A Mini-Project Report on

Gaming café Database Management

Submitted in 5th Semester

Bachelor of Engineering

In

Computer Science and Engineering

Submitted by

Name: Mohammed Ataaur Rahaman [1DS16CS721]

Name : Nishant Ranjan [1DS16CS724]

Under the guidance of

PROF. Anupama VP

Dept. of CSE, DSCE

2018-2019

Department of Computer Science and Engineering,

DAYANANDA SAGAR COLLEGE OF ENGINEERING

BANGALORE – 560078

Abstract

The Problem

- There are lot of gaming Cafes, but not many have an efficient way to manage its Gamers using a single Software.

The Solution

- A software application which helps in storing all the information of the gamers playing in the café.
- The café manager can store all its gamers profile in the Database management system built into the Gaming café application.

Implementation

- Application – tkinter (python)
- Database – sqlite3

Table of Contents

Introduction

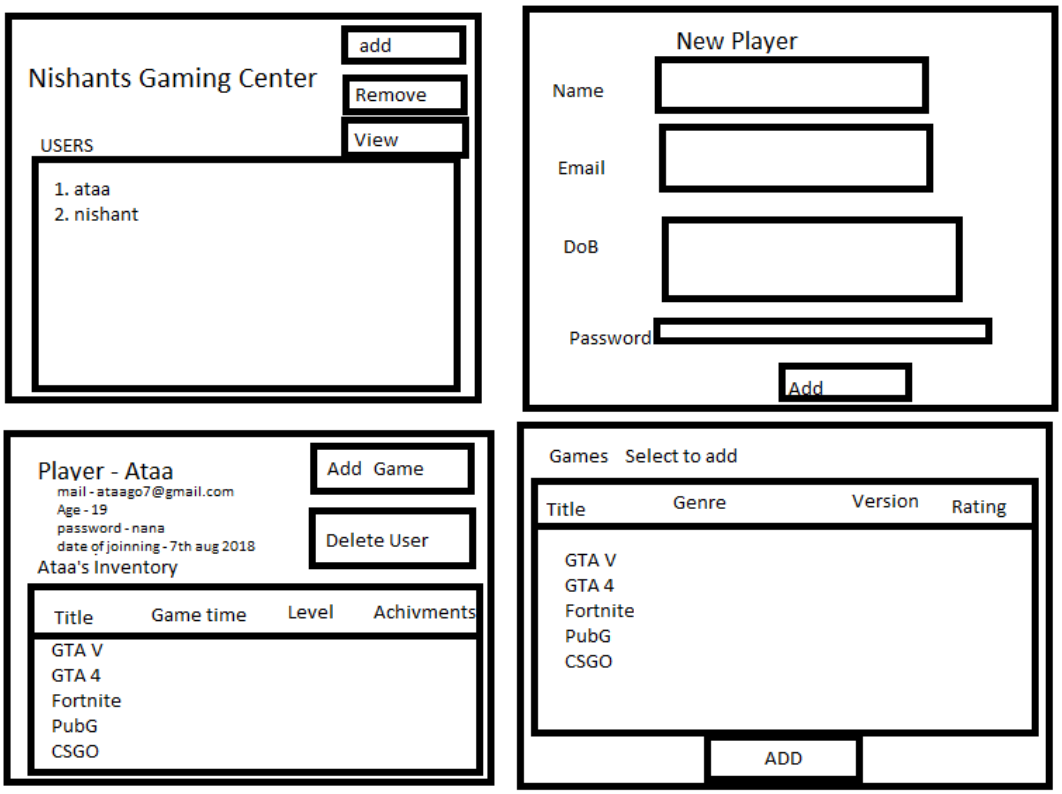
The Mini-Project aims to create a Database Management System for a Gaming Café to efficiently manage their gamers.

