

Roulette

A game by Callum Laidlaw



Description

 Description

The project is a roulette game, that when the user spins the wheel (clicks on the wheel) the program will randomly choose which slot the “ball” would land on. From there, depending what the user placed a bet on, colour or number, they will be given a respected amount back on what they placed on that slot. For colours this is a 1.25x return, and for numbers it’s a 1.5x return. However, there is a chance the ball will land on a green slot which will have an increased return rate of 1.75 times. The way that the user is able to input all this information (their bet) is via a different screen which contains a text box for the amount they wish to enter, and then a lot of buttons that the user will click onto to lock in their bet (they are able to chance this by clicking on another option however) and so from there, once a bet has been placed the user is able to then spin the wheel.

 Description

The project is mainly Software Development but implements database to store user data, the way that user’s are able to access and then gain more money or take more money out the bank is done via an username, password account system that’s stored in the login table within the database, the username is the primary key with the password being encrypted to prevent data leaks and allow it so only the user who created the account will know the password. Within the game, fetching from a database constantly will be impractical, so user data is stored within arrays to make the program more efficient and quicker, and the roulette wheel is stored within a 2D array.

The goal of the game is to gain the most money possible, user’s are able to keep track and compare the amount they’ve got to others but viewing the leaderboard page. The method used to determine who is top and bottom, is insertion sort, then reversing the array for easy use within the program. The leaderboard

 Description

page will only show the top 10 users along with the amount of money each player has.

The project will use python and pygame to be able to create the interface and logic behind the game, and will contain an connection to external SQL Database to store information of users. Using this database, when a user logins to their account, they information is loaded from the database, stored in properties within the user class, and manipulated using these properties and once the user quits the game, the data is saved to the database.

My program will integrate a simple database which uses 3 tables to store login information, bank information, and money information of each player. During the program, a connection to the database is opened to allow SQL queries such as UPDATE, SELECT, CREATE to be carried out. The database connection will be closed at an appropriate point in the program.



Analysis

 Boundaries

The program will need to be able to valid user login information such as passwords and usernames from a database where the password is encrypted and it’s needed to be decrypted so the project is able to compare the username and password. The program will need to be able to validate which number the ball landed on and be able to compare it with the user predicted number and then provide the corresponding money to the user’s database index. The database information will not be imported into the game data until the user is logged in and then the appropriate user information will be imported and will allow the software to update the value of money. This data will be stored inside a 2D array due to allowing the program to easily append on user’s previous games, and also allow the software to easily update the money value for the user logged in.

 Scope

The program will be made using modular programming. The deliverables will include:

* Full analysis of the project
* Working project
* Test plan with appropriate test data
* Test evaluation
* Full evaluation report on the project

 Constraints

The project will have some constraints which include:

**Time**

* The project and all documentation will need to be completed and submitted before the SQA deadline.
* It will only be me working on the project, and due to my limited knowledge of python and pygame, the

 Constraints

speed at which the project will be made will be slower.

* I have 11 free periods, which means I am able to work on the project throughout these periods and create the documentation required for each section.

**Legal**

* Since all the graphics and program is created by myself, there will be no copyright complications arising.
* The game concept will be original due to allowing users to view other users progress and compare it to there own.

**Technical**

* The computer that the program is being made on already has python and pygame installed therefore saving me time with installation of the essential software.

 Constraints

* Currently I have access to already made snippets of standard algorithms allowing me to be able to save time when needing to use these standard algorithms.
* I will also need to spend time setting up the server the database will be created on however this will not take long to be able to allow me to connect to it and store relevant pieces of data.

**Economic**

* All software and programs used within this project will be provided for free so there won’t be able cost to creating the project.

 Requirements Specification

**End user requirements**

 Requirements Specification

* The user should be able to click on the roulette wheel
* The user should be able to click on different buttons to create a prediction on what slot they think the roulette wheel ball will land in
* The user should be given money back based on the prediction they made
* The user should be able to access the bank and view different loans and their interest rates
* The user should be able to view a leaderboard showing the top 10 players with the most amount of money with the richest being at the top
* The user should be able to register for an account with a username and password
* The user should be able to login to their account with their username and password
* The user should be able to easily view their current account balance
* The user should be able to log in and out of accounts

 Requirements Specification

**Functional Requirements**

**Inputs**

* + Enter username and password inside a text box and then login
  + Click on buttons on a page to make a prediction
  + Click on the roulette wheel to spin it
  + Click on a button to view the leaderboard
  + Click on a button to view the bank loans + interest rates
  + Click on buttons to take out a bank loan
  + Click on button to login to account
  + Click on button to view register page
  + Click on button to view login page
  + Click on button to register an account
  + Click on button to go back to the main roulette button from any page

**Processes**

* + Start game

 Requirements Specification

* + Prompt login view with register button and login button
  + Load user data from external database once user has logged in
  + Load image of roulette wheel onto screen
  + Load leaderboard button
  + Load bank button
  + Load bid button
  + If leaderboard button has been pressed, then load top 10 users and display on leaderboard page
  + If bank button has been pressed, then load all bank options and then display them onto the bank page
  + If a bank option button is pressed, then check if the user is valid to receive a bank loan. If yes then add the appropriate amount of money to the user’s account and then link the appropriate amount of interest based on the amount of money the user took

 Requirements Specification

* + If bid button is pressed, then load all bid options and display them with colours and numbers being separate with an amount text box at the top of the screen.
  + On each page, display a “cancel” button which when pressed, will take the user back to the roulette wheel
  + When roulette is clicked, run animation of roulette spinning, and then randomly choose a slot for the ball to land on
  + Compare what slot the ball landed on and what slot the user predicted the ball would land on, and if the same provide the appropriate amount of money
  + Detect if money has been won, if won then subtract money from the winnings to give to back to the bank
  + Update the bank amount taken to subtract the winnings from the roulette wheel

