American University of Beirut
EECE 350 Final Project:
FIBERACE

Team Members

No.	Name	Email	ID	Weight
1	Majd Aboul Hosn	mga56@mail.aub.edu	202403227	25%
2	Tony Abi Haidar	tga22@mail.aub.edu	202400089	25%
3	Samer Barakat	srb13@mail.aub.edu	202401785	25%
4	Serena Stephan	sjs37@mail.aub.edu	202402011	25%

Feature Overview [1]

Name	Description	Status
Health System	Health varies depending on collision object, displayed at the top	FI
Multiplayer View	Two players on same screen, next to each other	FI
Multiple Obstacles	Spawning multiple types of obstacles with different damage levels	FI
Aid Kit	Add health upon collision	FI
Background Music	Looping audio during gameplay	FI
Car Movement	Move car using A/D or arrow keys smoothly	FI
Ammo Boxes	Collect to increase bullet count	FI
Game Difficulty	Partially implemented scaling of obstacle speed by integer n	PI
Game Over Logic	Show win/lose when health=0 or time ends for both players side-to-side	FI
Shooting	Press space to shoot obstacles to destroy them	FI
Sound of Collision	Collision sound over music	FI
Multiple Cars	Choose car color in garage (16 options)	FI
Choose Opponent	Send and accept/decline requests in a list of online players	FI
Email Verification	Verifying emails by sending a verification code using SMTP library	FI
Password encryption	salt hashing at the database using bcrypt module	FI
Account management	though name change, stats reset, and account deletion	FI

[1] FI: Fully implemented, PI: Partially implemented

1 Implementation Details

1.1 Database Functionalities and Programming Approach

The whole game was implemented in a functional programming approache. For the database, only the server can access it; which means all user data has to be sent to the server in order to be injected into the database for security. And passwords were also hashed in the database , adding an additional layer of security. "database.py" stores all the functions needed to manipulate the database (password checks, stats, leaderboard fetch, creating user credentials, validating emails, and game over stats updates).

1.2 Server Architecture

The server is always waiting for a message, and the message identifies the logic it has to follow. Additional data can also be sent or received, properly identified and extracted. Data transfer was achieved through long strings, of relevant information joined by a rare character sequence like (, ***) , and when the data needs to be processed at any side, the .split("***") method associated with the string is used.

1.3 Email Verification

On signup, a verification code is sent using Python's `smtplib`. Gmail app passwords are employed to bypass OAuth. Activation requires user entry of the received code.

1.4 Background Music Management

Always running just stop at game over.(It is implemented in the part titled music in the code) In the game over block , pygame.mixer.music.stop() stops the music when game over.

1.5 Opponent Selection Logic

A list of online players is displayed and refreshed whenever a new player joins, disconnects, or enters a game. A list of incoming and outgoing race requests is also shown. Note that players currently in a game do not appear in the "Available Players" list, which means they cannot be requested until they finish their game. Each outgoing request can be in one of four states: Pending, Accepted, Declined, or Disconnected. A request remains pending until the other player responds. If it is accepted, it immediately disappears from the "Your Requests" list. If it is declined or the requested player disconnects, the status changes to "Declined" or "Disconnected" respectively and is removed from the list after 5 seconds. An incoming request is displayed with the sender's name and "Accept" and "Decline" buttons. Once either button is clicked, the request disappears. If "Accept" is chosen, the original requester receives a "Confirm Play" pop-up with "Yes" and "No" options. If "Yes" is selected, the game begins successfully after peer-to-peer setup. Otherwise, the outgoing request is updated to show a Declined status, just as if the request had been declined earlier. We tested the code with more than two players, and it functioned successfully.

