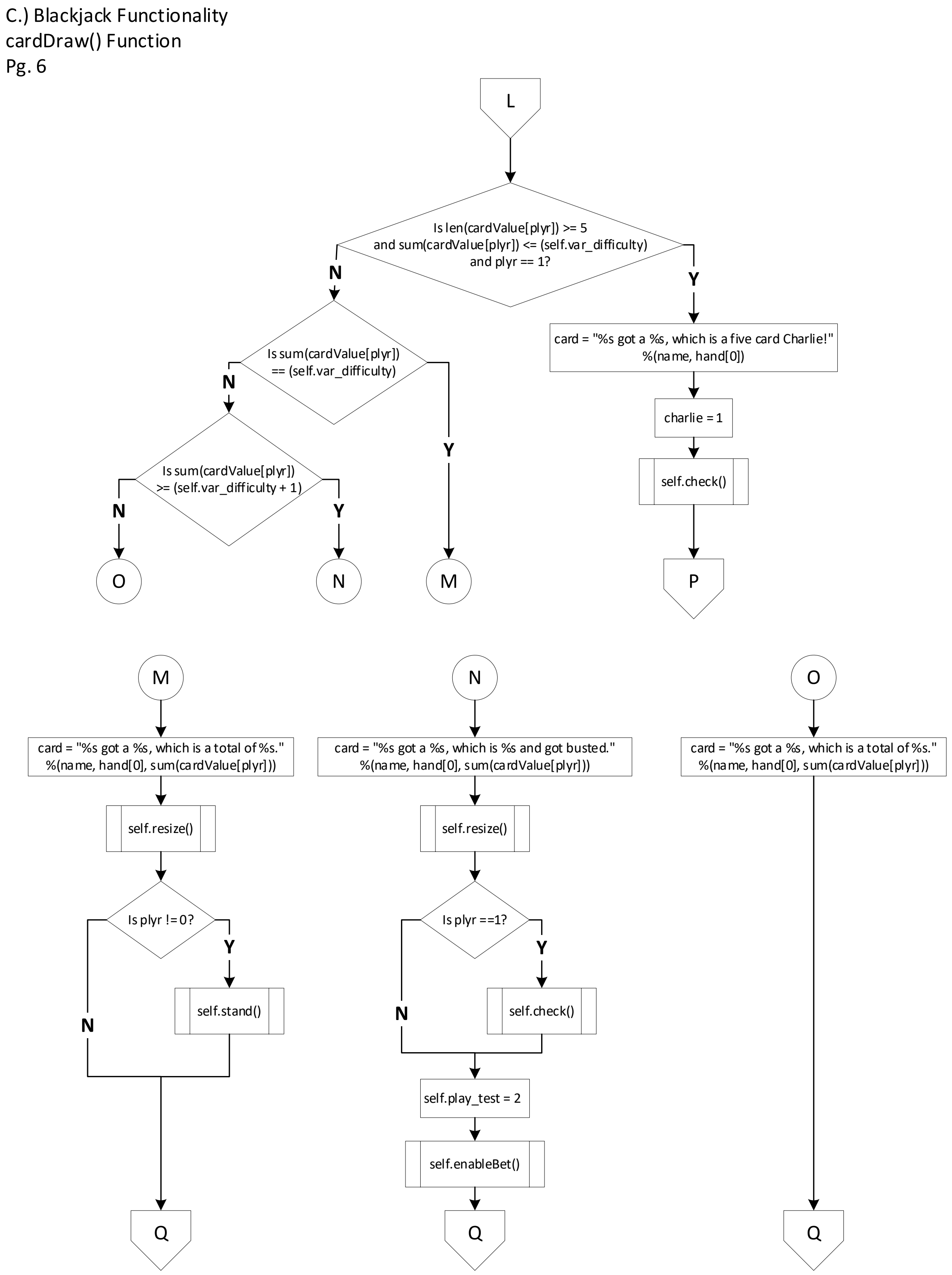

C.) Blackjack Functionality
cardDraw() Function
Pg. 6
L
Is len(cardValue[plyr]) >= 5 and sum(cardValue[plyr]) <= (self.var_difficulty) and plyr == 1?
N
Y
card = "%s got a %s, which is a five card Charlie!" %(name, hand[0])
charlie = 1
self.check()
P
Is sum(cardValue[plyr]) == (self.var_difficulty)
N
Y
Is sum(cardValue[plyr]) >= (self.var_difficulty + 1)
N
Y
O
N
M
M
card = "%s got a %s, which is a total of %s." %(name, hand[0], sum(cardValue[plyr]))
self.resize()
Is plyr != 0?
Y
N
self.stand()
Q
N
card = "%s got a %s, which is %s and got busted." %(name, hand[0], sum(cardValue[plyr]))
self.resize()
Is plyr ==1?
Y
N
self.check()
self.play_test = 2
self.enableBet()
Q
O
card = "%s got a %s, which is a total of %s." %(name, hand[0], sum(cardValue[plyr]))
Q

Figure 17

cardDraw() Function

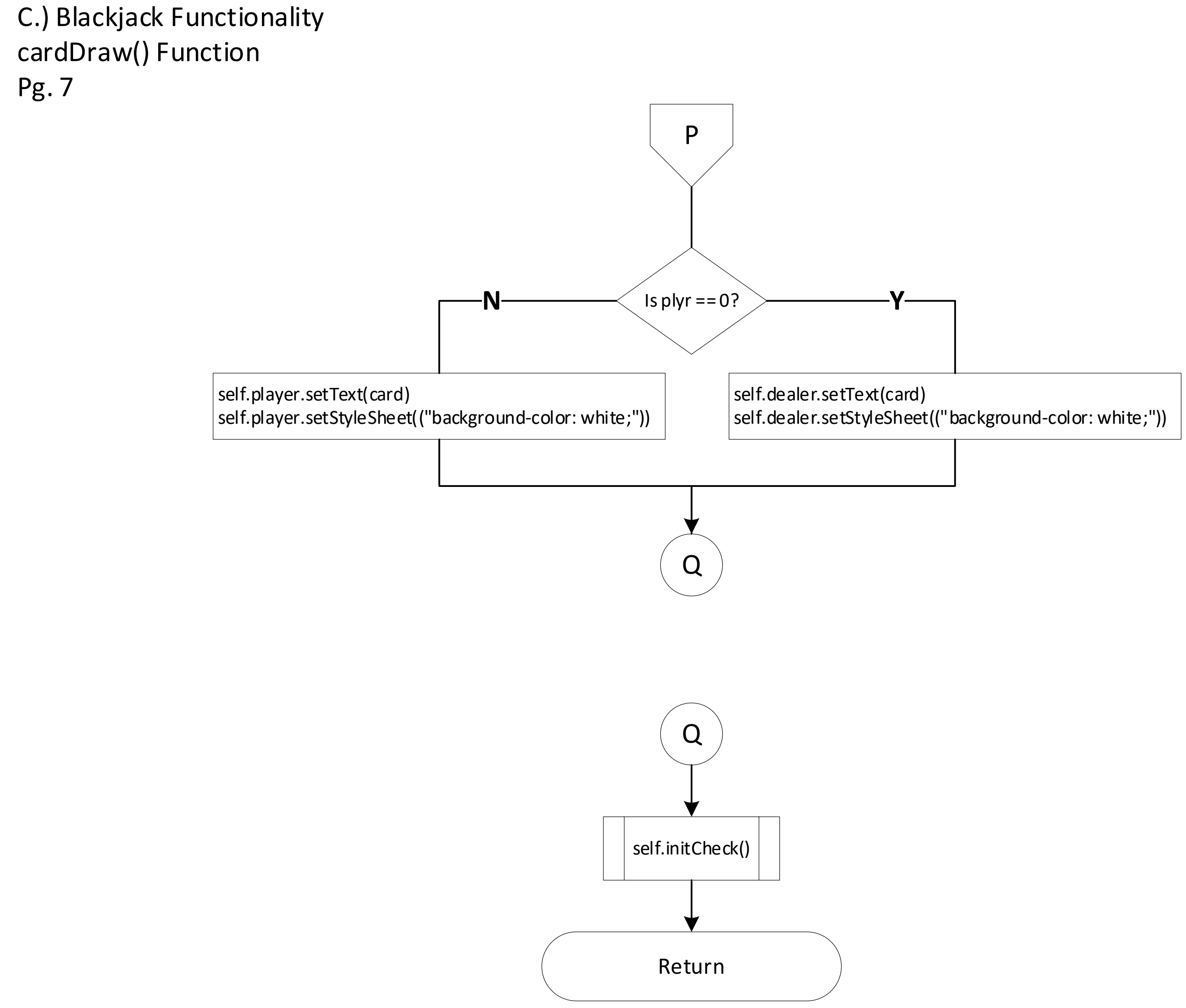

C.) Blackjack Functionality
cardDraw() Function
Pg. 7
P
Is plyr == 0?
N
Y
self.player.setText(card)
self.player.setStyleSheet(("background-color: white;"))
self.dealer.setText(card)
self.dealer.setStyleSheet(("background-color: white;"))
Q
Q
self.initCheck()
Return

Figure 18

testForAces() Function

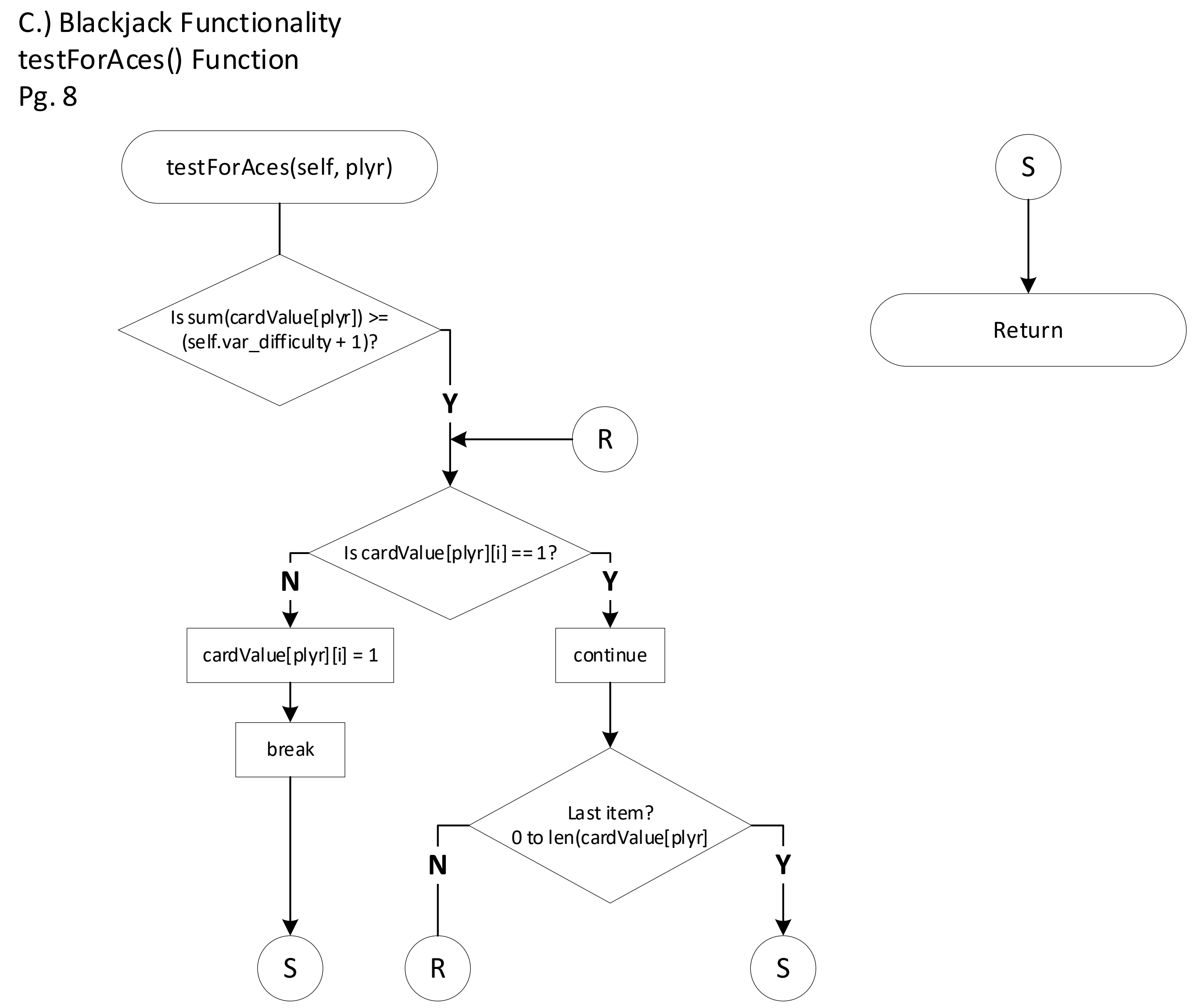

C.) Blackjack Functionality
testForAces() Function
Pg. 8
testForAces(self, plyr)
Is sum(cardValue[plyr]) >= (self.var_difficulty + 1)?
Y
R
Is cardValue[plyr][i] == 1?
N
Y
cardValue[plyr][i] = 1
continue
break
Last item? 0 to len(cardValue[plyr]
N
Y
S
R
S
S
Return