 Requirements Specification

* + Check if amount taken is 0, if 0 then remove the interest rate as the user has repaid their debts.
  + When user closes the game, save user data into the database, and then close the connection to database
  + Close game

Outputs

* + Display leaderboard
  + Display roulette wheel
  + Display login page
  + Display register page
  + Display bank page
  + Display bid page
  + Display bank options
  + Display bid options
  + Display leaderboard text
  + Display register button
  + Display login button
  + Display cancel button

 Requirements Specification

* + Display bid text box
  + Display username and password text box
  + Display “you won”, “you lost” text
  + Display user balance
  + Display user’s predicted slot

Project Planning

I’ve decided to use a chant chart to plan my project as I am able to easily view what I’m needing to create for each section of the project, which I cannot view in other planning methods.

Chart

Description automatically generated

UML Use Case Diagram

Diagram

Description automatically generated

Resources Required

* Windows > 10 OS
* Computer
* Sufficient memory for localhosted SQL Server
* USB Drive with store that allows database’s to be used
* Python >3.7
* Database visualer software
* Pyscripter with mySQL module and pygame module
* Microsoft excel, PowerPoint and Word
* Google chrome

 Advanced Higher Concepts Used

* 2d arrays of objects for the roulette wheel
* Use insertion sort to organize leaderboard data
* A project that surrounds software and database development.



Evaluation



Design

Insertion Sort with a 2D array

1. Loop through 2D array and store index as I
2. Set currentValue = array[I]
3. Set position = I
4. While position > 0 and array[position – 1][1] > currentValue[1] then
5. Set array[I] = array[position – 1]
6. Set position = position – 1
7. End while loop
8. Set array[position] = currentValue
9. End loop

Database Design: Data Dictionary

Table

Description automatically generated

Database Design: Data Dictionary

Table

Description automatically generated

Table

Description automatically generated

Database Design: Query Design

Table

Description automatically generated

Table

Description automatically generated

A picture containing table

Description automatically generated

Database Design: Query Design

Graphical user interface, text, application, email

Description automatically generated

Text, application

Description automatically generated with medium confidence

Table

Description automatically generated with medium confidence

Sprite Design

While working on the sprites, I had a few different ideas however decided on these sprites for the game as they looked the best:



Roulette Wheel



Top banner

Sprite Design



Register/Login Screen Register Button



Login Screen Login Button



Register Screen Cancel Button

Sprite Design



Close button used to close all menus



Bid button to open the bid menu



Bank button to open the bank menu

Sprite Design



Leaderboard button to open the leaderboard menu





The colour bid option used on the bid menu to select which colour to bid on









The number options used on the bed menu to select which number to bid on

Sprite Design







The bank options used on the bank menu to select how much a user wishes to withdraw

Background Design

For the background, it’s simply just a plain green colour to represent a green table which is used to play on.

Wireframes

Diagram

Description automatically generatedLogin Screen – displayed when the game is first launched

Wireframes

Register Screen – Display when Register button is pressed

Diagram

Description automatically generated

Wireframes

Main Screen – Displayed once an account has been logged into

Shape

Description automatically generated

Wireframes

Leaderboard – Displayed once Leaderboard button is clicked

Table

Description automatically generated

Wireframes

Bet – Displayed once the Bet Button is clicked

A picture containing diagram

Description automatically generated

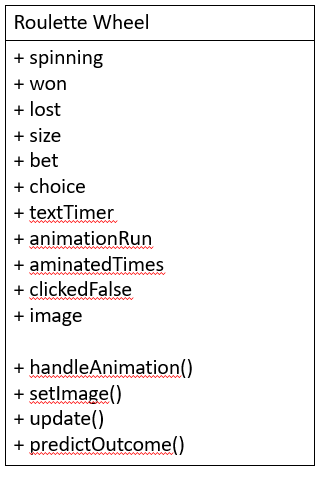
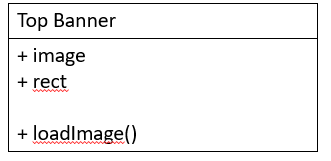
Wireframes

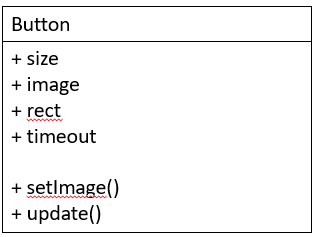
Bank – Displayed once the Bank Button is clicked

Table

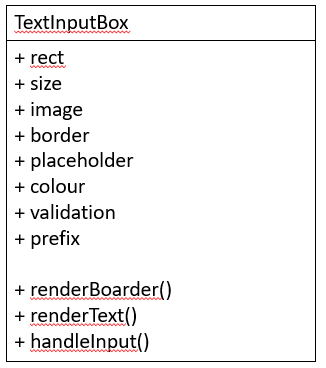
Description automatically generated

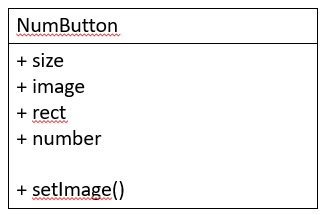
Class Diagram





Class Diagram









Implementation

Database Implementation

The connection and SQL for the database is all made possible using “mySQL”. The following are snippets of the code that use mySQL to interact with the database.