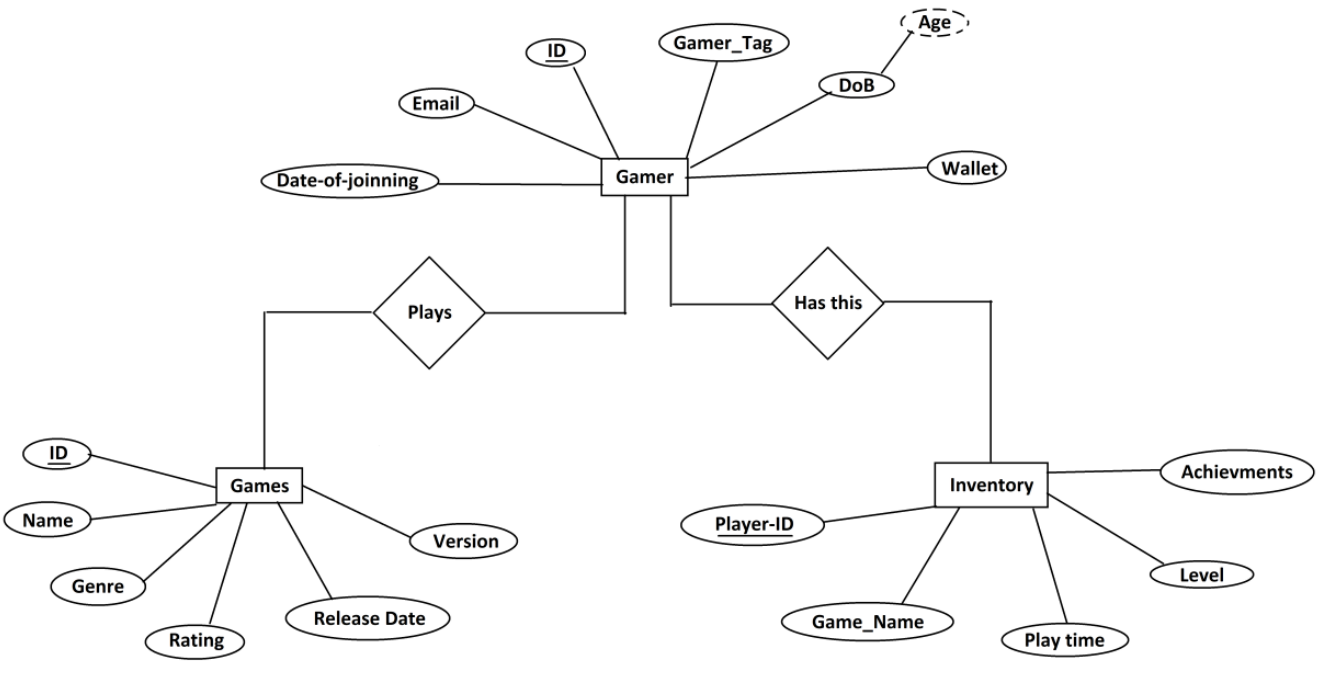
Features

- The application for the gaming café helps the manager to store its gamers data like, name, email, time spent, password etc.
- The application is connected to other client computers the gamer can login to his/her PC using the ‘gamer_tag’ and ‘password’ which is - managed from the application.
- The manager can keep track of the gamers time spent on the machine to charge accordingly.
- The manager can also keep track of the games being played by individual gamers

Design

- The database has tables for Gamer, Games and Inventory.
- The relations for the tables are:
  - Gamer play Games
  - Gamer has this Inventory





Implementation (source Code)