Figure 19

check() Function

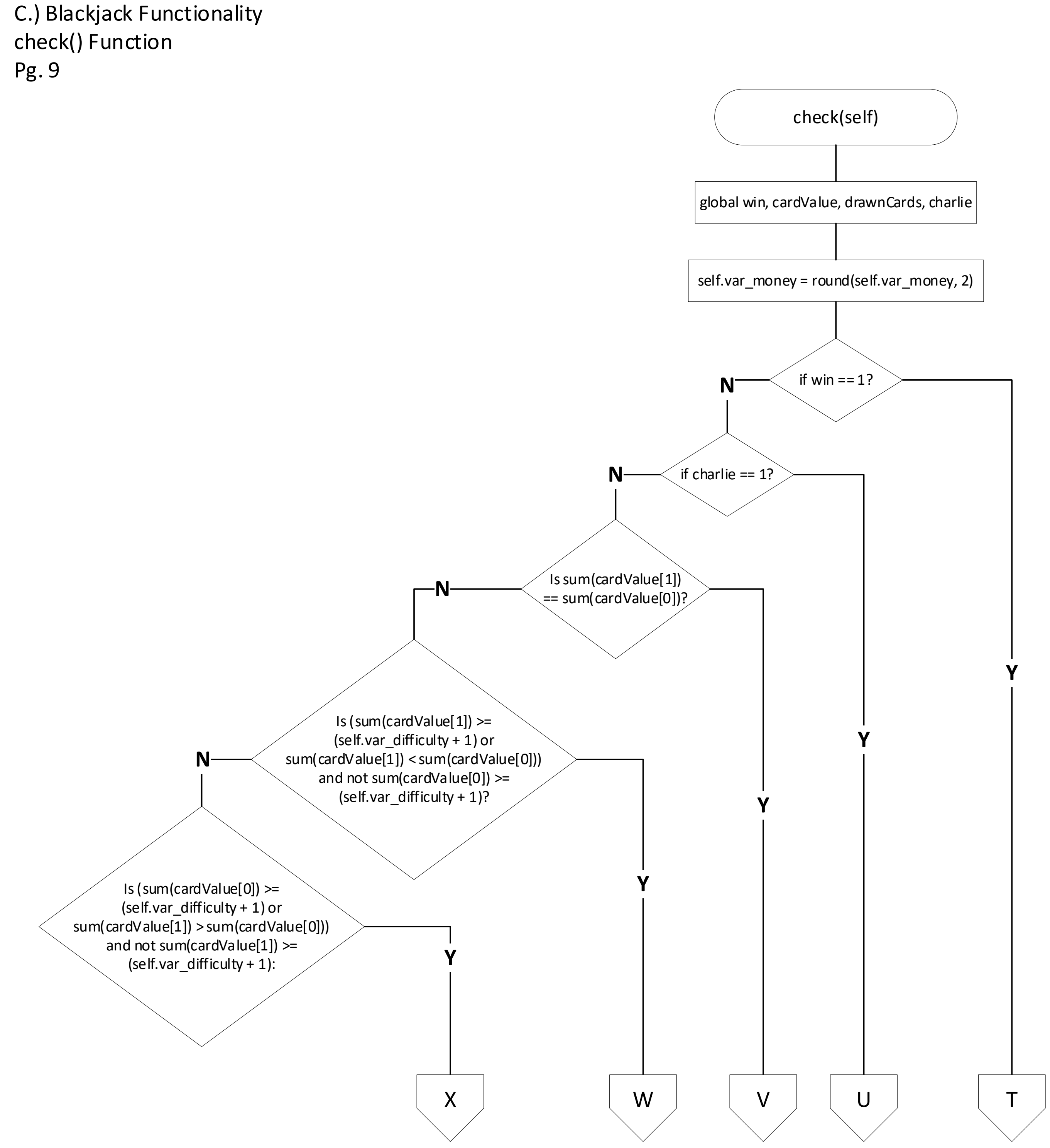

C.) Blackjack Functionality
check() Function
Pg. 9
check(self)
global win, cardValue, drawnCards, charlie
self.var_money = round(self.var_money, 2)
if win == 1?
if charlie == 1?
Is sum(cardValue[1]) == sum(cardValue[0])?
Is (sum(cardValue[1]) >= (self.var_difficulty + 1) or sum(cardValue[1]) < sum(cardValue[0])) and not sum(cardValue[0]) >= (self.var_difficulty + 1)?
Is (sum(cardValue[0]) >= (self.var_difficulty + 1) or sum(cardValue[1]) > sum(cardValue[0])) and not sum(cardValue[1]) >= (self.var_difficulty + 1):
N
N
N
N
Y
Y
Y
Y
Y
X
W
V
U
T

Figure 20

check() Function

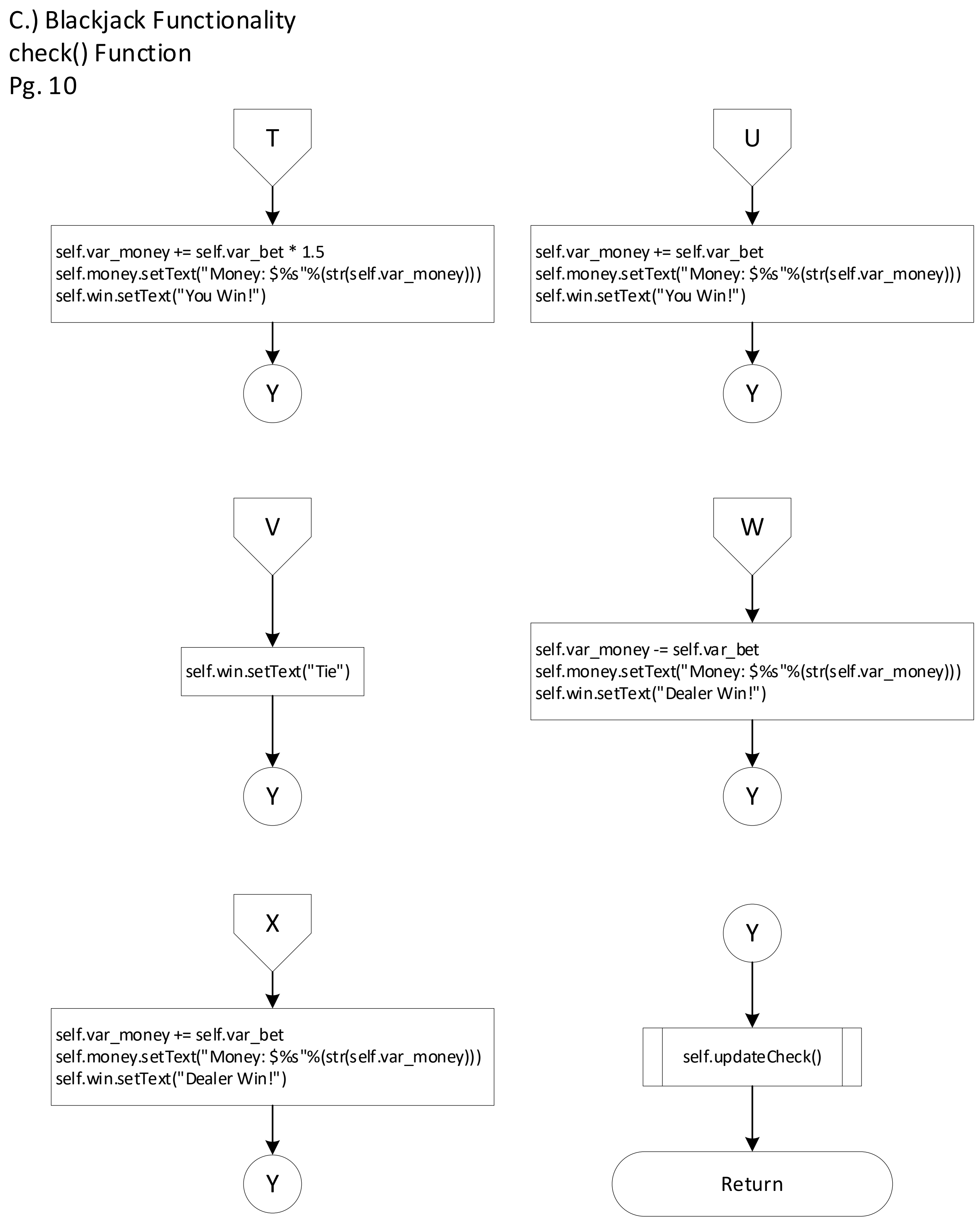

C.) Blackjack Functionality
check() Function
Pg. 10
T
self.var_money += self.var_bet * 1.5
self.money.setText("Money: $%s"%(str(self.var_money)))
self.win.setText("You Win!")
Y
U
self.var_money += self.var_bet
self.money.setText("Money: $%s"%(str(self.var_money)))
self.win.setText("You Win!")
Y
V
self.win.setText("Tie")
Y
W
self.var_money -= self.var_bet
self.money.setText("Money: $%s"%(str(self.var_money)))
self.win.setText("Dealer Win!")
Y
X
self.var_money += self.var_bet
self.money.setText("Money: $%s"%(str(self.var_money)))
self.win.setText("Dealer Win!")
Y
Y
self.updateCheck()
Return

Figure 21

intCheck() & updateCheck() Function

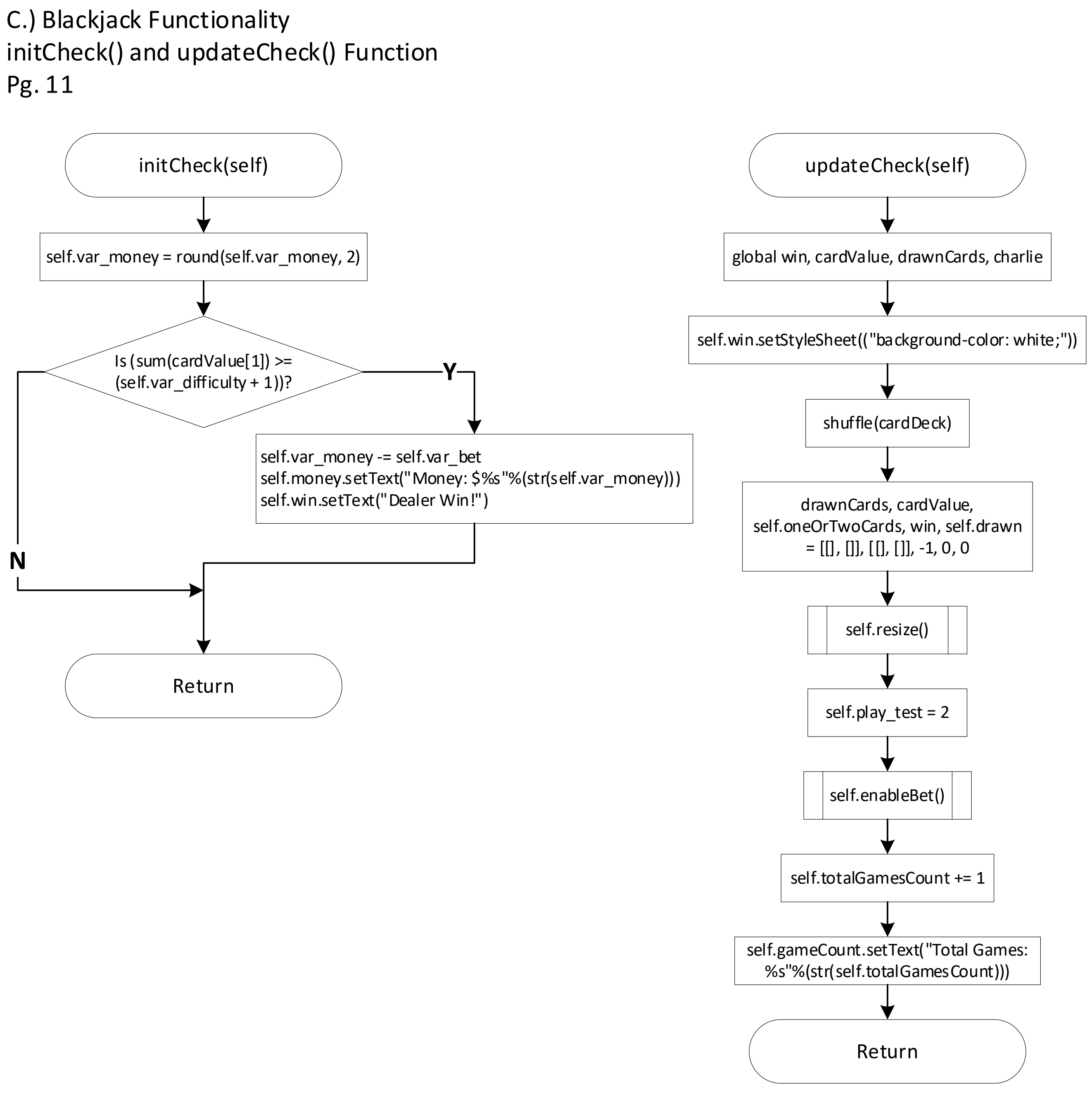

C.) Blackjack Functionality
initCheck() and updateCheck() Function
Pg. 11
initCheck(self)
self.var_money = round(self.var_money, 2)
Is (sum(cardValue[1]) >= (self.var_difficulty + 1))?
Y
N
self.var_money -= self.var_bet
self.money.setText("Money: $%s"%(str(self.var_money)))
self.win.setText("Dealer Win!")
Return
updateCheck(self)
global win, cardValue, drawnCards, charlie
self.win.setStyleSheet(("background-color: white;"))
shuffle(cardDeck)
drawnCards, cardValue, self.oneOrTwoCards, win, self.drawn = [[], []], [[], []], -1, 0, 0
self.resize()
self.play_test = 2
self.enableBet()
self.totalGamesCount += 1
self.gameCount.setText("Total Games: %s"%(str(self.totalGamesCount)))
Return

Figure 22

enableBet() & restart() Function

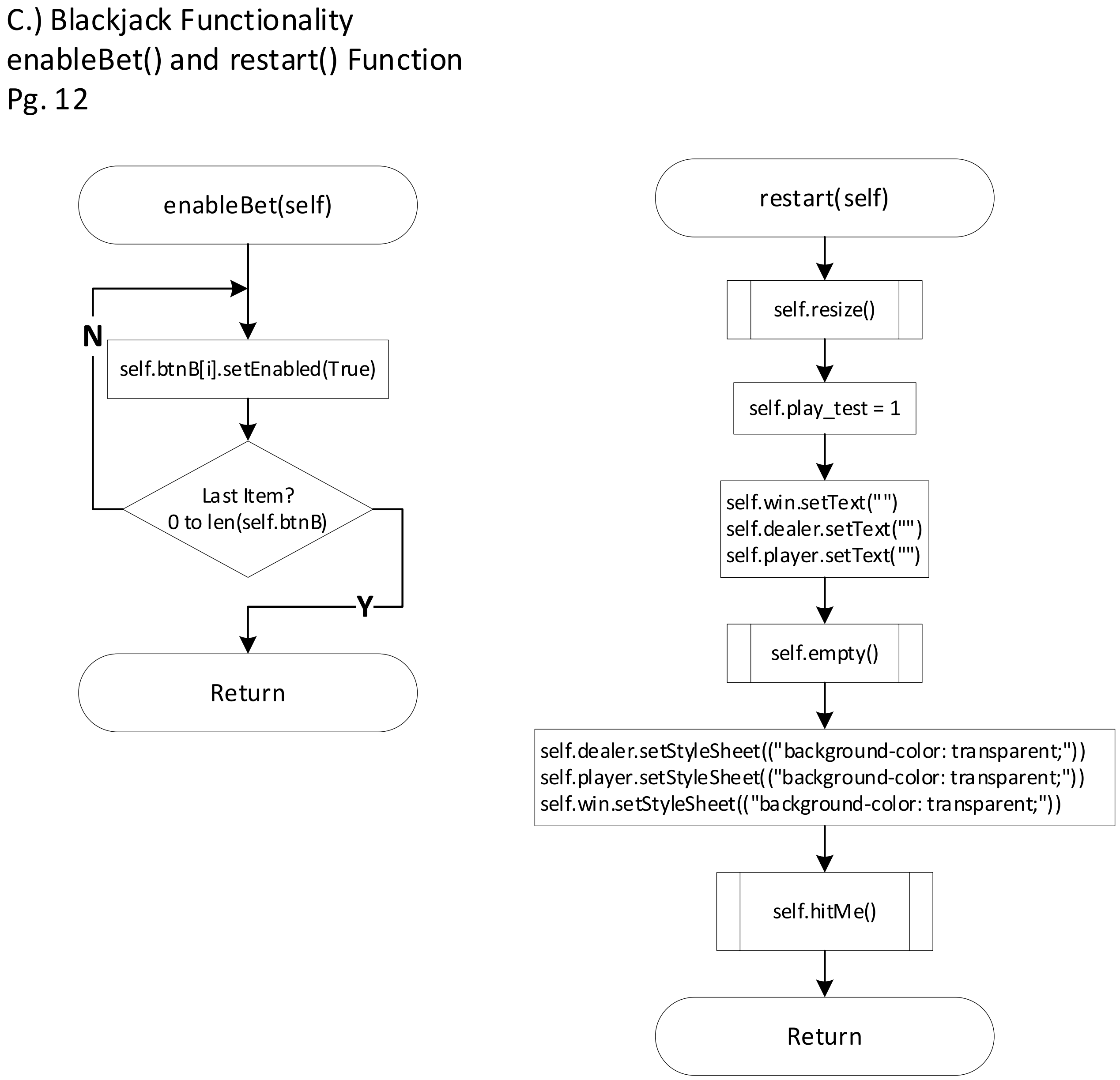

C.) Blackjack Functionality
enableBet() and restart() Function
Pg. 12
enableBet(self)
N
self.btnB[i].setEnabled(True)
Last Item?
0 to len(self.btnB)
Y
Return
restart(self)
self.resize()
self.play_test = 1
self.win.setText("")
self.dealer.setText("")
self.player.setText("")
self.empty()
self.dealer.setStyleSheet(("background-color: transparent;"))
self.player.setStyleSheet(("background-color: transparent;"))
self.win.setStyleSheet(("background-color: transparent;"))
self.hitMe()
Return