Text

Description automatically generatedConnection to database

Creating a cursor to the database



Database Implementation

Text

Description automatically generatedCreating tables if they do not exist

Inserting new users into the database

Database Implementation

Inserting new users into database

Text

Description automatically generated

Code – main.py

import pygame

import mysql.connector

from numpy import random

pygame.init() # Initalise pygame

# Connect to database

databaseConnnection = mysql.connector.connect(

    host="localhost",

    user="root",

    password="root",

    database="roulettegame"

)

database = databaseConnnection.cursor()

# Declare variables

bidMenuOpen = False

loginMenu = True

bankMenu = False

leaderboardMenu = False

registerMenu = False

gameRunning = False

bidMenuInvalidAmount = False

bidMenuInvalidTimer = 0

wonColourDelayTimer = 0

width = 750

height = 750

screen = pygame.display.set\_mode((width, height))

pygame.display.set\_caption("Roulette Game")

clock = pygame.time.Clock()

black =( 0, 0, 0 )

white = ( 255, 255, 255 )

# Declare User Class

class User:

    def \_\_init\_\_(self):

        self.username = False

        self.money = False

        self.unknown = False

        self.existing = False

        self.invalid = False

        self.registered = False

        self.bankInvalid = False

        self.bank = {

            "interest": 0,

            "amountTaken": 0,

            "subtract": 0

        }

        self.users = []

        self.collectAllUsers()

    def collectAllUsers(self):

        database.execute("SELECT username, money FROM data")

        self.users = database.fetchall()

        self.sortUsers()

    def setMoney(self, amount):

        self.money = round(amount)

    def addMoney(self, value):

        self.money += value

        self.money = round(self.money)

    def removeMoney(self, value):

        self.money -= value

        self.money = round(self.money)

    def login(self, username, password):

        database.execute("SELECT username, password FROM login WHERE username = \"" + str(username) + "\"")

        userInformation = database.fetchone()

        if userInformation == None:

            self.unknown = True

            return False

        elif password == userInformation[1]:

            database.execute("SELECT money FROM data WHERE username = \"" + str(username) + "\"")

            money = database.fetchone()

            database.execute(f"SELECT interest, amountTaken FROM bank WHERE username = \"{username}\"")

            bankDetails = database.fetchone()

            self.money = money[0]

            self.username = userInformation[0]

            self.bank["interest"] = bankDetails[0]

            self.bank["amountTaken"] = bankDetails[1]

            self.bank["subtract"] = self.bank["amountTaken"] - (self.bank["interest"] / 100)

            return True

        elif password != userInformation[1]:

            self.invalid = True

            return False

        return False

    def register(self, username, password):

        database.execute("SELECT username FROM login WHERE username = \"" + str(username) + "\"")

        userInformation = database.fetchone()

        if userInformation != None:

            self.existing = True

            return False

        database.execute(f"INSERT INTO login(username, password) VALUES (\"{username}\", \"{password}\")")

        database.execute(f"INSERT INTO data(username, money) VALUES (\"{username}\", 10000)")

        database.execute(f"INSERT INTO bank(username, interest, amountTaken) VALUES (\"{username}\", 0, 0)")

        databaseConnnection.commit()

        self.registered = True

        return True

    def saveMoney(self):

        if self.username != False:

            database.execute(f"UPDATE data SET money = \"{self.money}\" WHERE username = \"{self.username}\"")

            databaseConnnection.commit()

            return True

    def sortUsers(self):

        for i in range(1, len(self.users)):

            currentValue = self.users[i]

            position = i

            while position > 0 and self.users[position-1][1] > currentValue[1]:

                self.users[position] = self.users[position - 1]

                position -= 1

            self.users[position] = currentValue

        self.users.reverse()

    def payAmount(self, amount):

        self.bank["amountTaken"] = self.bank["amountTaken"] - amount

        if self.bank["amountTaken"] <= 0:

            self.bank["interest"] = 0

            self.bank["amountTaken"] = 0

        return self.bank["amountTaken"]

    def bankHandle(self, amount):

        interest = 10

        if amount >= 100 and amount <= 500:

            interest = 20

        elif amount >= 1000 and amount <= 5000:

            interest = 50

        elif amount == 50000:

            interest = 75

        if self.bank["interest"] <= .7:

            self.bank["amountTaken"] += amount

            self.bank["interest"] += interest / 100

            self.addMoney(amount)

            return True

        else:

            user.bankInvalid = True

            return True

user = User() # Declare user

# Declare RouletteSprite class

class RouletteSprite(pygame.sprite.Sprite):

    def \_\_init\_\_(self):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.size = [400, 400]

        self.image = pygame.Surface(self.size, pygame.SRCALPHA)

        self.rect = self.image.get\_rect()

        self.rect.x = 250 - 100

        self.rect.y = 260 - 50

        self.setImage()

        self.won = False

        self.lost = False

        self.animationRun = False

        self.animatedTimes = 0

        self.textTimer = 0

        self.clickedFalse = False

        self.possibilites = [

            [1,3,5,7,9,11,13,15,17,19],

            [2,4,6,8,10,12,14,16,18,20],

            [0]

        ]

        self.choice = {

            "colour": False,

            "number": False

        }

        self.bet = {

            "selection": False,

            "amount": False

        }

    def setImage(self):

        image\_notscaled = pygame.image.load("roulette.png")

        image = pygame.transform.scale(image\_notscaled, (self.size[0], self.size[1]))

        self.image.blit(image, (0,0))

    def update(self):

        if self.animationRun == True and self.animatedTimes < 50 or self.animatedTimes == 51:

            self.handleAnimation()

        if self.animatedTimes == 50:

            self.animatedTimes += 1

            self.setImage()

            self.animationRun = False

            self.predictOutcome()

        if self.choice["colour"] == self.bet["selection"] and self.choice["colour"] != False:

            if self.textTimer > 100:

                self.won = False

                self.textTimer = 0

            else:

                if self.won == False:

                    self.won = True

                    if self.choice["colour"] == "GREEN":

                        if user.bank["amountTaken"] > 0:

                            amount = self.bet["amount"]