1.6 Multiplayer View

The game screen is designed to accommodate two players by setting its width to be twice that of the background image. This creates a split-screen layout where the left half is dedicated to the

main player and the right half to the opponent. The background image is drawn twice,once at the x-coordinate 0 for the main player's lane, and once at half the screen width for the opponent's lane effectively mirroring the game environment for both players. To display the opponent's car and obstacles on the right side, their positions are shifted horizontally using a shift() function that adds half the screen width to their coordinates. Health, names and time are displayed at the top of the screen.

1.7 Health System

The game features a health system where each obstacle carries a specific "poison" value that determines how much damage it inflicts on the player upon collision. This is managed using two parallel lists—one storing the file paths of obstacle images, and the other storing the corresponding poison values, matched by index. A helper function, $\text{get}_poison(ob), is used to retrieve the correct poison value for any given ob$

1.8 Car Movement

The car's vertical position is set near the bottom of the screen with a slight margin to keep it visible, using car_pos.y = screen_height - 1.15 * car_img.get_height(). Keyboard input is captured in real-time using pygame.key.get_pressed(), allowing the player to move the car horizontally. The left and right arrow keys move the car within defined boundaries to prevent it from going off the road, with the left limit set at 40 pixels and the right limit based on half the screen width minus the car's width and an extra margin. Additionally, the 'A' and 'D' keys serve as alternative controls, enhancing flexibility for different players. Overall, this setup ensures smooth and restricted movement within the player's lane, improving both control and gameplay experience.

1.9 Collision Sound

This is handled using Pygame's mixer.Sound() system. When a collision is detected using colliderect(), the collision_sound.play() command is triggered immediately.

1.10 Shooting Mechanics

The shooting mechanic allows the player to fire bullets upward to destroy incoming obstacles. When the spacebar is pressed and bullets are available, a new bullet is created using the create_bullet() function, which initializes a small rectangle with an upward speed. Each bullet reduces the bullet_count by one. Bullets are stored in a list and updated every frame to move upward. If a bullet collides with an obstacle, both the bullet and the obstacle are removed from the screen.

1.11 Ammo Boxes

Ammo boxes are power-up items that reward the player with additional bullets. They are spawned at regular time intervals controlled by a timer (drop_ammo_timer). Each box appears at a random horizontal position within the main player's lane and descends from the top of the screen. When the player's car collides with an ammo box, detected using colliderect(), the box is removed and the player's bullet count increases by three.

1.12 Game Difficulty Feature

The game dfifficulty feature is partially implemented in the game logic code through scaling speed of obstacles by an integer n. It just requires being pipelined from the online players page through a drop down menu button.

1.13 Game Over Logic

The game over logic is triggered when either the player's health drops to zero or the game timer runs out, background music stops and a game over sound effect plays. The screen is cleared and replaced with a "GAME OVER!" message, along with detailed statistics for both players—including their names, remaining health, number of hits taken, and the total elapsed time. The winner is determined based on who has more health at the end. The game waits a few seconds to display the results and then exits.

1.14 Customizable Car

The game includes a customizable garage feature that allows players to choose between different car colors. These color options are implemented by importing multiple car images from a designated folder, with each image representing a unique color variation. Players can navigate through available car options using left and right arrow buttons.

1.15 Network Connections

The main server binds to a fixed port and IP and actively listens for incoming connections. Upon connection with the server, each client sends their IP and port, which is then stored in an "online-players" list. After a race is accepted, the main server sends a "SERVER" message to one user and a "CLIENT" message to the other. The "CLIENT" connects directly to the server and once the connection is established, both players exchange data in a bidirectional manner (send and receive data) using the same socket (we know that TCP is full duplex). The peer-to-peer connection is established until the game is over, and it is communicated again to the main server that the players are available. We chose TCP for its reliability. Critical game data (like race requests, player stats) must arrive intact and in order. While UDP offers faster transmission, our game doesn't require sub-millisecond precision. Therefore, TCP works perfectly fine.