Figure 23

reset() Function

C.) Blackjack Functionality
reset() Function
Pg. 13

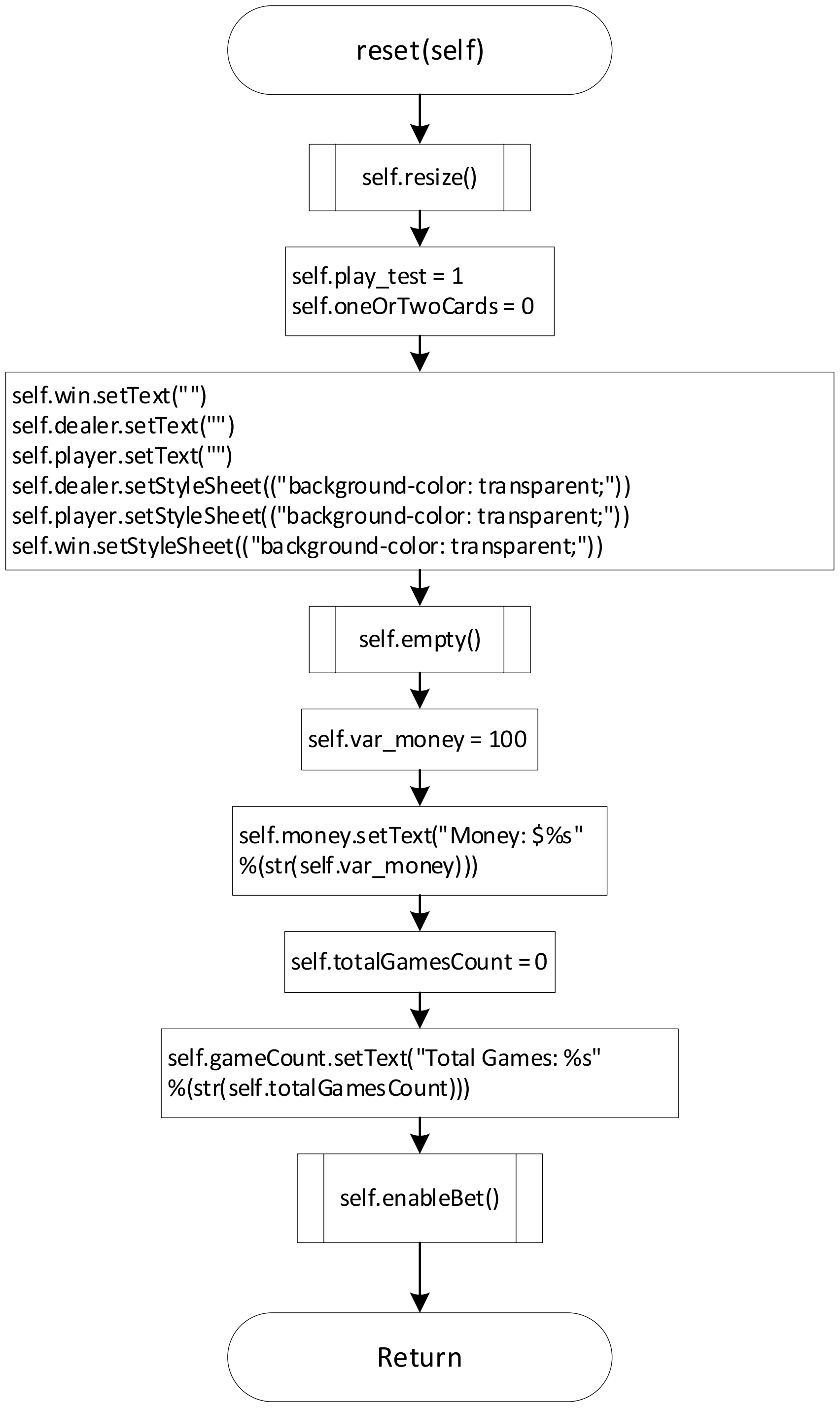
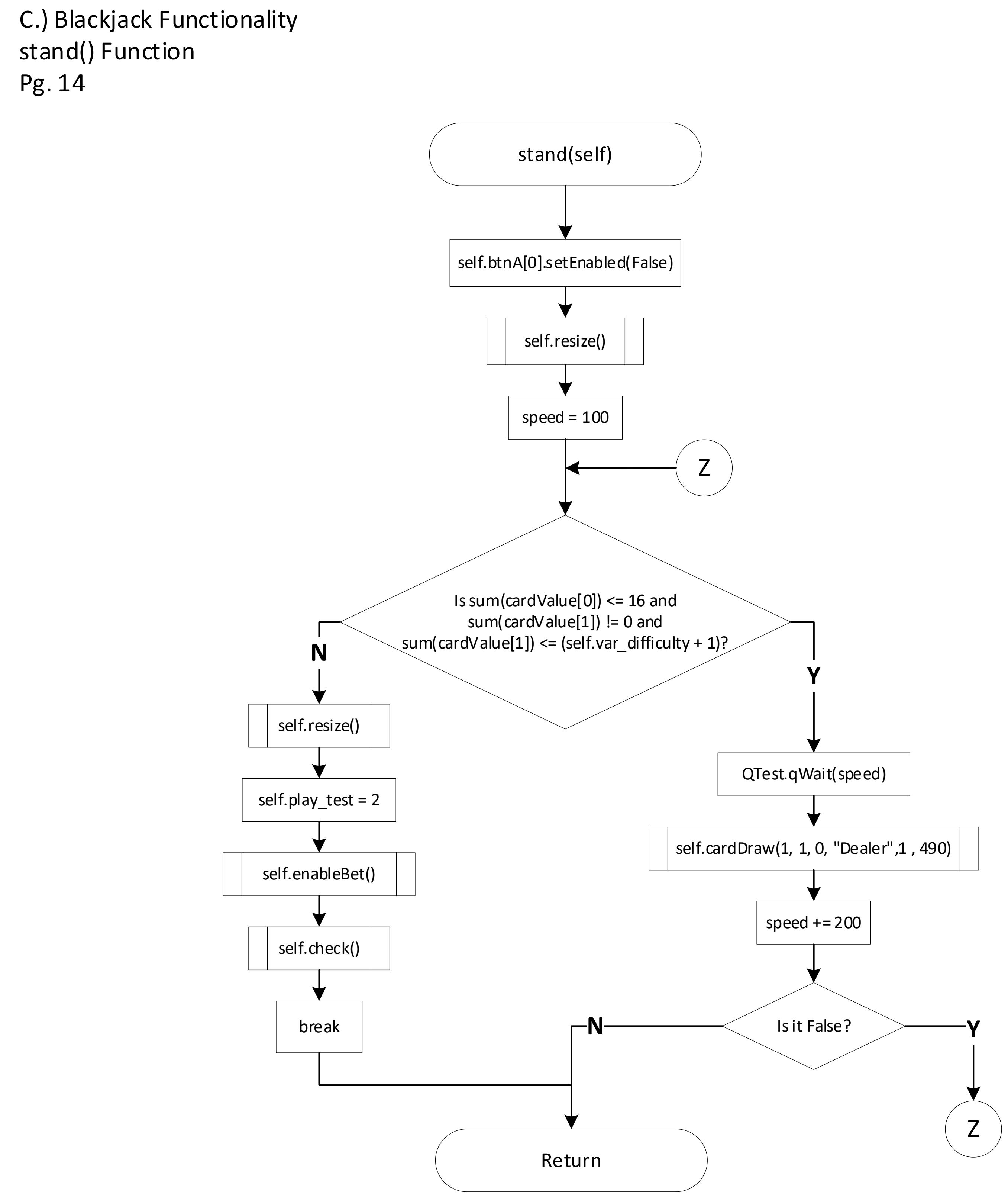

reset(self)
self.resize()
self.play_test = 1
self.oneOrTwoCards = 0
self.win.setText("")
self.dealer.setText("")
self.player.setText("")
self.dealer.setStyleSheet(("background-color: transparent;"))
self.player.setStyleSheet(("background-color: transparent;"))
self.win.setStyleSheet(("background-color: transparent;"))
self.empty()
self.var_money = 100
self.money.setText("Money: $%s" %(str(self.var_money)))
self.totalGamesCount = 0
self.gameCount.setText("Total Games: %s" %(str(self.totalGamesCount)))
self.enableBet()
Return

Figure 24

stand() Function


C.) Blackjack Functionality
stand() Function
Pg. 14
stand(self)
self.btnA[0].setEnabled(False)
self.resize()
speed = 100
Z
Is sum(cardValue[0]) <= 16 and sum(cardValue[1]) != 0 and sum(cardValue[1]) <= (self.var_difficulty + 1)?
N
Y
self.resize()
self.play_test = 2
self.enableBet()
self.check()
break
QTest.qWait(speed)
self.cardDraw(1, 1, 0, "Dealer",1 , 490)
speed += 200
Is it False?
N
Y
Z
Return

D. Algorithm

1. The game starts by shuffling the deck and dealing cards by giving two (2) cards to the player, and one (1) card to the dealer.
2. If player changes the difficulty of the game:
    a. If difficulty is “Easy” Then, Set threshold value is 30.
    b. Else If difficulty is “Standard” Then, Set threshold value is 21.
    c. Else If difficulty is “Intermediate” Then, Set threshold value is 16.
    d. Else If difficulty is “Hard” Then, Set threshold value is 12.
3. Set Bet value to 1$.
4. If player changes Bet value:
    a. If “Bet 1$” Then, Set bet value to 1$.
    b. Else If “Bet 5$” Then, Set bet value to 5$.
    c. Else If “Bet 10$” Then, Set bet value to 10$.
    d. Else If “Bet 20$” Then, Set bet value to 20$.
    e. Else If “Bet $” Then, Set bet value to player input.
5. Show the player both their cards and the dealer.
6. Ask the player if they want to hit or stand.
7. If the player chooses to hit, deal them another card and add its value to their total.
8. If the player's total exceeds 21, they lose the game and bet value is deducted to player’s money.
9. If the player chooses to stand, the dealer's will draw and show their second card.
10. If the dealer's total exceeds 21, the player wins and bet value is added to player’s money.
11. If the dealer's total is greater than the player's total and less than or equal to 21, the player loses the game and bet value is deducted to player’s money.
12. If the player's total is greater than the dealer's total and less than or equal to 21, the player wins and bet value is added to player’s money.
13. If both the player and dealer have the same total, it's a tie.
14. Ask the player if they want to play again by hitting again. If the player doees, go back to step 1. If not, the player can end the game by closing the application.