                            winnings = amount \* 1.75

                            toBank = winnings \* user.bank["interest"]

                            user.addMoney((winnings - toBank))

                            user.payAmount(toBank)

                        else:

                            amount = self.bet["amount"]

                            user.addMoney(amount \* 1.75)

                    else:

                        if user.bank["amountTaken"] > 0:

                            amount = self.bet["amount"]

                            winnings = amount \* 1.25

                            toBank = winnings \* user.bank["interest"]

                            user.addMoney((winnings - toBank))

                            user.payAmount(toBank)

                        else:

                            amount = self.bet["amount"]

                            user.addMoney(amount \* 1.25)

                    self.choice["colour"] = False

                    self.choice["number"] = False

                    self.bet["selection"] = False

                    self.bet["amount"] = False

        elif str(self.choice["number"]) == str(self.bet["selection"]) and self.choice["number"] != False:

            if self.textTimer > 100:

                self.won = False

                self.textTimer = 0

            else:

                if self.won == False:

                    self.won = True

                    if user.bank["amountTaken"] > 0:

                        amount = self.bet["amount"]

                        winnings = amount \* 1.25

                        toBank = winnings \* user.bank["interest"]

                        user.addMoney((winnings - toBank))

                        user.payAmount(toBank)

                    else:

                        amount = self.bet["amount"]

                        user.addMoney(amount \* 1.25)

                    self.choice["number"] = False

                    self.choice["colour"] = False

                    self.bet["selection"] = False

                    self.bet["amount"] = False

    def handleAnimation(self):

        if self.animatedTimes == 151:

            self.animatedTimes = 0

        orRect = self.image.get\_rect()

        image = pygame.transform.rotate(self.image, ((20 + 45) % 360))

        rect = orRect.copy()

        rect.center = image.get\_rect().center

        image = image.subsurface(rect).copy()

        self.image = image

        self.animatedTimes += 1

    def predictOutcome(self):

        colour = random.randint(0, 3)

        if colour == 2:

            self.choice["colour"] = "GREEN"

            self.choice["number"] = 0

            if self.choice["colour"] != self.bet["selection"]:

                self.choice["colour"] = False

                self.choice["number"] = False

                self.lost = True

                user.removeMoney(self.bet["amount"])

            return

        number = random.randint(0, 9)

        if colour == 0:

            self.choice["colour"] = "BLACK"

            self.choice["number"] = self.possibilites[colour][number]

        elif colour == 1:

            self.choice["colour"] = "RED"

            self.choice["number"] = self.possibilites[colour][number]

        if self.choice["colour"] != self.bet["selection"] and self.bet["selection"] != self.choice["number"] and self.choice["colour"] != False and self.bet["selection"] != False:

            self.choice["colour"] = False

            self.choice["number"] = False

            self.bet["selection"] = False

            self.lost = True

            user.removeMoney(self.bet["amount"])

        if self.choice["number"] != self.bet["selection"] and self.bet["selection"] != self.choice["colour"] and self.choice["number"] != False and self.bet["selection"] != False:

            self.choice["colour"] = False

            self.choice["number"] = False

            self.bet["selection"] = False

            self.lost = True

            user.removeMoney(self.bet["amount"])

class TopBanner(pygame.sprite.Sprite):

    def \_\_init\_\_(self):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.image = pygame.Surface([750, 200], pygame.SRCALPHA)

        self.rect = self.image.get\_rect()

        self.rect.y = 0

        self.rect.x = 0

        self.loadImage()

    def loadImage(self):

        image\_notscaled = pygame.image.load("topBanner.png")

        image = pygame.transform.scale(image\_notscaled, (750, 200))

        self.image.blit(image, (0, 0))

# Declare Button class

class Button(pygame.sprite.Sprite):

    def \_\_init\_\_(self, size, x, y, image):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.size = size

        self.image = pygame.Surface(size, pygame.SRCALPHA)

        self.rect = self.image.get\_rect()

        self.rect.x = x

        self.rect.y = y

        self.image.fill(pygame.Color(255, 223, 0))

        self.setImage(image)

        self.timeout = False

    def setImage(self, image):

        image\_notscaled = pygame.image.load(image)

        image = pygame.transform.scale(image\_notscaled, (self.size[0], self.size[1]))

        self.image.blit(image, (0,0))

    def update(self):

        if self.timeout == True:

            pygame.time.wait(100)

            self.timeout = False

class NumButton(pygame.sprite.Sprite):

    def \_\_init\_\_(self, size, x, y, image, number):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.size = size

        self.image = pygame.Surface(size, pygame.SRCALPHA)

        self.rect = self.image.get\_rect()

        self.rect.x = x

        self.rect.y = y

        self.image.fill(pygame.Color(255, 223, 0))

        self.setImage(image)

        self.number = number

    def setImage(self, image):

        image\_notscaled = pygame.image.load(image)

        image = pygame.transform.scale(image\_notscaled, (self.size[0], self.size[1]))

        self.image.blit(image, (0,0))

class TextInputBox(pygame.sprite.Sprite):

    def \_\_init\_\_(self, size, x, y, border, placeholder, validations = None, prefix = None):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.size = size

        self.image = pygame.Surface(size, pygame.SRCALPHA)

        self.rect = self.image.get\_rect()

        self.rect.x = x

        self.rect.y = y

        self.border = border

        self.active = False

        self.text = placeholder

        self.colour = black

        self.placeholder = placeholder

        self.validation = validations

        self.prefix = prefix

        self.image.fill((252,164,12))

        self.renderText()

    def renderBorder(self,):

        pygame.draw.rect(self.image, self.border, self.image.get\_rect().inflate(2, 2), 7)

    def renderText(self):

        font = pygame.font.SysFont("vgasys", 100)

        text\_rendered = font.render(self.text, True, self.colour)

        if self.prefix != None:

            text\_rendered = font.render(self.prefix + self.text, True, self.colour)

        self.image = pygame.Surface((self.image.get\_rect().width, text\_rendered.get\_height()+10), pygame.SRCALPHA)

        self.image.fill((252,164,12))

        self.image.blit(text\_rendered, (10, 5))

        self.renderBorder()

    def resetText(self):

        self.text = self.placeholder

        self.renderText()

    def handleInput(self, event):

        if event.type == pygame.KEYDOWN and self.active == True:

                if event.key == pygame.K\_RETURN:

                    self.active = False

                    if self.text == "":