Link to GitHub: https://github.com/Ataago/Gaming-Cafe-DBMS

DataBase.py

```
Import sqlite3

data_base_name = "MyGameCafe_V2.db"

'''This is the Data Base for the gaming center'''
def connect():
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    '''
    cur.execute("drop table test")
    cur.execute("create table if not exists test(id integer)")
    cur.execute("insert into test values('asdfas')")
    conn.commit()
    cur.execute("delete from owner where username = ''")
    print('Deleted')
    rows = cur.fetchall()
    print(rows)
    '''
    #cur.execute("DROP TABLE IF EXISTS owner")
    cur.execute("CREATE TABLE IF NOT EXISTS owner(username TEXT NOT NULL, password TEXt NOT NULL,
UNIQUE(username));")

    #cur.execute("DROP TABLE if exists gamer")
    cur.execute("CREATE TABLE IF NOT EXISTS gamer(id INTEGER PRIMARY KEY , gamer_name TEXT, email TEXT,
gamer_tag TEXT, age INTEGER, UNIQUE (email))")

    #cur.execute("DROP TABLE if exists games")
    cur.execute("CREATE TABLE IF NOT EXISTS games(game_id INTEGER PRIMARY KEY , game_name TEXT, genre TEXT,
release_date DATE, Version INTEGER, rating INTEGER )")

    #cur.execute("DROP TABLE if exists inventory")
    cur.execute("CREATE TABLE IF NOT EXISTS inventory(game_ID INTEGER, gamer_ID INTEGER, game_name TEXT,
play_time INTEGER, achievments TEXT, PRIMARY KEY(game_ID, gamer_ID))")
    conn.commit
    conn.close()

def view(table_name):
    """Returns all the tuples in any give table_name"""
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("SELECT * FROM %s" % table_name)
```

```
    rows = cur.fetchall()
    conn.close()
    return rows

def view_owner(username):
    """returns the tuple with username and password"""
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("SELECT * FROM owner WHERE username = '%s' " % (username))
    rows = cur.fetchall()
    conn.close()
    return rows

def view_gamer(gamer_ID):
    """returns the tuple based on gamer 'id'"""
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("SELECT * FROM gamer WHERE id = %s" % (gamer_ID))
    rows = cur.fetchall()
    conn.close()
    return rows

def view_inventory(gamer_ID):
    """Returns all the tuples in inventory(table) based on gamer_ID  """
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("SELECT * FROM inventory WHERE gamer_ID = %s" % (gamer_ID))
    rows = cur.fetchall()
    conn.close()
    return rows

def view_games(game_ID):
    """Returns the tuple based on the game_id"""
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("SELECT * FROM games WHERE game_id = %s" % (game_ID))
    rows = cur.fetchall()
    conn.close()
    return rows

def insert_owner(username, password):
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("INSERT or REPLACE INTO owner VALUES (?,?)", (username, password))
    print('inserted')
    conn.commit()
    conn.close()

def insert(id, gamer_name, email, gamer_tag, age):
```

```
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("INSERT or REPLACE INTO gamer VALUES (?,?,?,?,?)", (id, gamer_name, email, gamer_tag, age))
    conn.commit()
    conn.close()

def insert_games(game_id, game_name, genre, release_date , Version , rating  ):
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("INSERT or REPLACE INTO games VALUES (?,?,?,?,?,?)", (game_id, game_name, genre, release_date,
Version, rating))
    conn.commit()
    conn.close()

def insert_inventory(game_ID, gamer_ID, game_name, play_time, achievments):
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("INSERT or REPLACE INTO inventory VALUES (?,?,?,?,?)", (game_ID, gamer_ID, game_name, play_time,
achievments))
    conn.commit()
    conn.close()

def delete(id):
    connection = sqlite3.connect(data_base_name)
    cur = connection.cursor()
    cur.execute("DELETE FROM gamer WHERE id = ?", (id, ))
    print('Deleted: ', id)
    connection.commit()
    connection.close()

def delete_game(game_ID):
    '''Deletes a game form inventory table'''
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("DELETE FROM inventory WHERE game_ID = ?", (game_ID, ))
    print('Deleted "%d" game from "games"',id)
    conn.commit()
    conn.close()

def update_owner(username, password):
    conn = sqlite3.connect(data_base_name)
    cur = conn.cursor()
    cur.execute("UPDATE owner SET password = '%s' WHERE username = '%s'" % (password,username))
    conn.commit()
    conn.close()

#database commands
connect()
```

```
insert_games(123, 'GTA V', 'Openworld  ', '17th Sep, 2013',5, 92)
insert_games(101, 'Battlefield 4', 'Shooter    ','29th Oct, 2013', 4, 77)
insert_games(102, 'Battlefield V', 'Shooter    ','20th Nov, 2018', 5, 'NA')
insert_games(103, 'Grand Theft Auto IV', 'Adventure    ','29th Apr, 2008', 4, 83)
insert_games(104, 'Grand Theft Auto: San Andreas', 'Adventure    ','26th Oct, 2004', 3, 91)
insert_games(105, 'Farming Simulator 19', 'Simulator      ','20th Nov, 2018', 1, 'NA')
insert_games(106, 'Farming Simulator 17', 'Simulator      ','25th Oct, 2016', 5, 90)
insert_games(107, 'Red Dead Redemption', 'Adventure      ','18th May, 2010', 2, 91)
insert_games(108, 'Read Dead Redemption 2', 'Adventure   ','26th Oct, 2018',2,99)
insert_games(109, "PLAYERUNKNOWN'S BATTLEGROUNDS", 'Shooter      ','20th Dec, 2017', 8, 85)
insert_games(110, 'Assassins Creed III', 'Strategy      ','30th Oct, 2012', 5, 73)
insert_games(111, 'Assassins Creed', 'Strategy      ','13th Nov, 2007', 6, 74)
insert_games(112, 'The Witcher 3: Wild Hunt', 'Adventure      ','19th May, 2015', 4, 97)
insert_games(113, 'The Witcher 2: Assassins of Kings', 'Adventure      ','17th May, 2011', 4, 87)
```

LOGIN.PY

```
import
tkinter
        from tkinter import *
        from tkinter import messagebox

        import database
        import gui

        def test():
          print('Clicked')

        #Login Button click from login Frame
        def login():
          username = usernameEntry.get()
          password = passwordEntry.get()
          rows = database.view_owner(username)
          if not rows:
            print('Invalid user name or passowrd')
            messagebox.showerror("Oops!",'Invalid Username')
          else:
            if rows[0][1] == password:
              print(rows)
              loginWindow.destroy()
              gui.home(username)
            else:
              messagebox.showerror('Oops!','Invalid Password')
              forgotPassButton.grid(row = 2, column = 1, sticky = E)

        #signup button click form Login frame
        def signUp():
          new_username.set('')
          new_password.set('')
```