E. Pseudocode

Note: This pseudocode only covers "Section C: Blackjack Functions", some changes were made that may deviate from the main code.

```
from random import shuffle



Declare Integer drawnCards = [[], []]

Declare Integer cardValue = [[], []]

Declare String cardDeck = []

Declare Integer ans = 6

shuffle(cardDeck)

Declare Integer win = 0

Declare Integer charlie = 0



Declare Integer pic

Declare Integer drawn

Declare Integer picIndex

Declare Integer indent

Declare Integer play_test

Declare Integer oneOrTwoCards

Declare Integer var_money

Declare Integer var_difficulty
Declare Integer a
Declare String j, c


for a = 0 to ans
    for j = "Hearts" to "Diamonds" to "Spades" to "Clubs"
        for c = "1" to "2" to "3" to "4" to "5" to "6" to "7" to "8" to "9" to "10" to "Jack" to "Queen"
to "King" to "Ace"
            cardDeck.append(c " of " j)
        End for
    End for
End for


Module main()
    pic = []
    drawn = 0
    picIndex = 0
    indent = [150, 150]
    play_test = 1
    oneOrTwoCards = 0
    var_money = 100
    var_difficulty = 21
    Call play()
End module


Module play()
    If play_test == 1:
        Call hitMe()
    Else if play_test == 2:
        Call restart()
    End if
End module


Module empty()
    Declare Integer i
    For i = 0 to len(pic)
        pic[i].resize = (0,0)
        indent[0] = 150
        indent[1] = 150
        picIndex = 0
        pic = []
        cardValue = [[], []]
        drawnCards = [[], []]
        win = 0
        charlie = 0
    End for
End module


Module resize()
    Declare Integer i
    btnA[1].setEnabled = (False)
    For i = 0 to len(btnB)
        btnB[i].setEnabled = (False)
    End for
End module


Module bet(Integer n, Integer index)
    Declare Integer var_bet = n
    Declare Integer i
    For i = 0 to 5
```

```
        btnB[i].setStyleSheet("background-color:none;")
    End for
    btnB[index].setStyleSheet("background-color:black;color:white;")
    If var_bet > var_money:
        var_bet = var_money
    End if
    If var_bet < 0:
        var_bet = var_bet
    End if
Module end


Module hitMe()
    Declare String invalidBet
    Declare Integer i
    If var_money <= 0:
        invalidBet.QMessageBox()
        invalidBet.setWindowTitle("Invalid")
        invalidBet.setText("You have insufficient funds to bet at all!\n You can change your money
value in Settings > Change Value.")
        invalidBet.setIcon(QMessageBox.Warning)
        invalidBet.exec_()
        return
    End if
    For i = 0 to len(btnB)
        btnB[i].setEnabled = (False)
    End for
    If oneOrTwoCards < 0:
        oneOrTwoCards = 0
    End if
    If oneOrTwoCards == 0:
        btnA[1].setEnabled = (True)
        Call cardDraw(2, 2, 1, "You", 0, 95)
        Call cardDraw(1, 1, 0, "Dealer", 1, 490)
        oneOrTwoCards++
    Else if oneOrTwoCards > 0:
        cardDraw(1, 1, 1, "You", 0, 95)
    End uf
    var_money = round(var_money, 2)
    money.setText = ("Money: $%s"%(str(var_money)))
End module
```

```
Module cardDraw(Integer handLength, Integer OneOrTwo, Integer plyr, String name, Integer indentIndex, Integer line)
  Declare Integer hand = []
  Declare Integer = i
  For i = 0 to handLength
    hand.append(cardDeck[drawn])
    drawnCards[plyr].append(cardDeck[drawn])
    pic.append(QLabel())
    pic[picIndex].setPixmap = (QPixmap("img/%s.svg"%(cardDeck[drawn])))
    pic[picIndex].setGeometry = (indent[indentIndex], line, 238, 322)
    indent[indentIndex] = indent[indentIndex] + 50
    pic[picIndex].show()
    picIndex++
    drawn++
  End for
  For i = 0 to OneOrTwo
    If "Ace" = hand[i]:
      cardValue[plyr].append(11)
    Else if hand[i][:1] = "Q" or hand[i][:1] = "J" or hand[i][:1] = "K":
            cardValue[plyr].append(10)
        Else:
            cardValue[plyr].append(int(hand[i][0:2]))
        End If
    End for
    Call testForAces(plyr)
    If OneOrTwo == 2
        If sum(cardValue[plyr]) == (var_difficulty):
            card = name "got a" hand[0] "and a " hand[1] ", which is a blackjack!"
            win = 1
            Call check()
        Else:
            card = name "got a" hand[0] "and a " hand[1] ", which is a total of " sum(cardValue[plyr]))
"."
        End if
        player.setText = (card)
        player.setStyleSheet = (("background-color: white;"))
    Else if OneOrTwo == 1
        If len(cardValue[plyr]) >= 5 and sum(cardValue[plyr]) <= (var_difficulty) and plyr == 1:
            card = name " got a " hand[0] ", which is a five card Charlie!"
            charlie = 1
            Call check()
        Else if sum(cardValue[plyr]) == (var_difficulty):
            card = name " got a " hand[0] ", which is a total of" sum(cardValue[plyr]) "."
            Call resize()
            If plyr != 0:
                Call stand()
            End if
        Else if sum(cardValue[plyr]) >= (var_difficulty + 1):
            card = name " got a " hand[0] ", which is" sum(cardValue[plyr]) " and got busted."
            Call resize()
            If plyr == 1:
                call check()
            End if
            play_test = 2
            Call enableBet()
        Else:
            card = name " got a " hand[0] ", which is a total of" sum(cardValue[plyr]) "."
        End if
```

```
        If plyr == 0:
            dealer.setText(card)
            dealer.setStyleSheet(("background-color: white;"))
        Else:
            player.setText(card)
            player.setStyleSheet(("background-color: white;"))
        End if
        Call initCheck()
End module


Module initCheck()
    var_money = round(var_money, 2)
    If (sum(cardValue[1])) >= (var_difficulty + 1):
        var_money = var_money - var_bet
        money.setText("Money: $"str(var_money))
        win.setText("Dealer Win!")
        Call updateCheck()
    End if
End module
```

```
Module updateCheck()

    win.setStyleSheet(("background-color: white;"))

    shuffle(cardDeck)

    drawnCards = [[], []]

    cardValue = [[], []]

    oneOrTwoCards = -1

    win = 0

    drawn = 0

    Call resize()

    play_test = 2

    Call enableBet()

    totalGameCount++

    gameCount.setText("Total Games: " str(setTotalGamesCount))

End module
```

```
Module enableBet()
    Declare Integer i
    For i = 0 to len(btnB)
        btnB[i].setEnable(True)
    End for
End module


Module restart()
    Call resize()
    play_test = 1
    win.setText("")
    dealer.setText("")
    player.setText("")
    Call empty()
    dealer.setStyleSheet(("background-color: transparent;"))
    player.setStyleSheet(("background-color: transparent;"))
    win.setStyleSheet(("background-color: transparent;"))
    Call hitMe()
End module
```

```
Module reset()

    Call resize()

    play_test = 1

    oneOrTwoCards = 0

    win.setText("")

    dealer.setText("")

    player.setText("")

    dealer.setStyleSheet(("background-color: transparent;"))

    player.setStyleSheet(("background-color: transparent;"))

    win.setStyleSheet(("background-color: transparent;"))

    Call empty()

    var_money = 100

    money.setText("Money: $" str(var_money))

    totalGamesCount = 0

    gameCount.setText("Total Games: " str(self.totalGamesCount))

    Call enableBet()

End module
```

```
Module stand()
   btnA[0].setEnabled(False)
   Call resize()
   speed = 100
   While True
    If sum(cardValue[0]) <= 16 and sum(cardValue[1]) != 0 and sum(cardValue[1]) <= (var_difficulty + 1):
         QTest.qWait(speed)
         Call cardDraw(1, 1, 0, "Dealer",1 , 490)
         speed = speed + 200
      Else:
         Call resize()
         play_test = 2
         Call enableBet()
         Call check()
         btnA[0].setEnabled(True)
         break
      End if
   End while
End module
```

IV. Results

Figure 25

Default Game Status

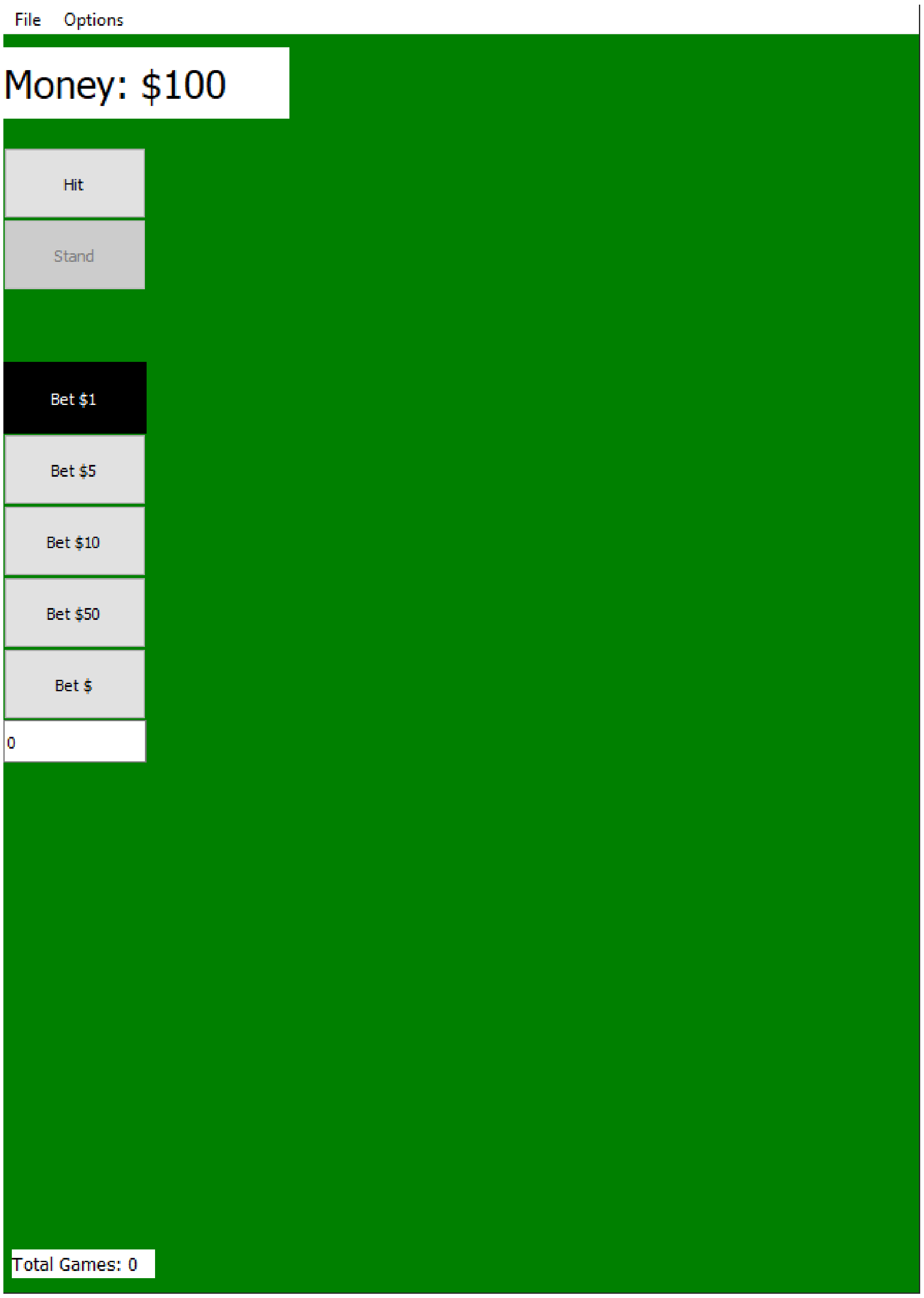
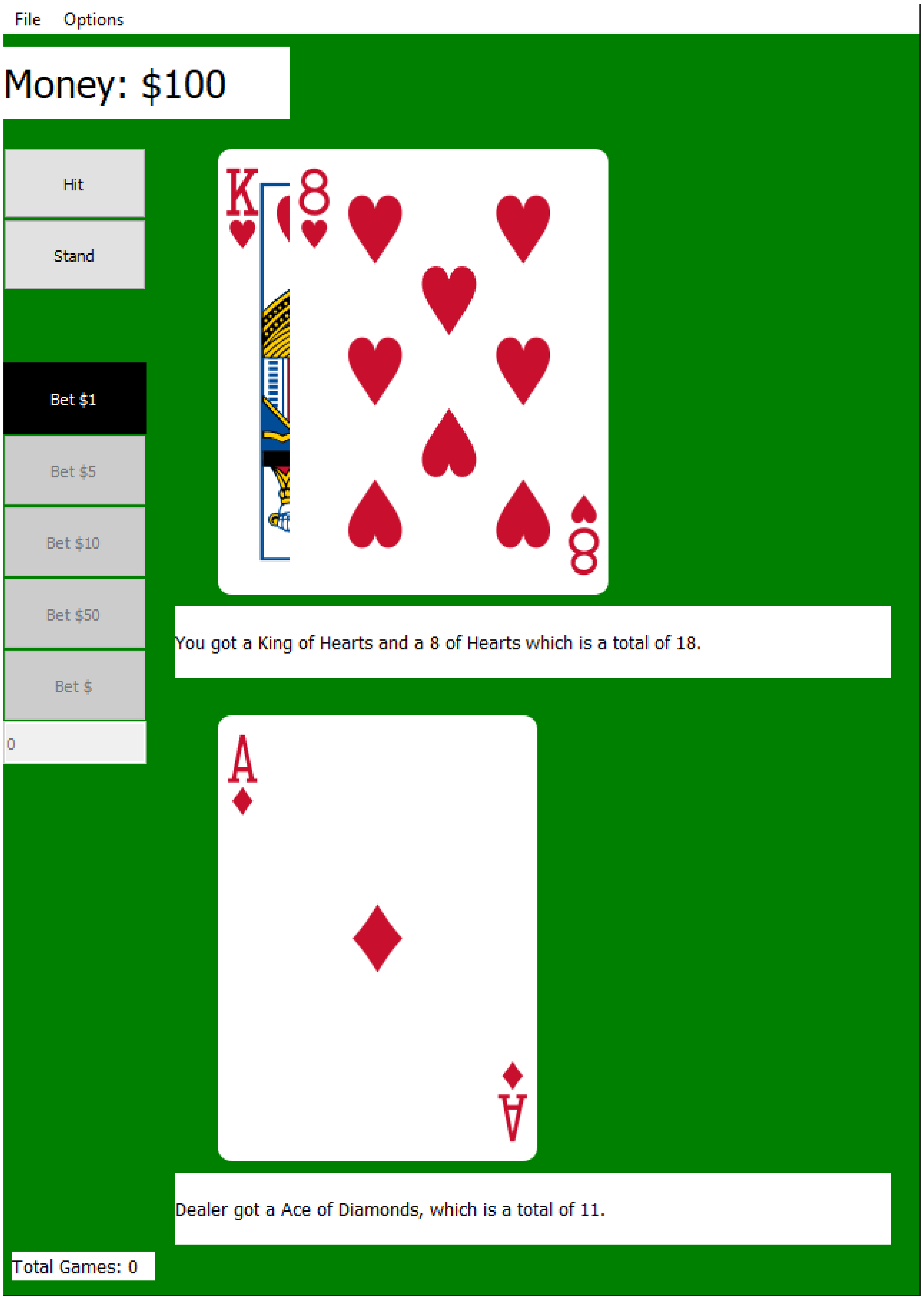


Figure 26

Game 1 - Start



Money: $100

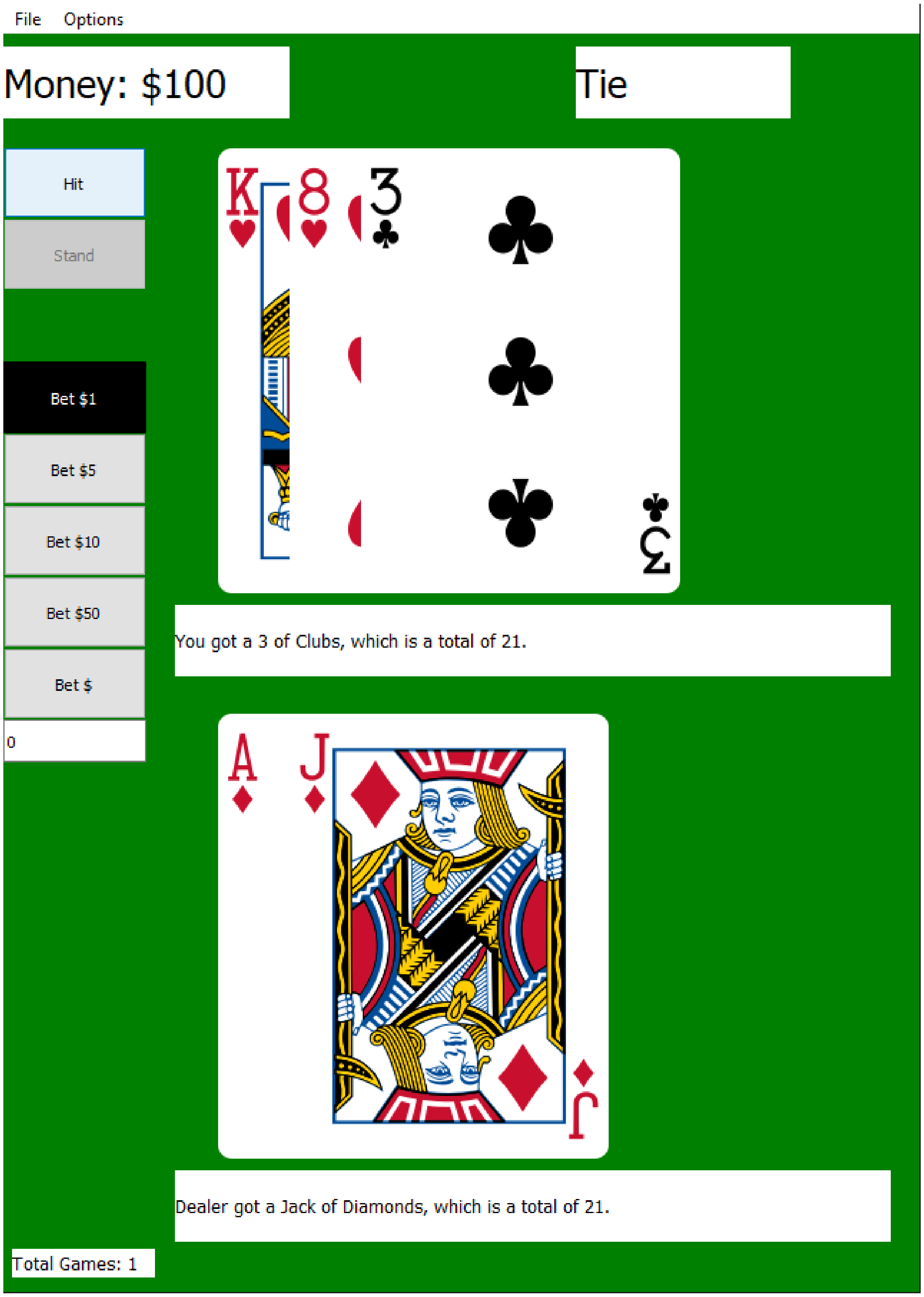
You got a King of Hearts and a 8 of Hearts which is a total of 18.