1.16 Real-Time Game State Synchronization

At every iteration of the game loop, each player sends their current state to their opponent and simultaneously receives the opponent's state. The received state is then rendered on the right half of the screen. This enables both players to see real-time updates of each other's actions during the game. Loading and Caching: The code contains several functions dedicated to loading and scaling images and sounds needed for the game. To optimize performance, these functions also implement caching mechanisms, storing the loaded assets in dictionaries or variables so they aren't reloaded from disk each time they are needed. Object Creation: Generating in-game objects dynamically. Obstacles are randomly selected from a predefined list, placed on the screen at a random position, and assigned a corresponding damage value. Aid kits are also randomly positioned and restore health when collected. Bullets are created and positioned relative to the car's location when the player fires. Game State and Synchronization: The game supports real-time synchronization between two players through

socket-based communication. Information about the player is sent to the opponent, and at the same time received from them and analyzed accordingly to update their information (position, health, ...) Game Logic Functions: Several utility functions manage in-game behaviors. update_bullets iterates through all bullets, moves them according to their speed, and removes any that move off-screen. draw_bullets is used to render the bullets on the screen. The shift function helps in positioning opponent objects by adding an offset, ensuring they appear on the correct side of the screen. Endgame Management: The game_over function is responsible for managing the end of the race. It handles stopping background music, playing a game-over sound, displaying statistics for both players, and sending the final results to the main server. It also renders a "GAME OVER" message and other information such as health, hits, and elapsed time. Main Game Loop – start_race(): The core of the game is implemented in the start_race function, which initializes Pygame, sets up the game window and audio, and starts a continuous loop where the gameplay unfolds. This loop processes player inputs, updates position, generates obstacles and aid-kits over time, handles collision detection, and keeps track of time and health. It also ensures real-time communication between the two players by sending and receiving data every frame. The game ends either when the time limit is reached or when a player's health drops to zero, at which point the endgame sequence is triggered.

Appendices

2 Individual Contributions[2]

Majd Aboul Hosn

- Report
- Sign-in Page (PyQT)
- Sign-UP Page (PyQT)
- Garage Page (PyQT)
- Leaderboard page (PyQt)
- Main menu Page (PyQt)
- List of online players page(PyQt)
- Shooting (GUI)
- Ammo Features (GUI)

Serena Stephan

- Connected the PyQt pages
- Wrote most of the functions and "if statements" in the server code along with the corresponding code on the client side across different pages
- Established the peer-to-peer communication

[2] All of these were fully implemented

- Implemented the backend of "pyqt_players" page: Handled the lists of incoming requests, outgoing requests and available players (all refreshed with every update and change of statuses).
- Successfully sent and received each player's game status (i.e. obstacles, aid kit, bullets, health, hits)