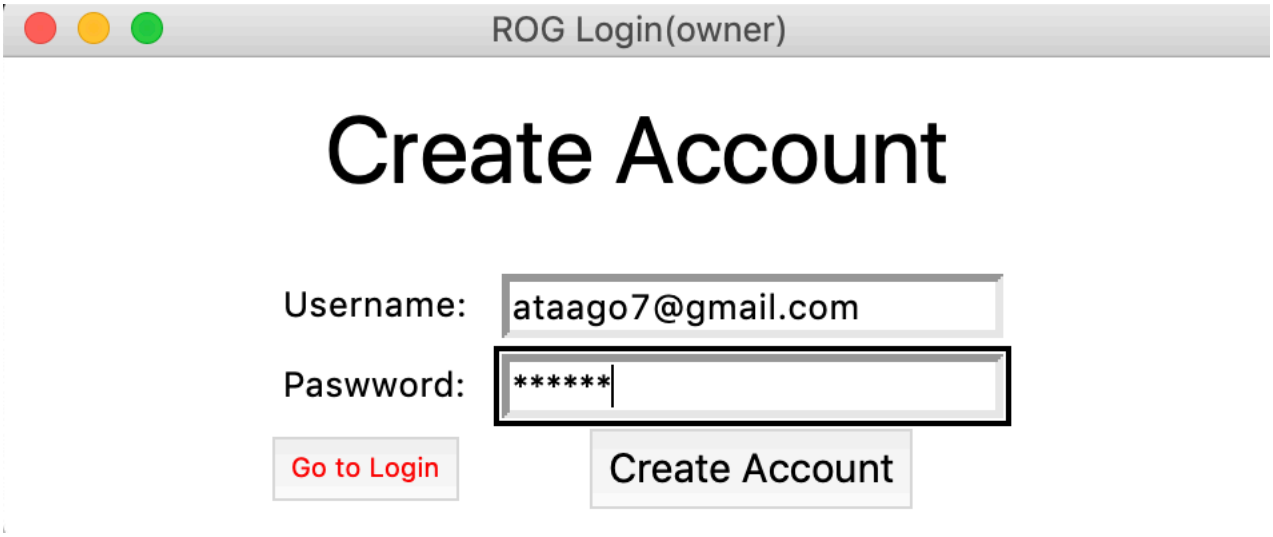
```
    loginFrame.pack_forget()
    Header['text'] = 'Create Account'
    createAccFrame.pack()

def createAccount():
    username = n_usernameEntry.get()
    password = n_passwordEntry.get()
    #dont create a empty username
    if username == '':
        messagebox.showwarning('Error','Enter username')
        return

    rows = database.view_owner(username)
    if rows:
        messagebox.showerror("Oops!",'Username Aldready Exists')
    else:
        database.insert_owner(username,password)
        messagebox.showinfo('Registered','Sucessfully Signed Up')
        gotoLogin()

def gotoLogin():
    username.set('')
    password.set('')
    createAccFrame.pack_forget()
    Header['text'] = 'LOGIN'
    loginFrame.pack()

#Password reset functions
def passReset():
    new_username.set('')
    new_username.set('')
    loginFrame.pack_forget()
    Header['text'] = "Password Reset"
    passResetFrame.pack()

def gotoLogin_from_passreset():
    username.set('')
    password.set('')
    passResetFrame.pack_forget()
    Header['text'] = "LOGIN"
    loginFrame.pack()

def passResetSubmit():
    username = usernameEntry1.get()
    new_password_1 = passresetEntry1.get()
    new_password_2 = passresetEntry2.get()

    rows = database.view_owner(username)
    if not rows:
```

```
        messagebox.showerror("Oops!",'Invalid Username')
    else:
        if new_password_1 != new_password_2:
            messagebox.showwarning('Error','Passwords do not match.')
        else:
            database.update_owner(username,new_password_1)
            messagebox.showinfo('Sucess','Password Changed')
            gotoLogin_from_passreset()

#main window login
loginWindow = Tk()
loginWindow.title("ROG Login(owner)")

def on_closing():
    if messagebox.askokcancel('Quit', 'Are you sure you want to quit?'):
        loginWindow.destroy()
loginWindow.protocol("WM_DELETE_WINDOW", on_closing)

Header = Label(loginWindow, text = 'LOGIN', font = ('',35), pady = 10)
Header.pack()

username = StringVar()
password = StringVar()
new_username = StringVar()
new_password = StringVar()
new_password2 = StringVar()

#login Frame
loginFrame = Frame(loginWindow, padx = 100, pady = 10)

usernameLabel = Label(loginFrame, text = "Username: ")
usernameLabel.grid(row = 0, column = 0)
usernameEntry = Entry(loginFrame, textvariable = username, bd = 3)
usernameEntry.grid(row = 0, column = 1)

passwordLabel = Label(loginFrame, text = "Paswword: ")
passwordLabel.grid(row = 1, column = 0)
passwordEntry = Entry(loginFrame, textvariable = password, show = '*', bd = 3)
passwordEntry.grid(row = 1, column = 1)

signUpButton = Button(loginFrame, text = 'Sign Up', font = ('',10), padx = 5, pady = 5, fg = 'red', command = signUp)
signUpButton.grid(row = 3, column = 0, sticky = W)

loginButton = Button(loginFrame, text = 'Login', font = ('',15), padx = 5, pady = 5, bg = 'blue', command = login)
loginButton.grid(row = 3, column = 1)

forgotPassButton = Button(loginFrame, text = 'Forgot password', font = ('',8), fg = 'blue', command = passReset)

loginFrame.pack()
```