Dealer got a Ace of Diamonds, which is a total of 11.

Total Games: 0

Figure 27

Game 1 - End (Tie)



Money: $100

Tie

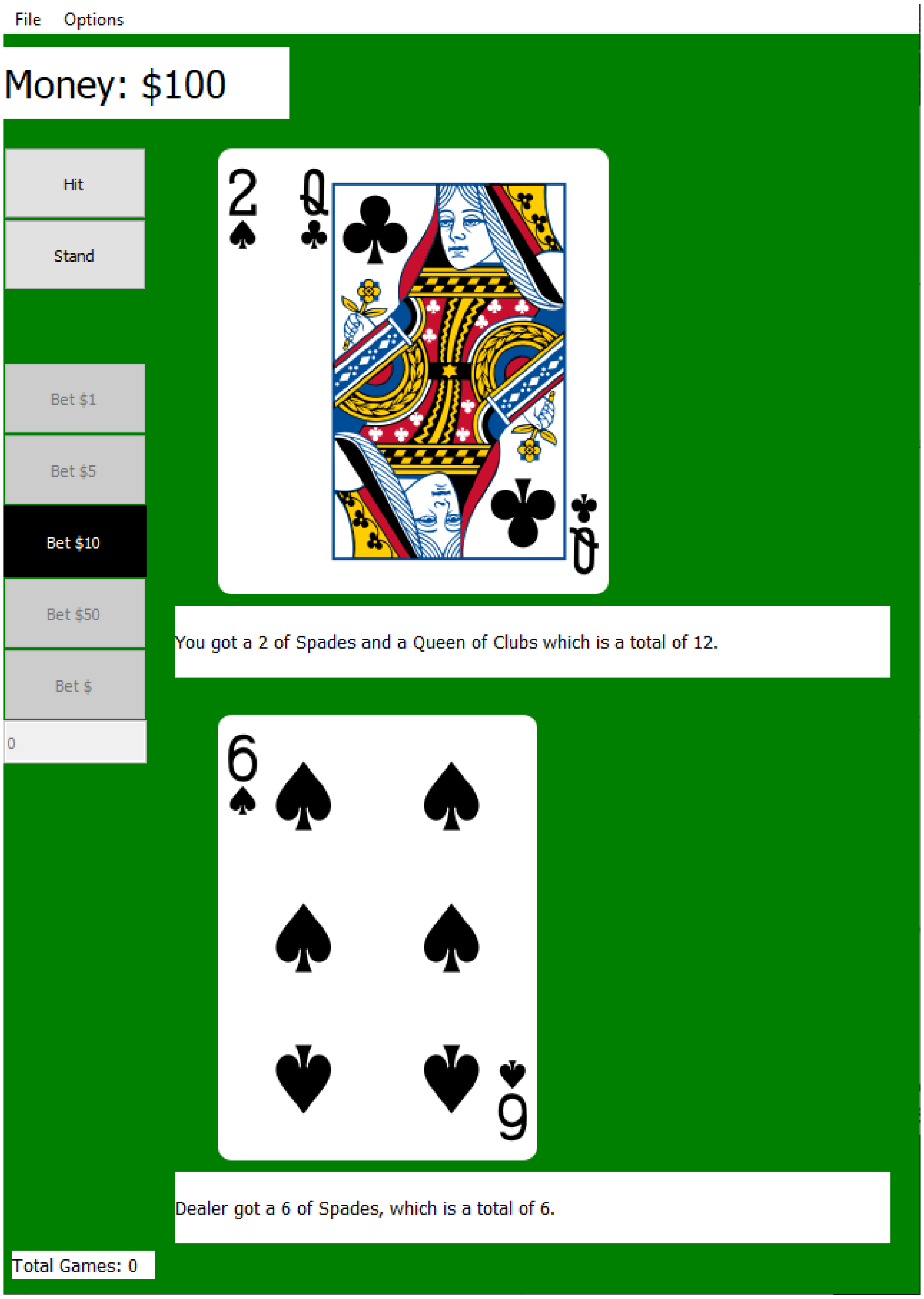
You got a 3 of Clubs, which is a total of 21.

Dealer got a Jack of Diamonds, which is a total of 21.

Total Games: 1

Figure 28

Game 2 - Start



Money: $100

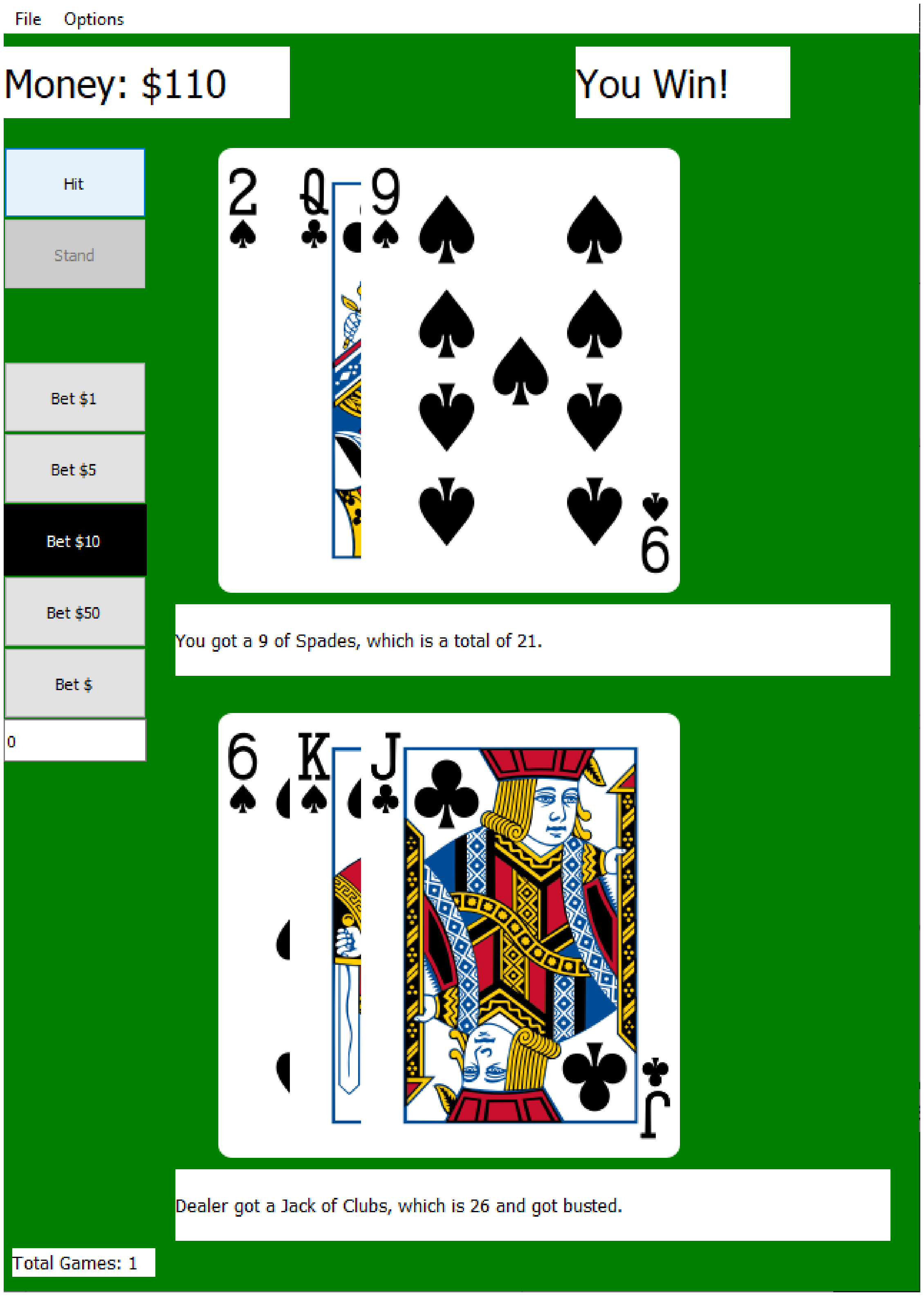
You got a 2 of Spades and a Queen of Clubs which is a total of 12.

Dealer got a 6 of Spades, which is a total of 6.

Total Games: 0

Figure 29

Game 2 - End (User Wins)



Money: $110

You Win!

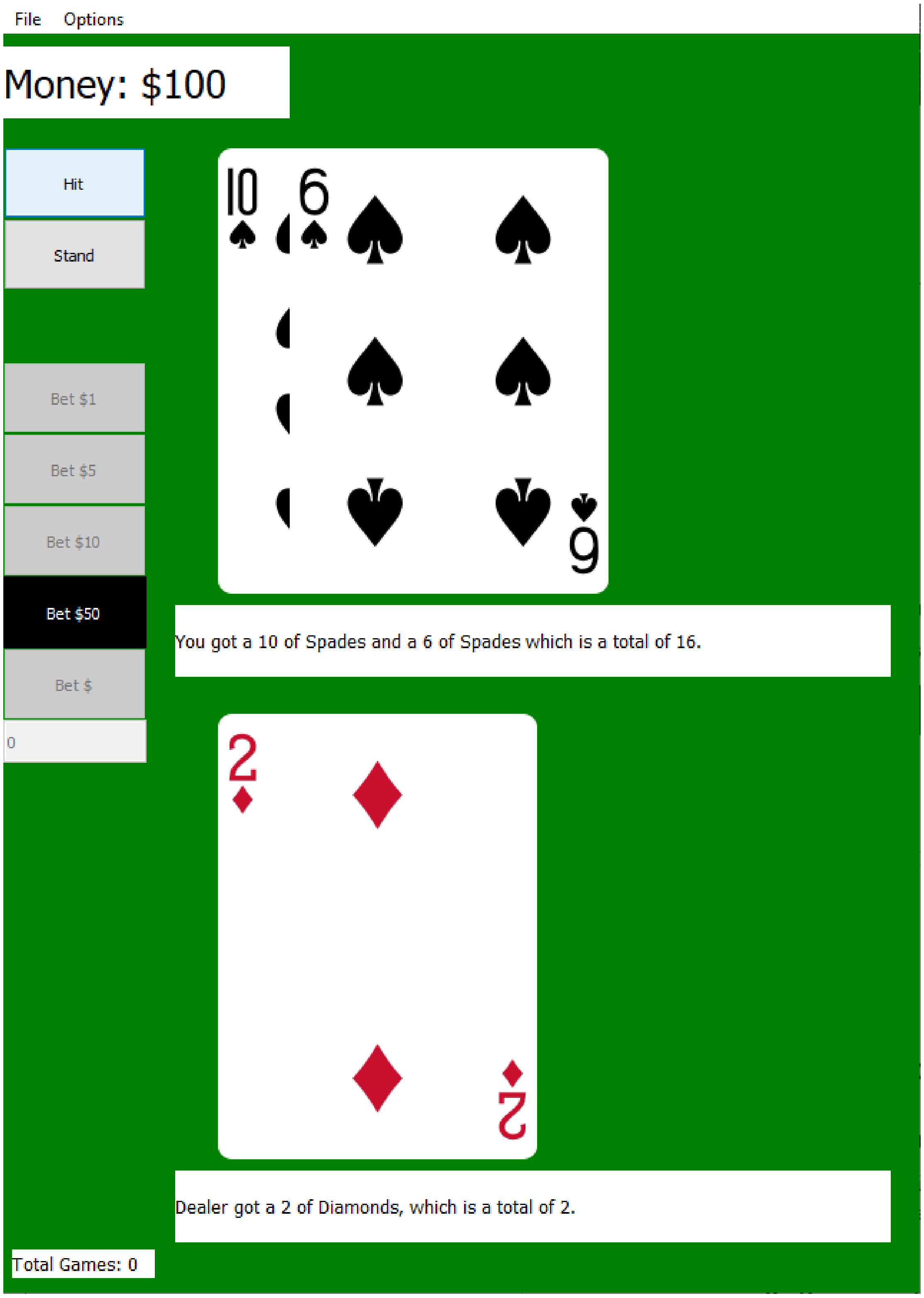
You got a 9 of Spades, which is a total of 21.

Dealer got a Jack of Clubs, which is 26 and got busted.