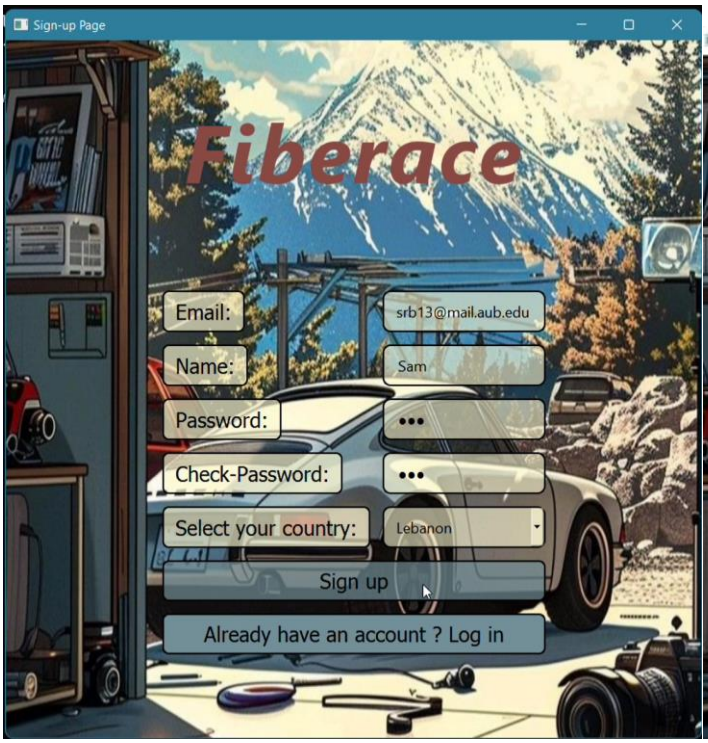
Tony Abi Haidar

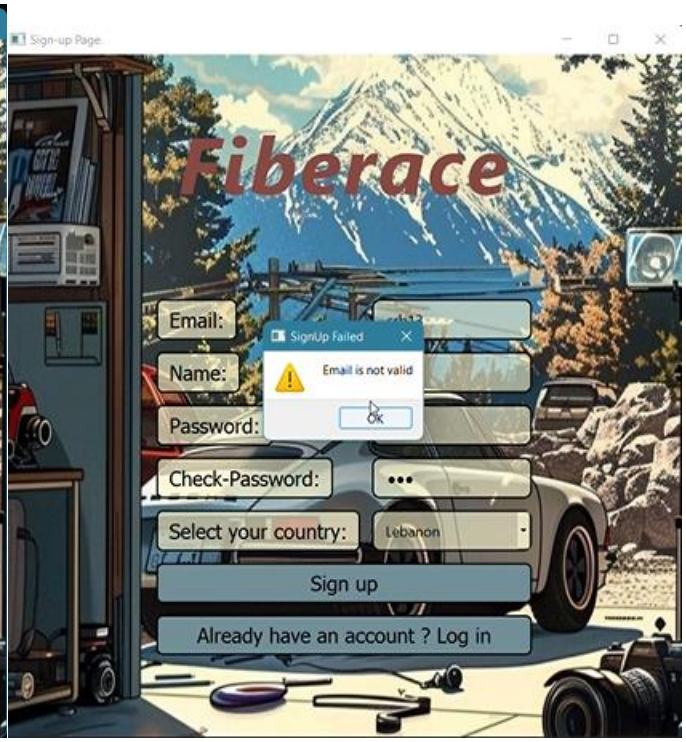
- Game Flow Design Functionalities:
  Music and Collision Sound Obstacles and Aid Kit generation
  Movement
  Car Movement
  Multiplayer Move
  Health Decreasing and Increasing
- Page Responsivity: changing size of elements depending on the window size
- Email Verification Feature
- Connection and lagging bugs fixing
- Project Required Video