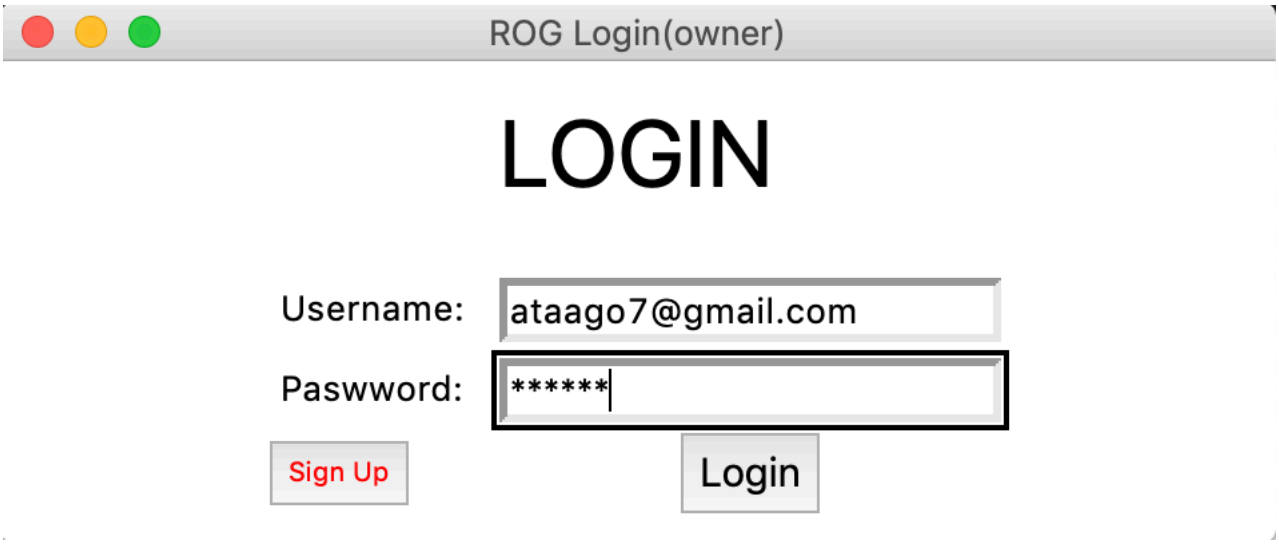
```
#Create Account frame
createAccFrame = Frame(loginWindow, padx = 100, pady = 10)

n_usernameLabel = Label(createAccFrame, text = "Username: ")
n_usernameLabel.grid(row = 0, column = 0)
n_usernameEntry = Entry(createAccFrame, textvariable = new_username, bd = 3)
n_usernameEntry.grid(row = 0, column = 1)

n_passwordLabel = Label(createAccFrame, text = "Paswword: ")
n_passwordLabel.grid(row = 1, column = 0)
n_passwordEntry = Entry(createAccFrame, textvariable = new_password, show = '*', bd = 3)
n_passwordEntry.grid(row = 1, column = 1)

gotoLoginButton = Button(createAccFrame, text = 'Go to Login', font = ('',10), padx = 5, pady = 5, fg = 'red', command = gotoLogin)
gotoLoginButton.grid(row = 2, column = 0, sticky = W)

createAccButton = Button(createAccFrame, text = 'Create Account', font = ('',15), padx = 5, pady = 5, bg = 'blue', command =
createAccount)
createAccButton.grid(row = 2, column = 1)



#Reset Password
passResetFrame = Frame(loginWindow, padx = 100, pady = 10)

usernameLabel = Label(passResetFrame, text = "Username: ")
usernameLabel.grid(row = 0, column = 0)
usernameEntry1 = Entry(passResetFrame, textvariable = new_username, bd = 3)
usernameEntry1.grid(row = 0, column = 1)

passresetLabel1 = Label(passResetFrame, text = "New Password: ")
passresetLabel1.grid(row = 1, column = 0)
passresetEntry1 = Entry(passResetFrame, textvariable = new_password, show = '*', bd = 3)
passresetEntry1.grid(row = 1, column = 1)

passresetLabel2 = Label(passResetFrame, text = "Re-Enter Password: ")
passresetLabel2.grid(row = 2, column = 0)
passresetEntry2 = Entry(passResetFrame, textvariable = new_password2, show = '*', bd = 3)
passresetEntry2.grid(row = 2, column = 1)

gotoLoginButton = Button(passResetFrame, text = 'Go to Login', font = ('',10), padx = 5, pady = 5, fg = 'red', command =
gotoLogin_from_passreset)
gotoLoginButton.grid(row = 3, column = 0, sticky = W)

submitButton = Button(passResetFrame, text = 'Submit', font = ('',15), padx = 5, pady = 5, bg = 'blue', command = passResetSubmit )
submitButton.grid(row = 3, column = 1)

loginWindow.mainloop()
```