Total Games: 1

Figure 30

Game 3 - Start



Money: $100

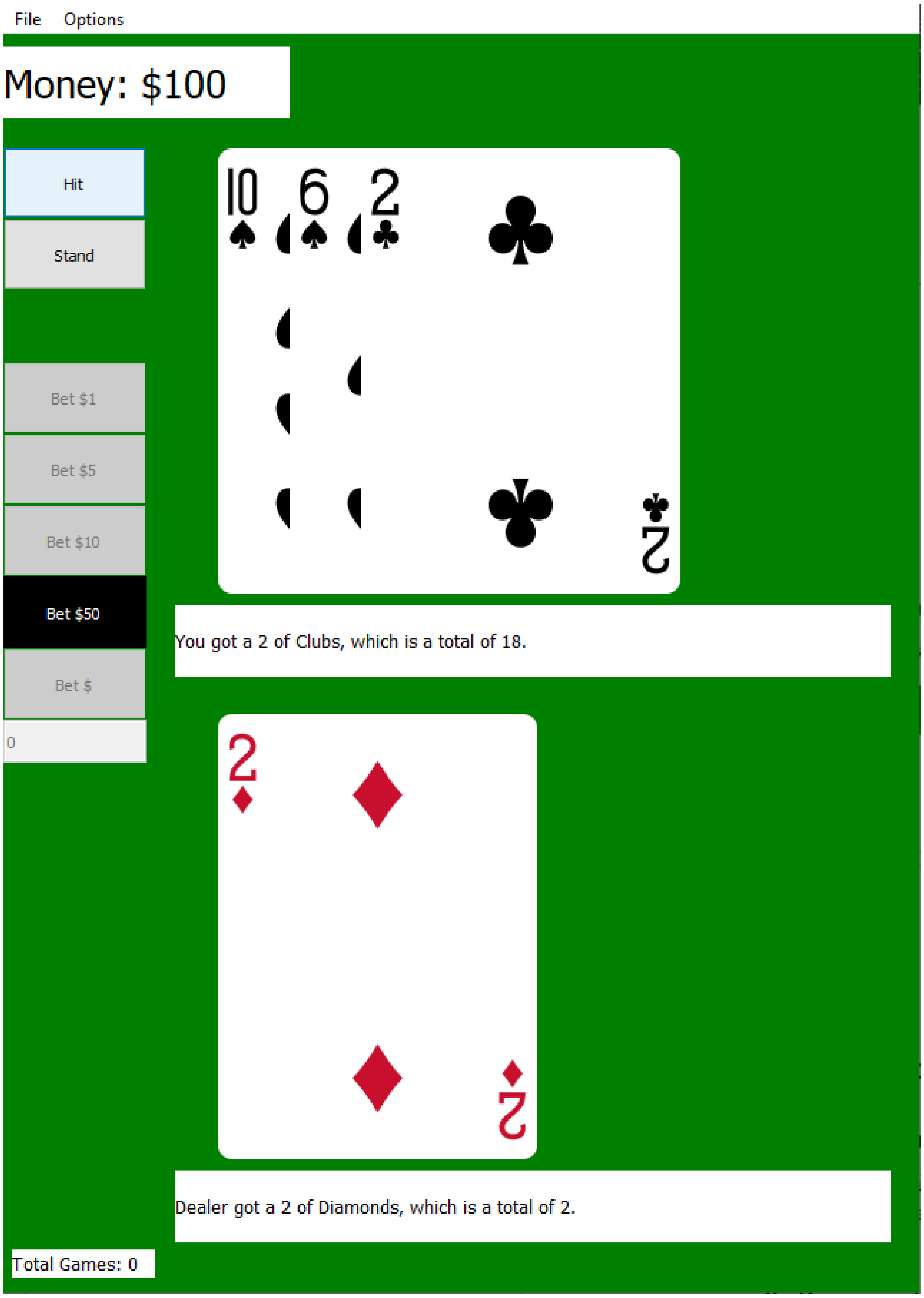
You got a 10 of Spades and a 6 of Spades which is a total of 16.

Dealer got a 2 of Diamonds, which is a total of 2.

Total Games: 0

Figure 31

Game 3 - Continuation



Money: $100

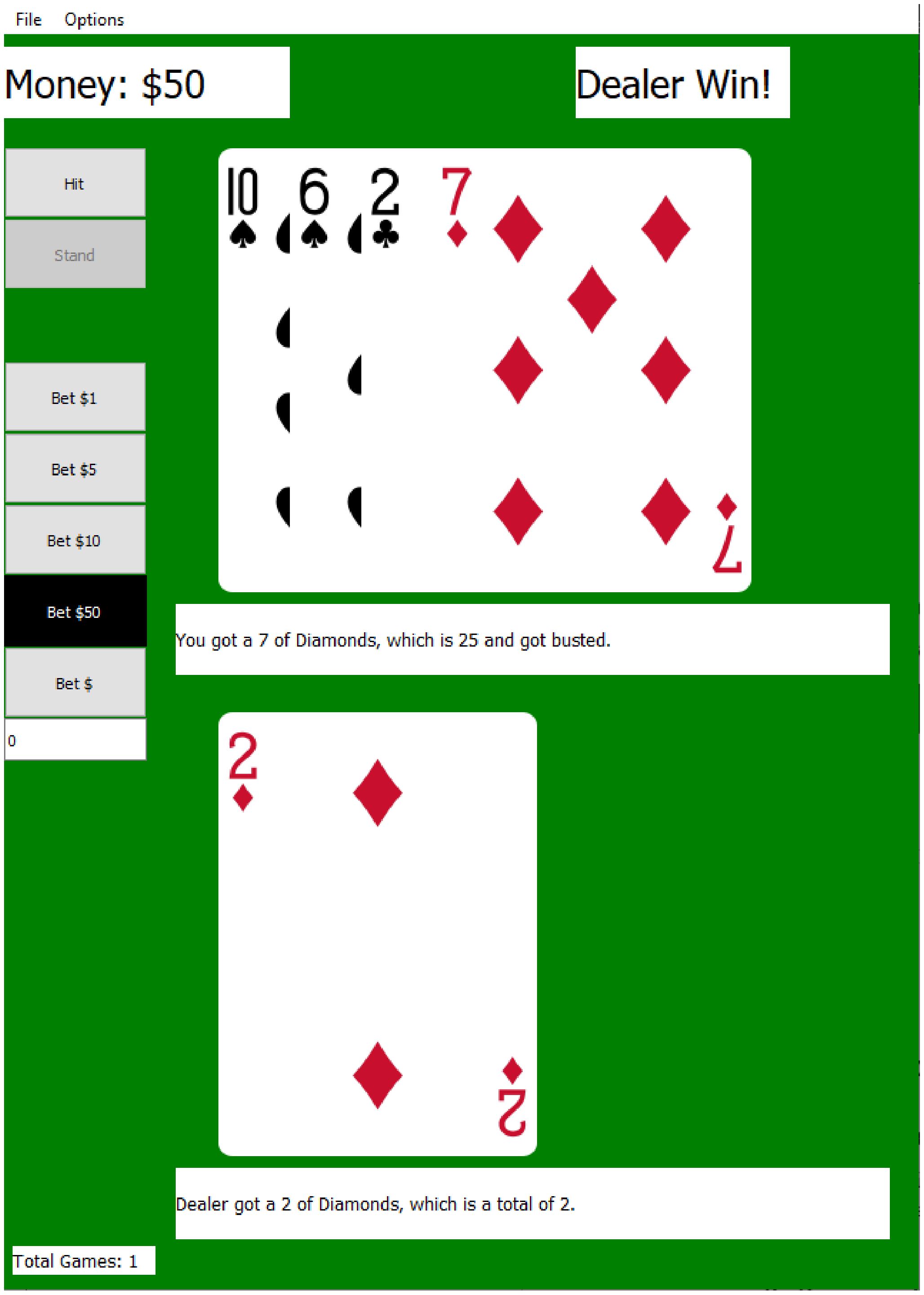
You got a 2 of Clubs, which is a total of 18.

Dealer got a 2 of Diamonds, which is a total of 2.

Total Games: 0

Figure 32

Game 3 - End (Dealer Wins)



Money: $50

Dealer Win!

You got a 7 of Diamonds, which is 25 and got busted.

Dealer got a 2 of Diamonds, which is a total of 2.

Total Games: 1

V. Discussion of Results

The project aimed to create a Blackjack game using the programming language Python. In order to address finding results the group programmed a Blackjack game that enables for the user to experience and play.

Game 1 - Tie

In Game 1 with Standard [21] difficulty, the user received two cards, namely the King of Hearts and the 8 of Hearts, after clicking the "Hit" button. The King of Hearts has an equivalent value of 10, and the 8 of Hearts has a value of 8, which results in a total of 18 for the user's current hand. Meanwhile, the dealer had the Ace of Diamonds, which has a value of 11. After the user played "Hit" one more time, they received a 3 of Clubs with a value of 3, resulting in the user's current hand totalling to 21. Furthermore, the dealer drew a Jack of Diamonds, which has a value of 10, resulting in the dealer's hand also totaling to 21. The evaluation of the game resulted in a "Tie" since both the user and the dealer have a hand of 21. Therefore, the user did not win any money.

Game 2 - Win

In Game 2 with Standard [21] difficulty, the user received two cards: the 2 of Spades and the Queen of Clubs, after clicking the "Hit" button. The 2 of Spades has a value of 2, and the Queen of Clubs has a value of 10, resulting in a total of 12 for the user's current hand. Meanwhile, the dealer had the 6 of Spades, which has a value of 6. After the user played "Hit" one more time, they received a 9 of Spades with a value of 9, resulting in the user's current hand totaling to 21. Furthermore, the dealer drew a King of Spades and a Jack of Clubs, both having a value of 10, resulting in the dealer's hand totaling to 26. The evaluation of the game resulted in the user winning since they had exactly 21, while the dealer went above 21 with a total of 26. Therefore, the user won the bet of $10, which is the amount they chose to bet.

Game 3 - Lose

In Game 3 with Standard [21] difficulty, the user received two cards: the 10 of Spades and the 6 of Spades, after clicking the "Hit" button. The 10 of Spades has a value of 10, and the 6 of Spades has a value of 6, resulting in a total of 16 for the user's current hand. Meanwhile, the dealer had the 2 of Diamonds, which has a value of 2. After the user played "Hit" one more time, they received a 2 of Clubs with a value of 2, resulting in the user's current hand totaling to 18. The user played "Hit" one more time and got the 7 of Diamonds with a value of 7, resulting in a total of 25 for the user's current hand. Therefore, the user lost $50 since they went over 21 and busted.

VI. Analysis, Conclusion and Future Directives

This chapter discussed the conclusion and recommendations regarding the project. It noted the significant findings and fundamental flaws, which summarized the overall project.

A. Conclusion

To simplify the results obtained from the project, Game 1 resulted in a Tie, Game 2 resulted in the user winning, and lastly, Game 3 resulted in the dealer winning. This project has established and programmed the basic rules of the game, Blackjack using the programming language Python successfully. These include the hit, stand, betting, and game evaluation.

B. Recommendations

There are limitations and flows that occur within the project that could be fixed if given enough consideration. The following section will provide possible recommendations that could help amplify the project.

Bugs in the Program

There are a couple of prominent bugs that crash the program. The first bug involves the betting system, where the user can specify the amount they wish to bet. If the user leaves the "Bet $" text box empty, the program will crash. Similarly, the second bug occurs in the settings menu. If the user leaves the "Change player money" text box empty, the program will also crash.

Areas to Improve On

The project was done using the Python programming language and PyQt, however, the implementation is not thoroughly optimal. This has resulted in making the code harder to maintain. Therefore, creating separate .py files for each function that works similarly like header files in C/C++ could make the code easier to manage and readable. In a case separating the main UI, settings UI, and blackjack functionality into separate .py files such as "blackjack_func.py" and "settings.py" files will be imported into the main Python program instead of integrating all functions in one .py file.

The UI/UX of the program could be greatly improved by utilizing the .ui Qt Builder files instead of integrating all PyQt functions into a single .py file, since most of the PyQt functions in the project code cost 300+ lines and implementing sound for feedback to the users as it can significantly change the user experience (e.g playing cards SFX) but this was planned in a very late stage of development so we were unable to implement sound in the program. Additionally, other programming languages could be utilized but the scope of the project is limited to Python.

References

Muldoon, K. (2021, May 3). *Applying game design logic to your design system.* https://uxdesign.cc/applying-game-design-logic-to-your-design-system-111a2116509

Pinto, R. (2018, August). *4 simple steps to designing a strong user experience.* https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/strong-audience-design/

Costa, C. D. (2023). *Top 10 Python GUI Frameworks for Developers*. Built In. Retrieved from https://builtin.com/software-engineering-perspectives/python-gui

Blackjack. (n.d.). BicycleCards. Retrieved from https://bicyclecards.com/how-to-play/blackjack/

Open Source Vector Playing Cards. Total Nonsense. (n.d.). Retrieved March 17, 2023, from https://totalnonsense.com/open-source-vector-playing-cards/

Playing card free icons designed by Freepik. Flaticon. (2020, November 18). Retrieved April 11, 2023, from https://www.flaticon.com/free-icon/playing-card_3769686

PyQt download. Riverbank Computing | Download. (n.d.). Retrieved March 17, 2023, from https://riverbankcomputing.com/software/pyqt/download

What is Python? Executive Summary. (n.d.). Python.org. Retrieved March 17, 2023, from https://www.python.org/doc/essays/blurb/

Appendices

The following appendix showcased the User's Manual, Source Code, Work breakdown, and Personal Data Sheet.

A. User's Manual

The User's Manual for the program BlackJack.

Prerequisites

The program needs the following dependencies / packages installed:

1. Python 3.9 (Python 3.10+ breaks PyQt5-tools)
    a. (Link for download: https://www.python.org/downloads/release/python-3913/)
2. PyQt5
    b. pip install PyQt5 (Run command prompt and type the command.)
3. PyQt5-tools [Optional]
    c. pip install PyQt5-tools (Run command prompt and type the command.)

Integrated Development Environment

The recommended IDE for the program to run is PyCharm. Install from the official site: https://www.jetbrains.com/pycharm/

Running the Program

Steps and procedures:

1. Download the code/program as Zip in GitHub. https://github.com/PercivalIII/py3AquamanProject
2. Open the “BlackJack” folder in PyCharm.
3. Once in PyCharm, open the file “blackjack.py” and run it.

Functions

The functions of the program will be discussed in this section, such as hit, stand, betting, amount of money, difficulty, reseting, and exiting.

Hit and Stand

Click “Hit” to request another card from the dealer. Click “Stand” to not ask for another card from the dealer and end the round.

Betting

To change the amount of money for betting, click on any of the betting amounts on the side, such as “Bet $1”, “Bet $5”, “Bet $10”, “Bet $50”, and “Bet $” (Specify the amount of money for betting).

Amount of Money

To change the amount of money for the entire game, go to “Options” in the top left corner then click on “Change Values” This will open the settings window. Go to the “Change player money” text box and input the amount of money wanted. Once accomplished, click “Apply” to apply the changes, then “Exit”

Difficulty

To change the difficulty for the entire game, go to “Options” in the top left corner then click on “Change Values” This will open the settings window. Click on any of the options such as “Easy [30]”, “Standard [21]”, “Intermediate [16], and “Hard [12]”. Once accomplished, click “Apply” to apply the changes, then “Exit”.

Reset the Game

To reset the game, go to “Options” in the top left corner then click on “Reset Game”. This wipe the player and dealer's card deck in the main program's GUI, then resets variables such as 'cardValue' and 'drawnCard', and also sets 'win' and 'charlie' to 0.