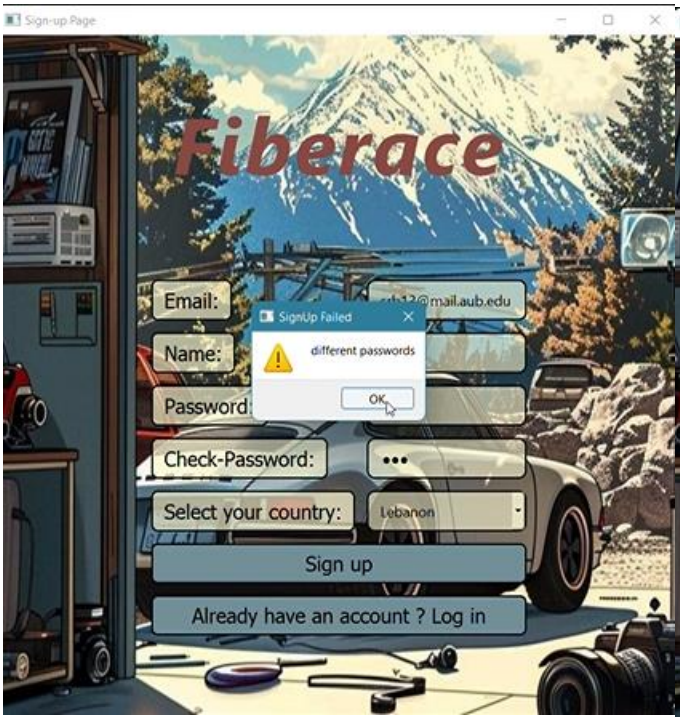
Samer Barakat

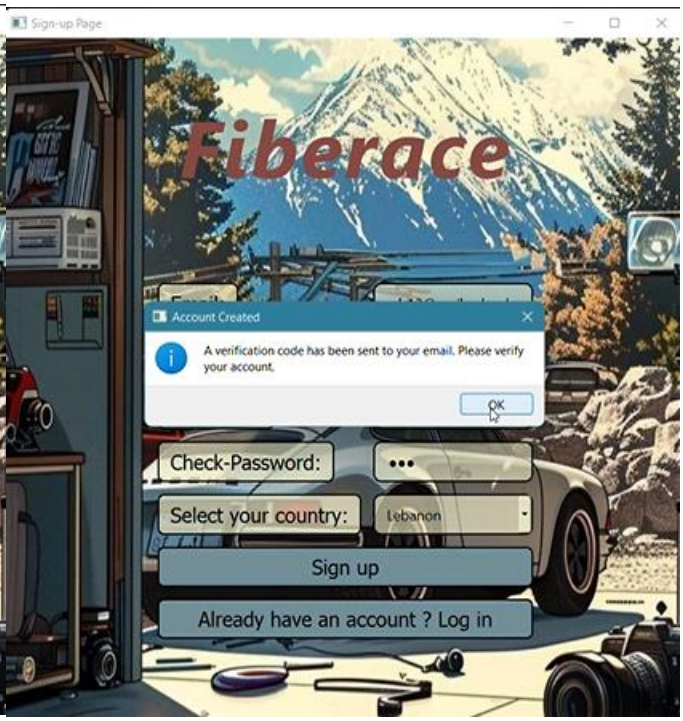
- Front end design of PYQT, color accents and hover effects (pyqt)
- Navigation Feasibility like buttons and layout (pyqt)
- Database functions (sqlite3)
- Server: Account management and navigation (sqlite3 ,socket, pyqt)
- Game and Game-Over element and text display (pygame)
- Settings functionalities (pyqt, socket)
- Statistics Page (pyqt)
- Password encryption (bcrypt)
- Bad input handling (warning boxes) (pyqt)
- Different health decrements (pygame)
- Integrating everyone's codes
- Server edge case handling in some cases (socket)
- Report editing finalization (LaTeX)