Results (Screen shots of the Program)

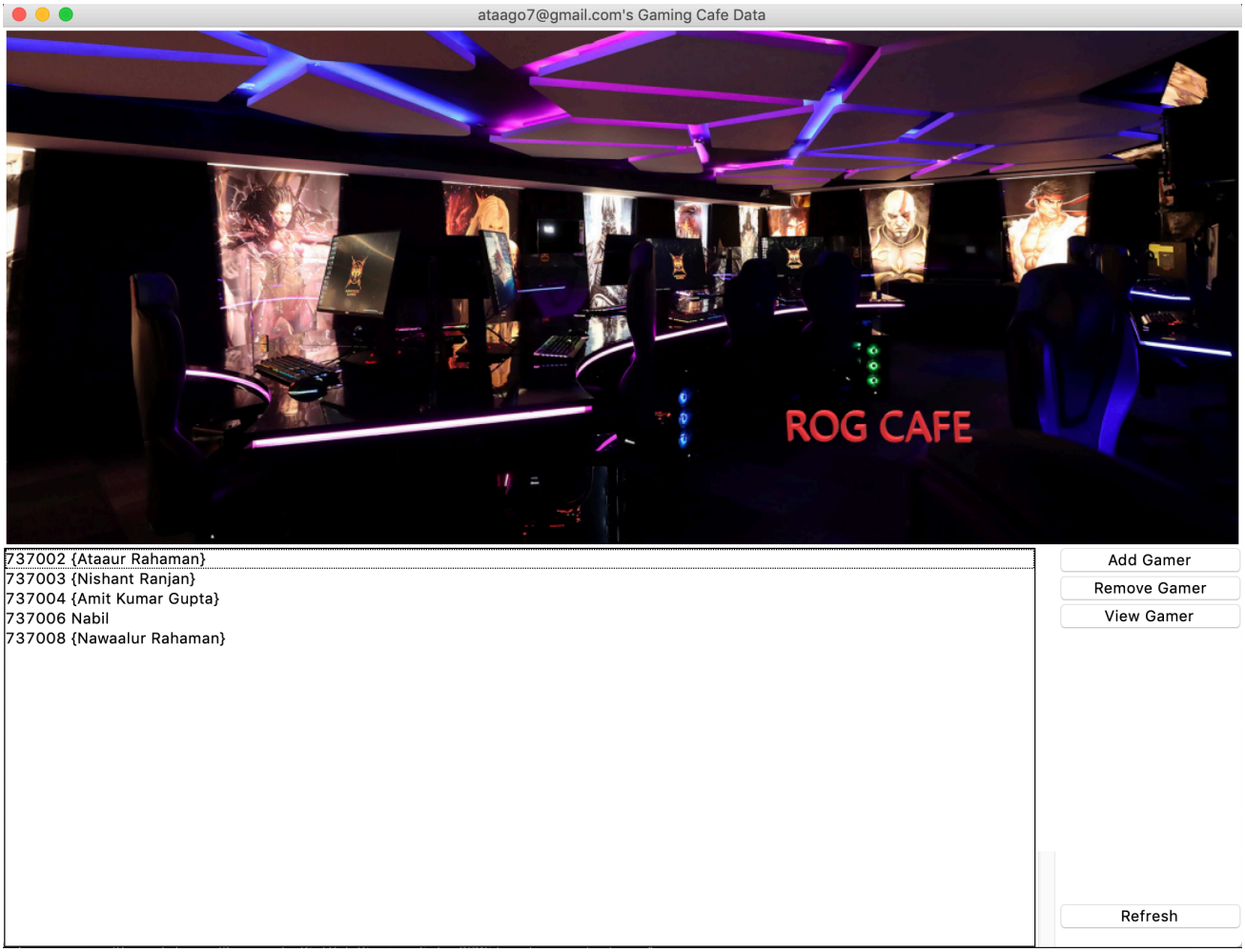
Sign Up window(owner):