Exiting the Program

To exit the game, go to “File” in the top left corner then click on “Exit application”

B. Source Code

```
import sys

# Importing several PyQt5 libraries for the program's user interface.

from PyQt5.QtWidgets import QApplication, QLabel, QPushButton, QLineEdit, QAction,
QMainWindow, QMessageBox

from PyQt5.QtGui import QIcon, QPixmap, QFont, QIntValidator

from PyQt5 import QtCore

# Import 'random' function for the shuffle mechanism in the blackjack game.

from random import shuffle

from PyQt5.QtTest import QTest


class Window(QMainWindow):


  # Initializes the PyQt5 formatting of the main program window.
```

```
def __init__(self):
    super(Window, self).__init__()
    self.setGeometry(0, 0, 640, 900)
    self.setFixedSize(640, 900)
    self.setWindowTitle("LBYCPA1 AQUAMAN - Blackjack")
    self.setWindowIcon(QIcon("favicon.ico.png"))

    Exit = QAction("&Exit application", self)
    Exit.setShortcut("Ctrl+Alt+E")
    Exit.triggered.connect(self.close)

    optionA = QAction("&Change Values", self)
    optionA.triggered.connect(self.openSettings)

    optionB = QAction("&Reset Game", self)
    optionB.triggered.connect(self.reset)

    mainMenu = self.menuBar()
    fileMenu = mainMenu.addMenu("&File")
    editMenu = mainMenu.addMenu("&Options")
    fileMenu.addAction(Exit)

    editMenu.addAction(optionA)

    editMenu.addAction(optionB)


    sshFile = "style.css"

    with open(sshFile,"r") as fh:

        self.setStyleSheet(fh.read())


    self.home() # Calls the function home() to load the main



#

# START

# A.) Main Program Graphical User Interface (GUI)

#



def home(self):

    # Declaring 'btnA' and 'btnB' as a array list for the buttons in home().

    # Each button is classified by its variable names.

    # 'btnA' is used for actions such as "Hit" and "Stand", while 'btnB' is used for betting.

    self.btnA=[]
self.btnB=[]


# Appending button elements in the btnA[] and btnB[] arrays.

self.btnA.append(QPushButton("Hit", self)) # btnA[0]

self.btnA.append(QPushButton("Stand", self)) # btnA[1]


self.btnB.append(QPushButton("Bet $1", self)) # btnB[0]

self.btnB.append(QPushButton("Bet $5", self)) # btnB[1]

self.btnB.append(QPushButton("Bet $10", self)) # btnB[2]

self.btnB.append(QPushButton("Bet $50", self)) # btnB[3]

self.btnB.append(QPushButton("Bet $", self)) # btnB[4]

self.btnB.append(QLineEdit("0", self)) # btnB[5]


self.resize()

# Every variable data here is used for storing information.

self.var_bet = 1 # The player's amount to bet.

self.pic = []

self.drawn = 0

self.picIndex = 0

self.indent = [150, 150]
self.play_test = 1 # If it's 1 -> The game starts, if it's 2 -> the UI application restarts the blackjack deck, then starts the game. Used for conditional statements.

self.oneOrTwoCards = 0

self.totalGamesCount = 0 # The player's overall game count, every game increments to 1.

self.var_money = 100 # The player's money.

self.var_difficulty = 21 # The value to get a blackjack in normal games in 21. Exceeding 21 is a instant lose. This is declared as a integer variable for the adjustable difficulty.


# 'btnA' buttons for hitting & standing.

self.btnA[0].clicked.connect(self.play)

self.btnA[0].resize(100, 50)

self.btnA[0].move(0, 100)

self.btnA[0].setShortcut("H")


self.btnA[1].clicked.connect(self.stand)

self.btnA[1].move(0, 150)

self.btnA[1].resize(100, 50)

self.btnA[1].setShortcut("S")


# Declaring 'money' and 'gameCount' as text elements, set as color white.
# Each text element gets the integer variable data from 'var_money' and 'totalGamesCount'
to show the player's money and their total games.

self.money = QLabel(("Money: $%s"%(str(self.var_money))), self)

self.money.setStyleSheet(("background-color: white;"))



self.gameCount = QLabel(("Total Games: %s"%(str(self.totalGamesCount))), self)

self.gameCount.setStyleSheet(("background-color: white;"))



# btnB buttons for betting.

self.btnB[0].clicked.connect(lambda: self.bet(1, 0))

self.btnB[0].resize(100, 50)

self.btnB[0].move(0, 250)

self.btnB[0].setEnabled(True)

self.btnB[0].setStyleSheet("background-color:black;color:white;")

self.btnB[0].setShortcut("1")



self.btnB[1].clicked.connect(lambda: self.bet(5, 1)) # Bet $5 button

self.btnB[1].resize(100, 50)

self.btnB[1].move(0, 300)

self.btnB[1].setEnabled(True)
```

```
        self.btnB[1].setShortcut("2")


        self.btnB[2].clicked.connect(lambda: self.bet(10, 2)) # Bet $10 button
        self.btnB[2].resize(100, 50)
        self.btnB[2].move(0, 350)
        self.btnB[2].setEnabled(True)
        self.btnB[2].setShortcut("3")


        self.btnB[3].clicked.connect(lambda: self.bet(50, 3)) # Bet $50 button
        self.btnB[3].resize(100, 50)
        self.btnB[3].move(0, 400)
        self.btnB[3].setEnabled(True)
        self.btnB[3].setShortcut("4")


        self.btnB[4].clicked.connect(lambda: self.bet(int(self.btnB[5].text()), 4)) # Bet $ button
option
        self.btnB[4].resize(100, 50)
        self.btnB[4].move(0, 450)
        self.btnB[4].setEnabled(True)
        self.btnB[4].setShortcut("5")
```

```
validator = QIntValidator()

self.btnB[5].setValidator(validator)

self.btnB[5].textEdited.connect(lambda: self.bet(int(self.btnB[5].text()), 4)) # Bet $ textbox

self.btnB[5].resize(100, 30)

self.btnB[5].move(0, 500)

self.btnB[5].setEnabled(True)


# Declaring fortA and fontB for adjusting text font sizes in elements.

fontA = QFont()

fontB = QFont()

fontA.setPointSize(20)

fontB.setPointSize(10)


# Element for money indicator (Player).

self.money.move(0, 30)

self.money.resize(200, 50)

self.money.setFont(fontA)


# Element for total play amount indicator.
```

```
self.gameCount.move(6, 870)

self.gameCount.resize(120, 20)

self.gameCount.setFont(fontB)


# Element for Player card deck text element.

self.player = QLabel("", self)

self.player.setStyleSheet(("background-color: transparent;"))

self.player.move(120, 420)

self.player.resize(500, 50)

self.player.setFont(fontB)


# Element for Dealer card deck text element.

self.dealer = QLabel("", self)

self.dealer.move(120, 815)

self.dealer.resize(500, 50)

self.dealer.setStyleSheet(("background-color: transparent;"))

self.dealer.setFont(fontB)


# Element for the "You Win!", "Dealer Win!", and "Tie!" indicator.

self.win = QLabel("", self)
```

```
        self.win.move(400, 30)

        self.win.resize(150,  50)

        self.win.setStyleSheet(("background-color: transparent;"))

        self.win.setFont(fontA)



        self.show() # Calls the function show() to initialize the changes made in PyQt5.



    # Function openSettings() is used for opening the Settings window.

    def openSettings(self):

        self.window = QMainWindow()

        self.settingsUI(self.window)

        self.window.show() # Calls the function show() to initialize and display  the settings
window.



    #

    # END

    # A.) Main Program Graphical User Interface (GUI)

    #



    #
```

```
# START

# B.) Settings Graphical User Interface (GUI)

#



# Function settingsUI() to initialize Settings window.

def settingsUI(self, settingsUI):



    # Declaring integer variables to separate it from the variables used for the blackjack game.

    # The integer variables here are used to check the changes made before processing it to the
real blackjack game decided by the user.

    self.difficultyValue = self.var_difficulty

    self.moneyValue = self.var_money

    self.difficultyValueChange = 21



    # Initializes the Qt5 formatting of the Settings window.

    settingsUI.setObjectName("settingsUI")

    settingsUI.resize(400, 200)

    settingsUI.setFixedSize(settingsUI.width(), settingsUI.height())

    settingsUI.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint, False)
```

```
# Elements for text indicators to the player money and difficulty value.

self.changeMoneyLabel = QLabel(settingsUI)

self.changeMoneyLabel.setGeometry(20, 20, 111, 16)

self.changeMoneyLabel.setObjectName("changeMoneyLabel")



self.changeDifficultyLabel = QLabel(settingsUI)

self.changeDifficultyLabel.setGeometry(20, 50, 101, 16)

self.changeDifficultyLabel.setObjectName("changeDifficultyLabel")



# Elements for textboxs.

# The player money textbox 'changePlayerMoneyLine' is editable to any numerical values
in order to change it by the user.

# The game difficulty textbox 'changeDifficultyLine' is read-only; only used as an indicator
for the user to see changes/updates.

self.changePlayerMoneyLine = QLineEdit(settingsUI)

self.changePlayerMoneyLine.setGeometry(140, 20, 113, 20)

self.changePlayerMoneyLine.setObjectName("changePlayerMoneyLine")



self.changeDifficultyLine = QLineEdit(settingsUI)

self.changeDifficultyLine.setGeometry(140, 50, 113, 20)

self.changeDifficultyLine.setObjectName("changeDifficultyLine")
```

```
        self.changeDifficultyLine.setReadOnly(True)


        # Elements for buttons in the Settings UI.

        self.applyButton = QPushButton(settingsUI)

        self.applyButton.setGeometry(200, 160, 75, 23)

        self.applyButton.setObjectName("applyButton")

        self.applyButton.clicked.connect(lambda: self.settingsUpdate())


        self.exitButton = QPushButton(settingsUI)

        self.exitButton.setGeometry(290, 160, 75, 23)

        self.exitButton.setObjectName("exitButton")

        self.exitButton.clicked.connect(settingsUI.close)


        self.difficultyStandard = QPushButton(settingsUI)

        self.difficultyStandard.setGeometry(30, 80, 120, 23)

        self.difficultyStandard.setObjectName("difficultyStandard")

        self.difficultyStandard.clicked.connect(lambda: self.difficultyChange(21)) # Standard
Difficulty [21] button


        self.difficultyEasy = QPushButton(settingsUI)
        self.difficultyEasy.setGeometry(30, 110, 120, 23)

        self.difficultyEasy.setObjectName("difficultyEasy")

        self.difficultyEasy.clicked.connect(lambda: self.difficultyChange(30)) # Easy Difficulty
[30] button



        self.difficultyIntermediate = QPushButton(settingsUI)

        self.difficultyIntermediate.setGeometry(30, 140, 120, 23)

        self.difficultyIntermediate.setObjectName("difficultyIntermediate")

        self.difficultyIntermediate.clicked.connect(lambda: self.difficultyChange(16)) #
Intermediate Difficulty [16] button



        self.difficultyHard = QPushButton(settingsUI)

        self.difficultyHard.setGeometry(30, 170, 120, 23)

        self.difficultyHard.setObjectName("difficultyHard")

        self.difficultyHard.clicked.connect(lambda: self.difficultyChange(12)) # Hard Difficulty
[12] button



        # Elements for the status of the game values in the Settings UI.

        self.currentStatusLabel = QLabel(settingsUI)

        self.currentStatusLabel.setGeometry(210, 80, 111, 16)

        self.currentStatusLabel.setObjectName("currentStatusLabel")
        self.playerMoneyStatus = QLabel(settingsUI)
        self.playerMoneyStatus.setGeometry(220, 100, 180, 16)
        self.playerMoneyStatus.setObjectName("playerMoneyStatus")

        self.blackjackDifficultyStatus = QLabel(settingsUI)
        self.blackjackDifficultyStatus.setGeometry(220, 115, 121, 16)
        self.blackjackDifficultyStatus.setObjectName("blackjackDifficultyStatus")

        self.retranslateUi(settingsUI) # Calls the function retranslateUi(settingsUI) to initialize the settings text elements.
        QtCore.QMetaObject.connectSlotsByName(settingsUI)

        # Calls the function setText to specific variable data to change the text content.
        self.changeDifficultyLine.setText("%d"%(self.difficultyValue))
        self.blackjackDifficultyStatus.setText("Blackjack difficulty = %d"%(self.var_difficulty))

        self.changePlayerMoneyLine.setText("%d"%(self.moneyValue))
        self.playerMoneyStatus.setText("Player Money = $%d"%(self.var_money))
```

```
    # Function difficultyChange(value) to adjust the difficulty of the blackjack game by changing
the '21' rule.

    def difficultyChange(self, value):

        self.difficultyValueChange = value

        self.changeDifficultyLine.setText("%d"%(value))



    # Function settingsUpdate() to apply every change made by the user to the blackjack game
such as their money value and the game difficulty.

    def settingsUpdate(self):

        self.var_difficulty = self.difficultyValueChange

        self.var_money = int(self.changePlayerMoneyLine.text())

        self.money.setText("Money: $%s"%(str(self.var_money)))

        self.changePlayerMoneyLine.setText("%d"%(self.moneyValue))

        self.playerMoneyStatus.setText("Player Money = $%d"%(self.var_money))

        self.blackjackDifficultyStatus.setText("Blackjack difficulty = %d"%(self.var_difficulty))



        # Intiializes the PyQt5 pop-up messagebox to remind the user that the game's settings has
been changed/updated.

        self.confirmUser = QMessageBox()

        self.confirmUser.setWindowTitle("Confirm")

        self.confirmUser.setText("The changes has been made!")
        self.confirmUser.setIcon(QMessageBox.Information)

        self.confirmUser.exec_() # Calls function exec_() to display the messagebox.


    # Function retranslateUi(settingsUI) to initialize the Settings text elements.

    def retranslateUi(self, settingsUI):

        _translate = QtCore.QCoreApplication.translate

        settingsUI.setWindowTitle(_translate("settingsUI", "Settings"))

        self.changeMoneyLabel.setText(_translate("settingsUI", "Change player money:"))

        self.changeDifficultyLabel.setText(_translate("settingsUI", "Change difficulty:"))

        self.applyButton.setText(_translate("settingsUI", "Apply"))

        self.exitButton.setText(_translate("settingsUI", "Exit"))

        self.difficultyStandard.setText(_translate("settingsUI", "Standard [21]"))

        self.difficultyEasy.setText(_translate("settingsUI", "Easy [30]"))

        self.difficultyIntermediate.setText(_translate("settingsUI", "Intermediate [16]"))

        self.difficultyHard.setText(_translate("settingsUI", "Hard [12]"))

        self.currentStatusLabel.setText(_translate("settingsUI", "Current status:"))

        self.playerMoneyStatus.setText(_translate("settingsUI", "Player Money = $"))

        self.blackjackDifficultyStatus.setText(_translate("settingsUI", "Blackjack difficulty = 21"))


    #
```

```
    # END

    # B.) Settings Graphical User Interface (GUI)

    #



    #

    # START

    # C.) Blackjack Functionality

    #



    # Function play() to start the game depending on the 'play_test' value.

    def play(self):

        if self.play_test == 1:

            self.hitMe()

        elif self.play_test == 2:

            self.restart()



    # Function empty() to wipe the player and dealer's card deck in the main program's GUI, resets
variables such as 'cardValue' and 'drawnCard', and also sets 'win' and 'charlie' to 0.

    def empty(self):

        global drawnCards, cardValue, win, charlie
        for i in range(0, len(self.pic)):
            self.pic[i].resize(0, 0)
        self.indent[0], self.indent[1], self.picIndex, self.pic, cardValue, drawnCards = 150, 150, 0,
[], [[], []], [[], []]
        win = 0
        charlie = 0