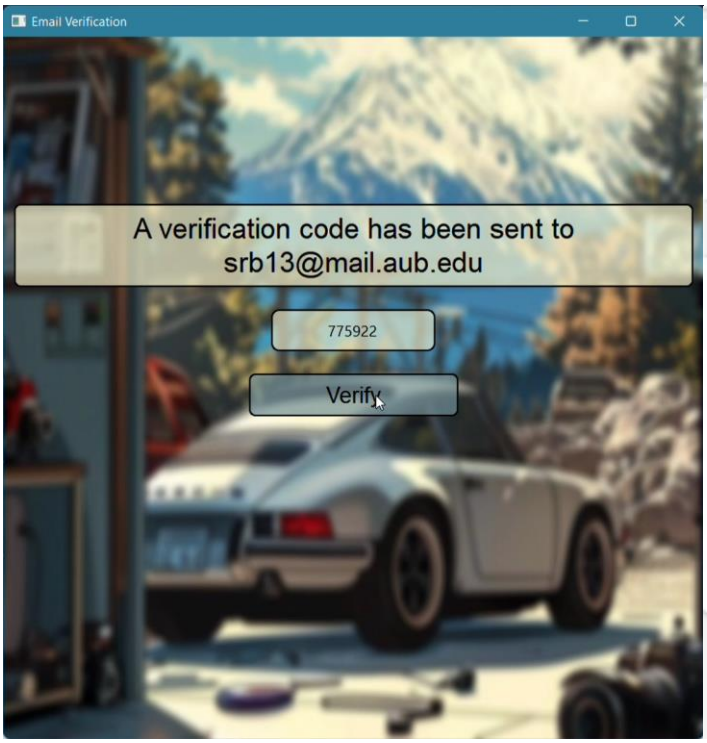
3 Game GUI and flow Snippets

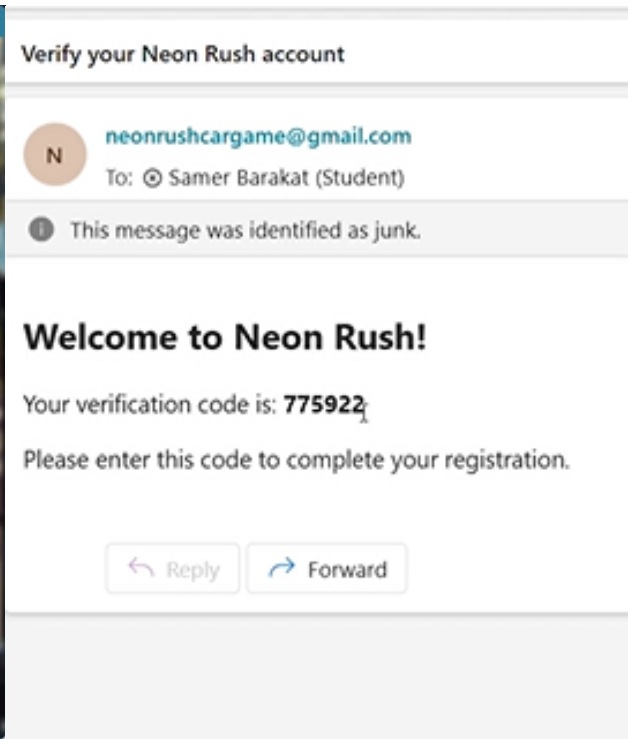

Sign-up Page
Fiberace
Email:
srb13@mail.aub.edu
Name:
Sam
Password:
Check-Password:
Select your country:
Lebanon
Sign up
Already have an account ? Log in

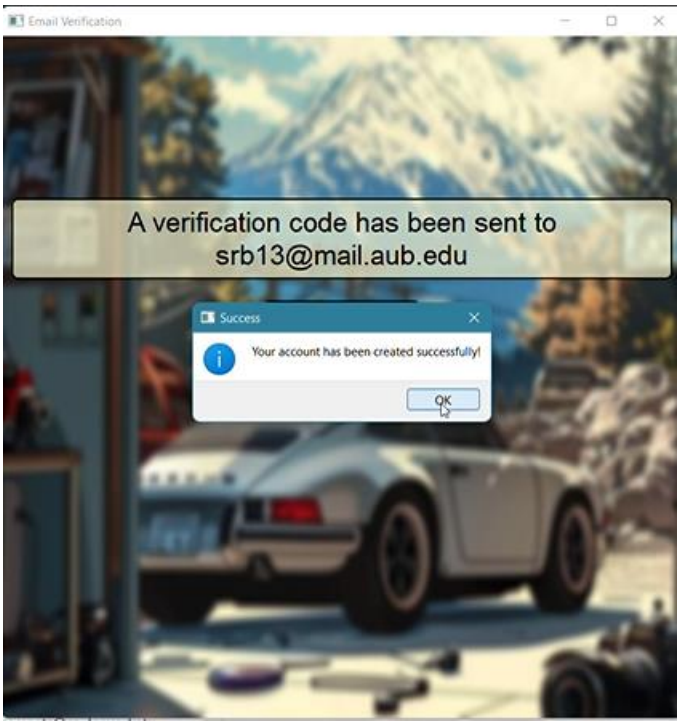

Sign-up Page
Fiberace
Email:
SignUp Failed
Email is not valid
OK
Name:
Password:
Check-Password:
Select your country:
Lebanon
Sign up
Already have an account ? Log in