                        self.text = self.placeholder

                elif event.key == pygame.K\_BACKSPACE:

                    self.text = self.text[:-1]

                else:

                    if self.validation == "numbers":

                        if not event.unicode.isnumeric():

                            return

                    self.text += str(event.unicode)

                self.renderText()

        elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

                pos = pygame.mouse.get\_pos()

                if self.rect.collidepoint(pos):

                    if self.active == False:

                        self.active = True

                        self.text = ""

                        self.renderText()

                else:

                    self.active = False

                    if self.text == "":

                        self.text = self.placeholder

                        self.renderText()

# Declare Sprites

gameSprites = pygame.sprite.Group()

bidMenuSprites = pygame.sprite.Group()

bidMenuNumbers = pygame.sprite.Group()

loginSprites = pygame.sprite.Group()

registerSprites = pygame.sprite.Group()

bankSprites = pygame.sprite.Group()

leaderboardSprites = pygame.sprite.Group()

roulette = RouletteSprite()

topBanner = TopBanner()

usernameLogin = TextInputBox([700, 75], 25, 250, black, "Username")

passwordLogin = TextInputBox([700, 75], 25, 350, black, "Password")

loginButton = Button([200, 100], 150, 450, "loginButton.png")

registerButtonLogin = Button([200, 100], 400, 450, "registerButton.png")

usernameRegister = TextInputBox([700, 75], 25, 250, black, "Username")

passwordRegister = TextInputBox([700, 75], 25, 350, black, "Password")

registerButton = Button([200, 100], 150, 450, "registerButton.png")

cancelButtonRegister = Button([200, 100], 400, 450, "cancelButton.png")

bidButton = Button([200, 100], 550, 650, "bidButton.png")

bankButton = Button([200, 100], 340, 650, "bankButton.png")

leaderboardButton = Button([300, 100], 0, 650, "leaderboardButton.png")

closeBidMenu = Button([200, 100], 550, 0, "closeButton.png")

greenBid = Button([200, 100], 40, 320, "greenBet.png")

redBid = Button([200, 100], 280, 320, "redBet.png")

blackBid = Button([200, 100], 520, 320, "blackBet.png")

bidAmountBox = TextInputBox([700, 75], 25, 150, black, "Amount", "numbers", "$")

closeBankMenu = Button([200, 100], 550, 0, "closeButton.png")

TenOp = Button([150, 75], 60, 200, "bankOptions/10.png")

TwenOp = Button([150, 75], 220, 200, "bankOptions/20.png")

ThriOp = Button([150, 75], 380, 200, "bankOptions/30.png")

FivOp = Button([150, 75], 540, 200, "bankOptions/50.png")

HunOp = Button([150, 75], 60, 350, "bankOptions/100.png")

THunOp = Button([150, 75], 220, 350, "bankOptions/200.png")

ThHunOp = Button([150, 75], 380, 350, "bankOptions/300.png")

FHunOp = Button([150, 75], 540, 350, "bankOptions/500.png")

ThoOp = Button([200, 100], 60, 500, "bankOptions/1000.png")

TThoOp = Button([200, 100], 280, 500, "bankOptions/2000.png")

FThoOp = Button([200, 100], 500, 500, "bankOptions/5000.png")

FiThoOp = Button([250, 100], 250, 630, "bankOptions/50000.png")

closeLeaderboardMenu = Button([200, 100], 550, 0, "closeButton.png")

n1 = NumButton([50,50], 50 + 53, 500, "numbers/1.png", 1)

n2 = NumButton([50,50], 50 + (2 \* 53), 500, "numbers/2.png", 2)

n3 = NumButton([50,50], 50 + (3 \* 53), 500, "numbers/3.png", 3)

n4 = NumButton([50,50], 50 + (4 \* 53), 500, "numbers/4.png", 4)

n5 = NumButton([50,50], 50 + (5 \* 53), 500, "numbers/5.png", 5)

n6 = NumButton([50,50], 50 + (6 \* 53), 500, "numbers/6.png", 6)

n7 = NumButton([50,50], 50 + (7 \* 53), 500, "numbers/7.png", 7)

n8 = NumButton([50,50], 50 + (8 \* 53), 500, "numbers/8.png", 8)

n9 = NumButton([50,50], 50 + (9 \* 53), 500, "numbers/9.png", 9)

n10 = NumButton([50,50], 50 + (10 \* 53), 500, "numbers/10.png", 10)

n11 = NumButton([50,50], 50 + (1 \* 53), 575, "numbers/11.png", 11)

n12 = NumButton([50,50], 50 + (2 \* 53), 575, "numbers/12.png", 12)

n13 = NumButton([50,50], 50 + (3 \* 53), 575, "numbers/13.png", 13)

n14 = NumButton([50,50], 50 + (4 \* 53), 575, "numbers/14.png", 14)

n15 = NumButton([50,50], 50 + (5 \* 53), 575, "numbers/15.png", 15)

n16 = NumButton([50,50], 50 + (6 \* 53), 575, "numbers/16.png", 16)

n17 = NumButton([50,50], 50 + (7 \* 53), 575, "numbers/17.png", 17)