    # Function resize() to update the changes made in the main program's GUI overall.
    def resize(self):
        self.btnA[1].setEnabled(False)
        for i in range(0, len(self.btnB)):
            self.btnB[i].setEnabled(False)


    # Function bet(n, index) to process the player's bet;
    def bet(self, n, index):
        self.var_bet = n
        for i in range(0, 5):
            self.btnB[i].setStyleSheet("background-color:none;")
        self.btnB[index].setStyleSheet("background-color:black;color:white;")
        if self.var_bet > self.var_money:
            self.var_bet = self.var_money

        if self.var_bet < 0:

            self.var_bet = self.var_bet



    # Function hitMe() for the player to hit a card.

    def hitMe(self):

        # When the player has less than $0 money, this message is triggered and terminates the
function.

        if self.var_money <= 0:

            self.invalidBet = QMessageBox()

            self.invalidBet.setWindowTitle("Invalid")

            self.invalidBet.setText("You have insufficient funds to bet at all!\n You can change your
money value in Settings > Change Value.")

            self.invalidBet.setIcon(QMessageBox.Warning)

            self.invalidBet.exec_() # Calls function exec_() to display the messagebox.

            self.enableBet()

            return



        # When the player starts to hit the dealer, all betting options will be inaccessible until the
game ends.

        for i in range(0, len(self.btnB)):
```

```
    self.btnB[i].setEnabled(False)


if self.oneOrTwoCards < 0:
    self.oneOrTwoCards = 0


# This is where the game starts.
# Player draws 2 cards, while dealer draws 1 card.
if self.oneOrTwoCards == 0:
    self.btnA[1].setEnabled(True)
    self.cardDraw(2, 2, 1, "You", 0, 95)
    self.cardDraw(1, 1, 0, "Dealer", 1, 490)
    self.oneOrTwoCards += 1


# If the player has one or two cards in their card deck, they can only draw 1 card.
# The dealer is also affected to this.
elif self.oneOrTwoCards > 0:
    self.cardDraw(1, 1, 1, "You", 0, 95)


self.var_money = round(self.var_money, 2) # Turns 'var_money' into an decimal if the
player's win contains less than $1.
```

```
        self.money.setText("Money: $%s"%(str(self.var_money)))


    # Function cardDraw() if the player or dealer draws a card.

    # The arguments of cardDraw(self, handLength, OneOrTwo, plyr, name, indentIndex, line)
goes as follows:

    # - handLength is the amount of cards in their deck.

    # - OneOrTwo if the

    # - plyr to classify the owner of the card deck. plyr = 1 is the dealer, while plyr = 2 is the
player.

    # - name to indicate whose card draw is, and is also used for the player / dealer card deck
status in the text box below.

    # - indentIndex and line is used for formatting purposes for the card deck being displayed in
the main program's GUI.

    def cardDraw(self, handLength, OneOrTwo, plyr, name, indentIndex, line):

        global win, drawnCards, cardValue, charlie # Function to access the global variable data
'win', 'drawnCards', 'cardValue', and 'charlie'.

        hand = []

        for i in range(0, handLength):

            # Appends the content of the item specified in 'cardDeck[self.drawn]' in 'card' and
'drawnCards[plyr].

            hand.append(cardDeck[self.drawn])

            drawnCards[plyr].append(cardDeck[self.drawn])
            # The process of displaying the player / dealer card deck in the main program's GUI.

            # Technically, the vector images '.svg' of the 52 standard card-deck are used for
displaying the player / dealer card deck.

            # The vector images are named in the same structure as 'cardDeck[]' items are called.


            # The structures goes: ["1 to 9 or King/Queen/Jack" of "Diamonds/Clubs/Hearts/Spades"]


            self.pic.append(QLabel(self))

            self.pic[self.picIndex].setPixmap(QPixmap("img/%s.svg"%(cardDeck[self.drawn])))

            self.pic[self.picIndex].setGeometry(self.indent[indentIndex], line, 238, 322)

            self.indent[indentIndex] += 50

            self.pic[self.picIndex].show()


            # Integer variables increments to 1.

            self.picIndex += 1

            self.drawn += 1


        # Looping process to calculate the value sum of the player / dealer card deck.

        for i in range(0, OneOrTwo):
            if "Ace" in hand[i]:

                cardValue[plyr].append(11)

            elif hand[i][:1] in ("Q", "J", "K"):

                cardValue[plyr].append(10)

            else:

                cardValue[plyr].append(int(hand[i][0:2])) # 0:2 takes the 2 first letters from the string.

                # Checks the contents of array list 'hand[i]', and only checks the number.

                # e.g If 'hand[1]' contains '6 of Diamonds', 'cardValue[plyr]' will only append the
integer '6' only.



        self.testForAces(plyr)



        # Every operation calculates the sum of every item (individual card value), stored in the
array list 'cardValue[plyr]'.

        if OneOrTwo == 2:

            if sum(cardValue[plyr]) == (self.var_difficulty):

                card = "%s got a %s and a %s, which is a blackjack!"%(name, hand[0], hand[1])

                win = 1

                self.check()

            else:
                card = "%s got a %s and a %s which is a total of %s."%(name, hand[0], hand[1],
sum(cardValue[plyr]))

            self.player.setText(card)

            self.player.setStyleSheet(("background-color: white;"))



        elif OneOrTwo == 1:

            if len(cardValue[plyr]) >= 5 and sum(cardValue[plyr]) <= (self.var_difficulty) and plyr
== 1: # If the player's card deck contains 5 cards, while the player's card deck sum value is
higher than the dealer's card deck.

                card = "%s got a %s, which is a five card Charlie!"%(name, hand[0])

                charlie = 1

                self.check()

            elif sum(cardValue[plyr]) == (self.var_difficulty):

                card = "%s got a %s, which is a total of %s."%(name, hand[0], sum(cardValue[plyr]))
# If the player's card deck sum value is equivalent to 'var_difficulty' value.

                self.resize()

                if plyr != 0:

                    self.stand()

            elif sum(cardValue[plyr]) >= (self.var_difficulty + 1): # If the player's card deck sum
value is less than or equal to 'var_difficulty + 1' value.

                card = "%s got a %s, which is %s and got busted."%(name, hand[0],
sum(cardValue[plyr]))
```

```
            self.resize()

            if plyr == 1:

                self.check()

            self.play_test = 2

            self.enableBet()

        else:

            card = "%s got a %s, which is a total of %s."%(name, hand[0], sum(cardValue[plyr]))


        if plyr == 0: # Updates the dealer's card deck text box if plyr == 0.

            self.dealer.setText(card)

            self.dealer.setStyleSheet(("background-color: white;"))

        else: # Updates the player's card deck text box if plyr != 0, assuming plyr == 1.

            self.player.setText(card)

            self.player.setStyleSheet(("background-color: white;"))


    self.initCheck()


# Function testForAces() to check for 'Ace' cards in the player / dealer drawn card.F

def testForAces(self, plyr):

    if sum(cardValue[plyr]) >= (self.var_difficulty + 1):
        for i in range(0, len(cardValue[plyr])):
            if "Ace" in drawnCards[plyr][i]:
                if cardValue[plyr][i] == 1:
                    continue
                else:
                    cardValue[plyr][i] = 1
                    break


    # Function check() to process if the player or dealer is the winner, otherwise it's a tie.
    def check(self):
        global win, cardValue, drawnCards, charlie # Function to access the global variable data
'win', 'drawnCards', 'cardValue', and 'charlie'.
        self.var_money = round(self.var_money, 2) # Turns 'var_money' into an decimal in case the
player's win contains less than $1.
        if win == 1:
            self.var_money += self.var_bet * 1.5
            self.money.setText("Money: $%s"%(str(self.var_money)))
            self.win.setText("You Win!")


        elif charlie == 1:
            self.var_money += self.var_bet
            self.money.setText("Money: $%s"%(str(self.var_money)))

            self.win.setText("You Win!")


        elif sum(cardValue[1]) == sum(cardValue[0]):

            self.win.setText("Tie")


        elif (sum(cardValue[1]) >= (self.var_difficulty + 1) or sum(cardValue[1]) <
sum(cardValue[0])) and not sum(cardValue[0]) >= (self.var_difficulty + 1):

            self.var_money -= self.var_bet

            self.money.setText("Money: $%s"%(str(self.var_money)))

            self.win.setText("Dealer Win!")


        elif (sum(cardValue[0]) >= (self.var_difficulty + 1) or sum(cardValue[1]) >
sum(cardValue[0])) and not sum(cardValue[1]) >= (self.var_difficulty + 1):

            self.var_money += self.var_bet

            self.money.setText("Money: $%s"%(str(self.var_money)))

            self.win.setText("You Win!")


        self.updateCheck()
```

```
 # Function initCheck() works at the very beginnning of the game. This is used for blackjack
games when the difficulty value is lower than 21.

 def initCheck(self):

     self.var_money = round(self.var_money, 2) # Turns 'var_money' into an decimal in case the
player's win contains less than $1.

     if (sum(cardValue[1]) >= (self.var_difficulty + 1)):

         self.var_money -= self.var_bet

         self.money.setText("Money: $%s"%(str(self.var_money)))

         self.win.setText("Dealer Win!")

         self.updateCheck()


 # Function updateCheck() is done after check() has been executed or when the game has
decided who wins the game. Function resets some variables and shuffles the card deck.

 def updateCheck(self):

     global win, cardValue, drawnCards, charlie # Function to access the global variable data
'win', 'drawnCards', 'cardValue', and 'charlie'.

     self.win.setStyleSheet(("background-color: white;"))

     shuffle(cardDeck) # Shuffles the 'cardDeck' array list.

     drawnCards, cardValue, self.oneOrTwoCards, win, self.drawn = [[], []], [[], []], -1, 0, 0 #
Resets back values and conditions of the blackjack game back from the start.

     self.resize()

     self.play_test = 2 # Sets 'play_test' = 2.
```

```
        self.enableBet() # Enable access to the 'btnB[]' buttons.

        self.totalGamesCount += 1 # Integer variable 'totalGamesCount' increments to  1.

        self.gameCount.setText("Total Games: %s"%(str(self.totalGamesCount)))  # Updates the
'gameCount' text element.




    # Function enableBet() to enables betting options by enabling the 'btnB' variable buttons
functionality.

    def enableBet(self):

        for i in range(0, len(self.btnB)):

            self.btnB[i].setEnabled(True)


    # Function restart() to restart the blackjack game.

    def restart(self):

        self.resize()

        self.play_test = 1 # Sets 'play_test' = 2.

        self.win.setText("")

        self.dealer.setText("")

        self.player.setText("")

        self.empty()

        self.dealer.setStyleSheet(("background-color: transparent;"))
        self.player.setStyleSheet(("background-color: transparent;"))

        self.win.setStyleSheet(("background-color: transparent;"))

        self.hitMe() # Starts the blackjack game.


    # Function reset() to reset the blackjack game back from the very beginning; resets the player
money to 100, resets the player's total play count back to 0.

    def reset(self):

        self.resize()

        self.play_test = 1

        self.oneOrTwoCards = 0

        self.win.setText("")

        self.dealer.setText("")

        self.player.setText("")

        self.dealer.setStyleSheet(("background-color: transparent;"))

        self.player.setStyleSheet(("background-color: transparent;"))

        self.win.setStyleSheet(("background-color: transparent;"))

        self.empty()

        self.var_money = 100

        self.money.setText("Money: $%s"%(str(self.var_money)))

        self.totalGamesCount = 0
        self.gameCount.setText("Total Games: %s"%(str(self.totalGamesCount)))

        self.enableBet()


    # Function stand() if the player decides to stand in their current card deck, but it's also used if
the player's card value sum is equals to 'var_difficulty' when it's player's turn.

    def stand(self):

        self.btnA[0].setEnabled(False)

        self.resize()

        speed = 100 # Declares 'speed' to add delay when the dealer is drawing their card.

        while True:

            if sum(cardValue[0]) <= 16 and sum(cardValue[1]) != 0 and sum(cardValue[1]) <=
(self.var_difficulty + 1):

                QTest.qWait(speed) # Set speed limit to add delay when the dealer is drawing their
cards.

                self.cardDraw(1, 1, 0, "Dealer",1 , 490)

                speed += 200

            else:

                self.resize()

                self.play_test = 2

                self.enableBet()

                self.check()
                self.btnA[0].setEnabled(True)

                break


# Global variables primarily used by the "Blackjack Functionality" section.

drawnCards = [[], []]

cardValue = [[], []]

cardDeck = []

ans = 6 # The total decks you want to play with.

shuffle(cardDeck) # Shuffles the 'cardDeck' array list.

win = 0

charlie = 0


# This is where all of the 52 playing cards are processed, all of them are compiled and stored in the 'cardDeck' array list.

# if shuffle(cardDeck) is called, the array list in 'cardDeck' is shuffled in random order.

for a in range(0, ans):

    for j in ("Hearts", "Diamonds", "Spades", "Clubs"):

        for i in ("2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"):

            cardDeck.append("%s of %s"%(i, j))
    #

    # END

    # C.) Blackjack Functionality

    #


# Initializes the entire program.

if __name__ == '__main__':

    app = QApplication(sys.argv)

    Window()

    sys.exit(app.exec_())
```

C. Work breakdown

Table 4

Work Breakdown

Student Name	**Tasks Assigned**	**Percentage of the Work Contribution**
Macaranas, Percival III Quinto	Background of the Study Problem Statement Objectives Methodology Conceptual Framework – IPO Chart (Input-Process-Output-Chart) Pseudocode GitHub Manager	30%
Narvaja, Shawn Karlsten Atacador	Table of Contents List of Figures List of Tables Significance of the Project Conceptual Framework – IPO Chart (Input-Process-Output-Chart) Results Discussion of Results User's Manual Source Code Trello Manager	30%

Table 5

Work Breakdown

Student Name	**Tasks Assigned**	**Percentage of the Work Contribution**
Ong, Franco Miguel Carlos	Coding Hierarchy Chart Flowchart Pseudocode Proofreader References	40%

D. Personal Data Sheet

Macaranas, Percival III Quinto

De La Salle University - Manila

Bachelor of Science - Computer Engineering

Narvaja, Shawn Karlsten Atacador

De La Salle University - Manila

Bachelor of Science - Computer Engineering

Ong, Franco Miguel Carlos

De La Salle University - Manila

Bachelor of Science - Computer Engineering