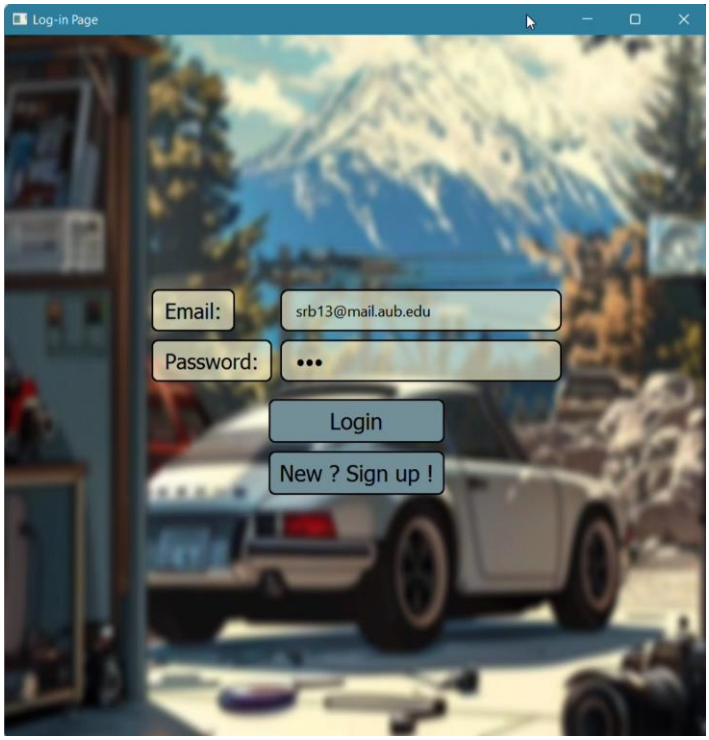

Sign-up Page
Fiberace
Email:
SignUp Failed
different passwords
OK
Name:
Password
Check-Password:
Select your country:
Lebanon
Sign up
Already have an account ? Log in

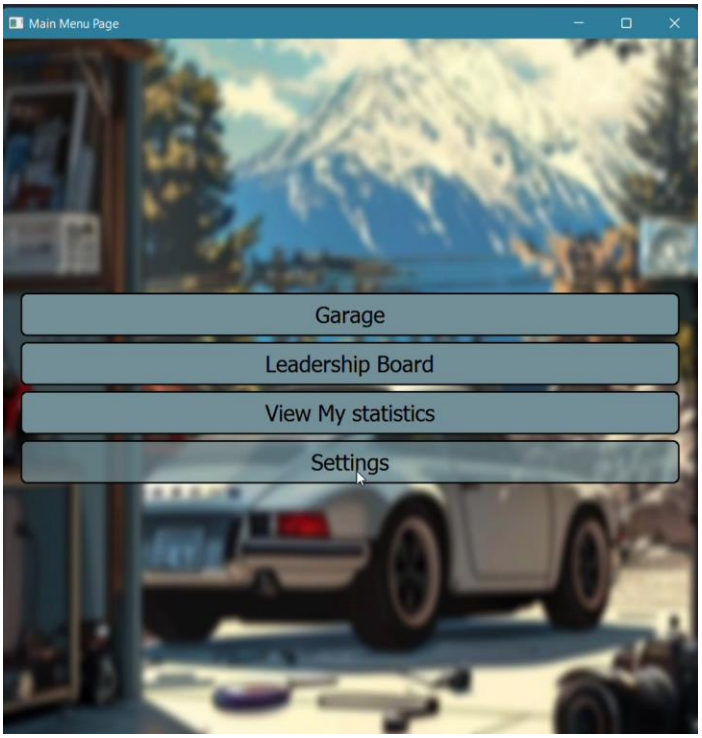

Sign-up Page
Fiberace
Account Created
A verification code has been sent to your email. Please verify your account.
OK
Check-Password:
Select your country:
Lebanon
Sign up
Already have an account ? Log in