n18 = NumButton([50,50], 50 + (8 \* 53), 575, "numbers/18.png", 18)

n19 = NumButton([50,50], 50 + (9 \* 53), 575, "numbers/19.png", 19)

n20 = NumButton([50,50], 50 + (10 \* 53), 575, "numbers/20.png", 20)

gameSprites.add(bidButton)

gameSprites.add(roulette)

gameSprites.add(bankButton)

gameSprites.add(leaderboardButton)

gameSprites.add(topBanner)

bidMenuSprites.add(blackBid)

bidMenuSprites.add(redBid)

bidMenuSprites.add(greenBid)

bidMenuSprites.add(closeBidMenu)

bidMenuSprites.add(bidAmountBox)

bidMenuSprites.add(n1)

bidMenuSprites.add(n2)

bidMenuSprites.add(n3)

bidMenuSprites.add(n4)

bidMenuSprites.add(n5)

bidMenuSprites.add(n6)

bidMenuSprites.add(n7)

bidMenuSprites.add(n8)

bidMenuSprites.add(n9)

bidMenuSprites.add(n10)

bidMenuSprites.add(n11)

bidMenuSprites.add(n12)

bidMenuSprites.add(n13)

bidMenuSprites.add(n14)

bidMenuSprites.add(n15)

bidMenuSprites.add(n16)

bidMenuSprites.add(n17)

bidMenuSprites.add(n18)

bidMenuSprites.add(n19)

bidMenuSprites.add(n20)

loginSprites.add(usernameLogin)

loginSprites.add(passwordLogin)

loginSprites.add(loginButton)

loginSprites.add(registerButtonLogin)

registerSprites.add(usernameRegister)

registerSprites.add(passwordRegister)

registerSprites.add(registerButton)

registerSprites.add(cancelButtonRegister)

bankSprites.add(closeBankMenu)

bankSprites.add(TenOp)

bankSprites.add(TwenOp)

bankSprites.add(ThriOp)

bankSprites.add(FivOp)

bankSprites.add(HunOp)

bankSprites.add(THunOp)

bankSprites.add(ThHunOp)

bankSprites.add(FHunOp)

bankSprites.add(ThoOp)

bankSprites.add(TThoOp)

bankSprites.add(FThoOp)

bankSprites.add(FiThoOp)

leaderboardSprites.add(closeLeaderboardMenu)

# Game loop

running = True

while running:

    animationRun = False

    for event in pygame.event.get():

        if loginMenu == True:

            passwordLogin.handleInput(event)

            usernameLogin.handleInput(event)

        elif registerMenu == True:

            usernameRegister.handleInput(event)

            passwordRegister.handleInput(event)

        elif bidMenuOpen == True:

            bidAmountBox.handleInput(event)

        if event.type == pygame.QUIT or event.type == pygame.KEYDOWN and event.key == pygame.K\_ESCAPE:

            user.saveMoney()

            running = False

        elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            pos = pygame.mouse.get\_pos()

            if roulette.rect.collidepoint(pos) and bidMenuOpen == False and bankMenu == False and leaderboardMenu == False and gameRunning == True:

                if roulette.bet["selection"] != False and roulette.textTimer == 0:

                    roulette.animationRun = True

                else:

                    roulette.clickedFalse = True

            if bidButton.rect.collidepoint(pos) and gameRunning == True and bidMenuOpen == False and bankMenu == False and leaderboardMenu == False:

                bidMenuOpen = True

            if bankButton.rect.collidepoint(pos) and gameRunning == True and bidMenuOpen == False and leaderboardMenu == False:

                bankMenu = True

            if leaderboardButton.rect.collidepoint(pos) and gameRunning == True and bidMenuOpen == False and bidMenuOpen == False:

                leaderboardMenu = True

            if closeBidMenu.rect.collidepoint(pos) and bidMenuOpen == True:

                bidMenuOpen = False

                bidAmountBox.text = f"{bidAmountBox.prefix} {bidAmountBox.placeholder}"

            if greenBid.rect.collidepoint(pos) and bidMenuOpen == True:

                if bidAmountBox.placeholder in bidAmountBox.text or int(bidAmountBox.text) > user.money:

                    bidMenuInvalidAmount = True

                else:

                    roulette.bet["selection"] = "GREEN"

                    roulette.bet["amount"] = int(bidAmountBox.text)

                    bidMenuOpen = False

            if redBid.rect.collidepoint(pos) and bidMenuOpen == True:

                if bidAmountBox.placeholder in bidAmountBox.text or int(bidAmountBox.text) > user.money:

                    bidMenuInvalidAmount = True

                else:

                    roulette.bet["selection"] = "RED"

                    roulette.bet["amount"] = int(bidAmountBox.text)

                    bidMenuOpen = False

            if blackBid.rect.collidepoint(pos) and bidMenuOpen == True:

                if bidAmountBox.placeholder in bidAmountBox.text or int(bidAmountBox.text) > user.money:

                    bidMenuInvalidAmount = True

                else:

                    roulette.bet["selection"] = "BLACK"

                    roulette.bet["amount"] = int(bidAmountBox.text)

                    bidMenuOpen = False

            if closeLeaderboardMenu.rect.collidepoint(pos) and leaderboardMenu == True:

                leaderboardMenu = False

            if closeBankMenu.rect.collidepoint(pos) and bankMenu == True:

                bankMenu = False

                gameRunning = True

            if registerButtonLogin.rect.collidepoint(pos) and cancelButtonRegister.rect.collidepoint(pos):

                if loginMenu == True and gameRunning == False:

                    loginMenu = False

                    registerMenu = True

                    usernameLogin.resetText()

                    passwordLogin.resetText()

                    user.unknown = False

                    user.invalid = False

                elif loginMenu == False and gameRunning == False:

                    loginMenu = True

                    registerMenu = False

                    usernameRegister.resetText()

                    passwordRegister.resetText()

                    user.registered = False

                    user.existing = False

            if loginButton.rect.collidepoint(pos) and loginMenu == True:

                logged = user.login(usernameLogin.text, passwordLogin.text)

                if logged != False:

                    loginMenu = False

                    gameRunning = True

            if registerButton.rect.collidepoint(pos) and registerMenu == True:

                user.register(usernameRegister.text, passwordRegister.text)

            if TenOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(10)

                bankMenu = False

                gameRunning = True

            if TwenOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(20)

                bankMenu = False

                gameRunning = True

            if ThriOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(30)

                bankMenu = False

                gameRunning = True

            if FivOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(50)

                bankMenu = False

                gameRunning = True

            if HunOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(100)

                bankMenu = False

                gameRunning = True

            if THunOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(200)

                bankMenu = False

                gameRunning = True

            if ThHunOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(300)

                bankMenu = False

                gameRunning = True

            if FHunOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(500)

                bankMenu = False

                gameRunning = True

            if ThoOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(1000)

                bankMenu = False

                gameRunning = True

            if TThoOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(2000)

                bankMenu = False

                gameRunning = True

            if FThoOp.rect.collidepoint(pos) and bankMenu == True:

                user.bankHandle(5000)

                bankMenu = False

                gameRunning = True

            if FiThoOp.rect.collidepoint(pos):

                if bankMenu == True and gameRunning == False:

                    bankMenu = False

                    gameRunning = True

                    user.bankHandle(50000)

                elif gameRunning == True and bankButton.rect.collidepoint(pos):

                    gameRunning = False

                    bankMenu = True

            for number in bidMenuNumbers:

                if number.rect.collidepoint(pos) and bidMenuOpen == True:

                    if bidAmountBox.placeholder in bidAmountBox.text or int(bidAmountBox.text) > user.money:

                        bidMenuInvalidAmount = True

                    else:

                        roulette.bet["selection"] = str(number.number)

                        roulette.bet["amount"] = int(bidAmountBox.text)

                        bidMenuOpen = False

        elif event.type == pygame.KEYDOWN and event.key == pygame.K\_l:

            variable = input("Enter the variable you wish to see: ")

            variables = {

                "gameRunning": gameRunning,

                "bankMenu": bankMenu,

                "bidMenuOpen": bidMenuOpen,

                "registerMenu": registerMenu,

                "leaderboardMenu": leaderboardMenu

            }

            print(variables[variable])

    if loginMenu == True:

        screen.fill((0, 128, 0))

        font = pygame.font.SysFont("vgasus", 120)

        loginTitle = font.render("Login", True, black)

        unknownUser = font.render("Unknown User", True, (255, 0, 0))

        invalidPass = font.render("Invalid Password", True, (255, 0, 0))

        if user.unknown:

            screen.blit(unknownUser, (75, 600))

        elif user.invalid:

            screen.blit(invalidPass, (50, 600))

        screen.blit(loginTitle, (280, 90))

        loginSprites.draw(screen)

        loginSprites.update()

    elif registerMenu == True:

        screen.fill((0, 128, 0))

        font = pygame.font.SysFont("vgasus", 120)

        registerTitle = font.render("Register", True, black)

        existingUser = font.render("Alright registered", True, (255, 0, 0))

        font = pygame.font.SysFont("vgasus", 95)

        succReg = font.render("Registerd Succesfully", True, (0, 255, 0))

        if user.existing:

            screen.blit(existingUser, (25, 600))

        elif user.registered:

            screen.blit(succReg, (25, 600))

        screen.blit(registerTitle, (250, 90))

        registerSprites.draw(screen)

        registerSprites.update()

    elif bankMenu == True:

        screen.fill((0, 128, 0))

        titleFont = pygame.font.SysFont("vgasus", 100)

        titleText = titleFont.render("Bank", True, black)

        optionFont = pygame.font.SysFont(None, 70)

        optionSmall = optionFont.render("Small Interest", True, black)

        optionMedium = optionFont.render("Regular Interest", True, black)

        optionLarge = optionFont.render("Large Interest", True, black)

        screen.blit(titleText, (120, 25))

        screen.blit(optionSmall, (220, 140))

        screen.blit(optionMedium, (200, 290))

        screen.blit(optionLarge, (200, 440))

        bankSprites.draw(screen)

        bankSprites.update()

    elif leaderboardMenu == True:

        screen.fill((0, 128, 0))

        titleFont = pygame.font.SysFont("vgasus", 100)

        titleText = titleFont.render("Leaderboards", True, black)

        leaderboardFont = pygame.font.SysFont(None, 50)

        screen.blit(titleText, (5, 25))

        users = user.users

        length = 10

        if len(users) < 10:

            length = len(users)

        for i in range(0, length):

            leaderboardText = leaderboardFont.render(f"{i + 1}. {users[i][0]} >> {users[i][1]}", True, black)

            screen.blit(leaderboardText, (0, (110 + (i \* 50))))

        leaderboardSprites.draw(screen)

        leaderboardSprites.update()

    elif bidMenuOpen == True:

        screen.fill((0, 128, 0))

        font = pygame.font.SysFont("vgasys", 48)

        menuTitle = font.render("Bid Menu", True, black)

        colourOption = font.render("Bid on Colour", True, black)

        numberOption = font.render("Bid on Number", True, black)

        invalidAmount = font.render("Invalid Amount", True, (255, 0, 0))

        screen.blit(menuTitle, (280, 25))

        screen.blit(colourOption, (250, 250))

        screen.blit(numberOption, (250, 450))

        if bidMenuInvalidAmount == True:

            if bidMenuInvalidTimer >= 100:

                bidMenuInvalidAmount = False

                bidMenuInvalidTimer = 0

            screen.blit(invalidAmount, (250, 700))

            bidMenuInvalidTimer += 1

        bidMenuSprites.draw(screen)

        bidMenuNumbers.draw(screen)

        bidMenuSprites.update()

        bidMenuNumbers.update()

    else:

        screen.fill((0, 128 ,0))

        font = pygame.font.SysFont(None, 100)

        winFont = pygame.font.SysFont(None, 170)

        gameSprites.draw(screen)

        gameSprites.update()

        if roulette.bet["selection"] == False:

            betText = font.render("None", True, black)

            screen.blit(betText, (270, 115))

        else:

            betText = font.render(str(roulette.bet["selection"]), True, black)

            screen.blit(betText, (270, 110))

        if roulette.textTimer >= 100:

            if roulette.won: roulette.won = False

            if roulette.lost: roulette.lost = False

            if roulette.clickedFalse: roulette.clickedFalse = False

            if user.bankInvalid: user.bankInvalid = False

            roulette.textTimer = 0

        if roulette.won:

            if wonColourDelayTimer < 5:

                winText = winFont.render("You won!", True, (255, 0, 0))

                screen.blit(winText, (120, 350))

            if wonColourDelayTimer >= 5 and wonColourDelayTimer < 10:

                winText = winFont.render("You won!", True, (0, 255, 0))

                screen.blit(winText, (120, 350))

            if wonColourDelayTimer >= 10 and wonColourDelayTimer < 15:

                winText = winFont.render("You won!", True, (0, 0, 255))

                screen.blit(winText, (120, 350))

                wonColourDelayTimer = 0

            wonColourDelayTimer += 1

            roulette.textTimer += 1

        elif wonColourDelayTimer > 0 and roulette.won == False:

            wonColourDelayTimer = 0

        elif roulette.lost == True:

            lostText = winFont.render("You lost!", True, (255, 0, 0))

            screen.blit(lostText, (120, 350))

            roulette.textTimer += 1

        elif roulette.clickedFalse:

            placeBetFont = pygame.font.SysFont(None, 80)

            placeBetText = placeBetFont.render("Place a bet before spinning", True, (222, 0, 0))

            screen.blit(placeBetText, (10, 350))

            roulette.textTimer += 1

        elif user.bankInvalid:

            invalidBank = pygame.font.SysFont(None, 70)

            invalidText = invalidBank.render("Cannot take any more money!", True, (222, 0,0))

            screen.blit(invalidText, (15, 350))

            roulette.textTimer += 1

        moneyFont = pygame.font.SysFont(None, 60)

        moneyText = moneyFont.render(f"${user.money}", True, black)

        screen.blit(moneyText, (370, 30))

    if user.money < 0:

        user.setMoney(0)

    pygame.display.update()

    clock.tick(60)

pygame.quit()



Testing

Testing

Integrative Testing

There was issues when reading data in from the database, due to how the connection will originally made, however after researching the issues a bit, I was able to find another method to read data in which is more efficient.

Accessibility Testing

Due to myself being dyslexic, I was able to determine if it was easy to read, and if there was anything I could’ve added to make it more readable for people with dyslexia, and I’ve concluded that the program is readable for dyslexic people and the game works in a way so it’s not needing to be heavily reliant on reading information.

Component Testing

After making each component of the games, for example the sprites, I’d always test them out to make sure they worked, and always catch and fix and issues that arise from them. Here is a few examples:

* Roulette Wheel Spin Animation

The original issue with this was that it was taking up too much memory and causing the game to crash due to how the animation was originally made, which when the wheel animation was activated, the wheel would make thousands of copies of itself spinning around, and now different surfaces are made for the wheel sprite to be rotated and put onto which saves a lot more memory and causing the program not to crash while still having the same effect.

Component Testing

* Button Location

Certain buttons would have an error where when clicked, a button on another screen in the same location would be pressed causing a the issue where both buttons are pressed even when 1 was intended to be pressed. The way to fix this issue was to check which menu screen the button was pressed on, if it was screen X and button X was pressed, only button X would be detected, but if it was screen Y and button Y was pressed, only button X would be detected.

* Leaderboard Sort

There was an error with the leaderboard where the sorting wouldn’t work, and under further investigation the reverse the Array was causing this issue and to fix it, I did some research and figured out that python has a premade function to reverse arrays.

Database Testing

Graphical user interface, application

Description automatically generatedAs shown by the screenshots below the database is successfully able to take in and store data written in from the program

Graphical user interface, application

Description automatically generated

A picture containing graphical user interface

Description automatically generated



Evaluation

Evaluation of Solution

Fitness for purpose

* Looking at functional and end-user requirements stated at the beginning then the program meets most requirements. One requirement it’s not meeting is passwords aren’t encrypted due to software issues with the school computers.

Robustness

* The program doesn’t often crash, and if it crashes it’s due to the database not being online, apart from that the program and able to handle all input errors.

Maintainability

* The program is not maintainable due to the poor readability of the code, there isn’t comments explaining different functions, however it is laid out so most things separated.

Evaluation of Solution

The program I’ve made is achieves the original goal of being a game that works as intended and is easy for user usage, although there is some security aspects that I would like changed, however technology and software issues hasn’t allowed that. The program is able to save user data to database, user’s are able to log into their own accounts, user’s are able to take money out at a bank and have interest amounts added to their accounts which takes money of their winnings.

The program is bug free, however I predict under commercial use the program will generate more bugs that will need to be addressed and fixed.

Overall I’m happy with the program and I am pleased with the overall final product reflecting on all the issues and programs I’ve had to overcome while making the program, and if I was to remake the code, I’d make the code more readable and thus making maintainability better in the process.