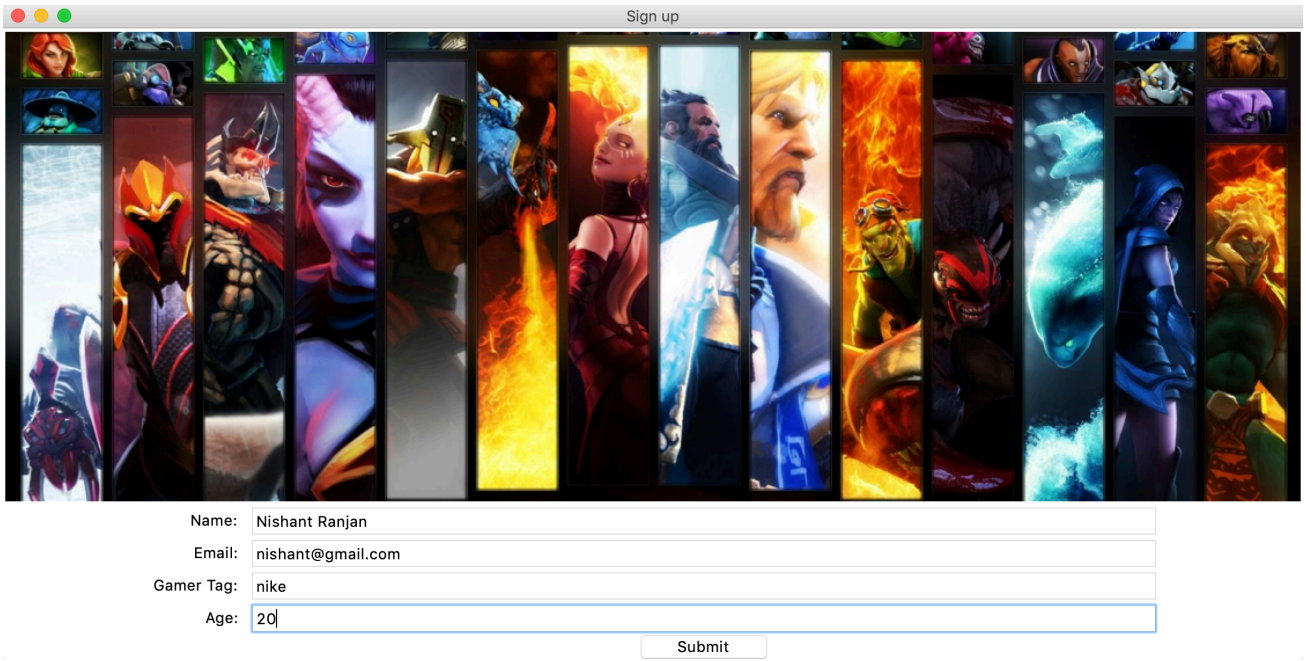
Login Screen(owner):



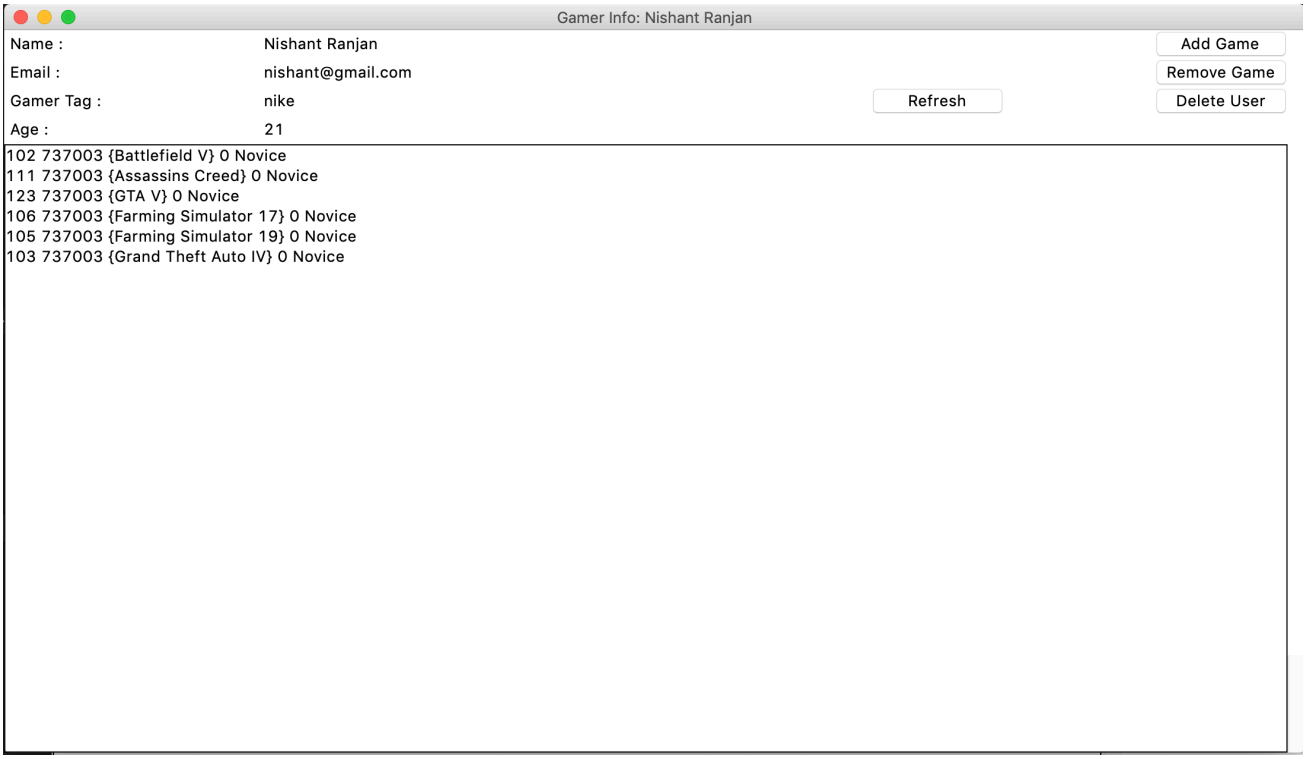
Main window:



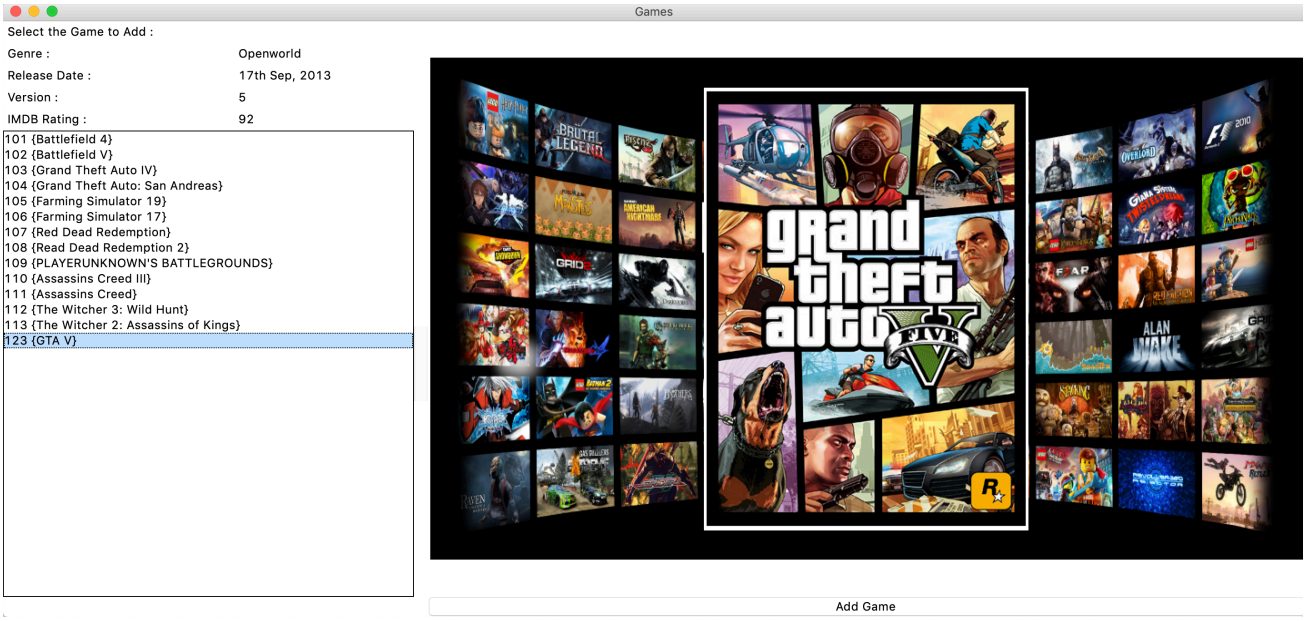
Gamer Signup:



View Gamer Window:



Add Game window:



Conclusions and Future enhancements

In this Gaming world, A **LAN Gaming Center** is a business where one can use a computer connected over a LAN to other computers, primarily for the purpose of playing multiplayer computer games. Use of these computers or game consoles costs a fee, usually per hour or minute. It may or may not serve as a regular café as well, with food and drinks being served. Many game centers have evolved in recent years to also include console gaming (Xbox, GameCube, PlayStation 2).

Managing these kind of setups is kind of tough job. For this very purpose we have made a Database Management Software which makes easier to manage all the Gamers in our Café with many features, to mention one, knowing the time played by an individual to charge accordingly.

In the future we are planning to actually implement the Database Management system using networking concepts and the LAN Local network to establish this software for every user(gamer) to login into his system having accounts made by this software. Futher we will be looking forward to log game play time from the user clients directly to the Gaming software.

References

https://www.tutorialspoint.com/python/

https://www.tutorialspoint.com/dbms/

https://www.w3schools.com/sql/