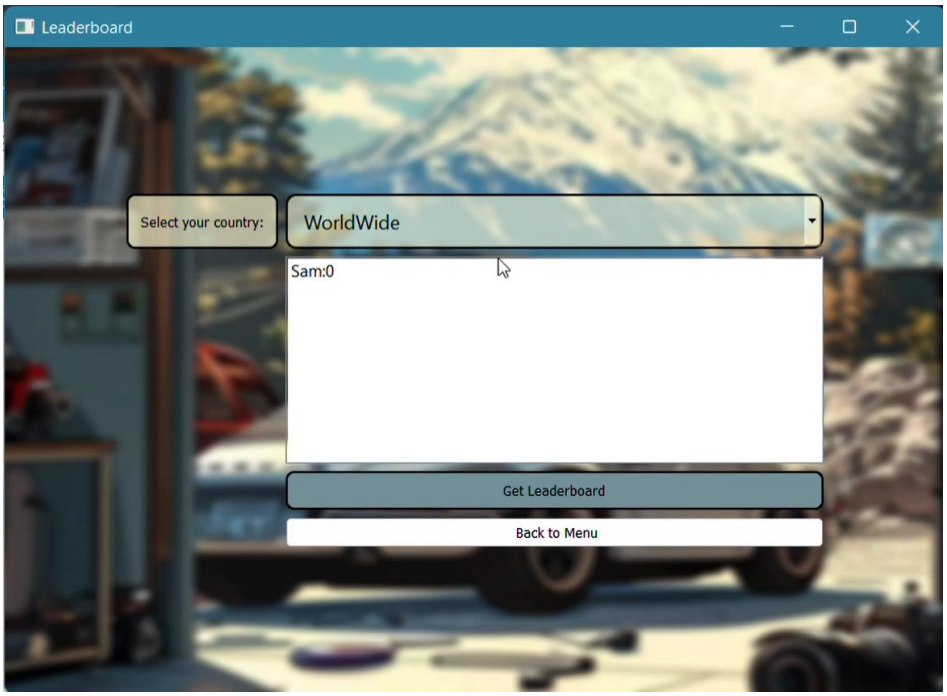

Email Verification
A verification code has been sent to srb13@mail.aub.edu
775922
Verify

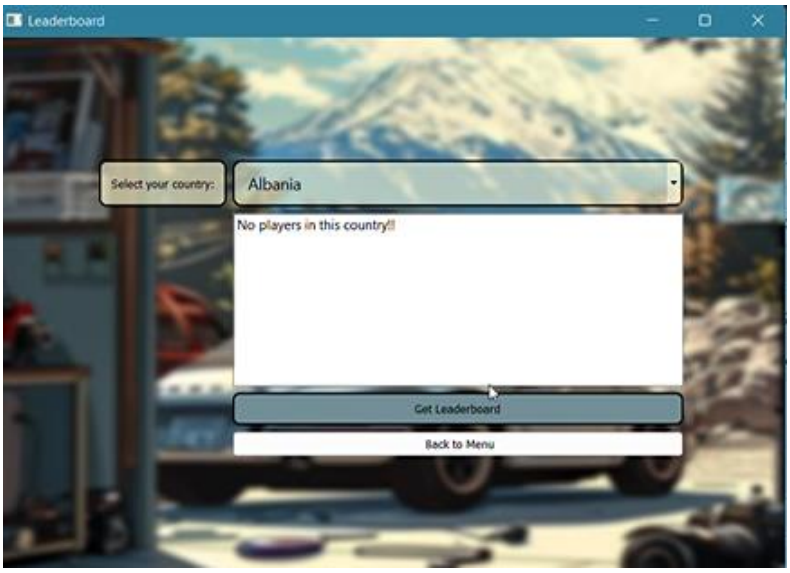

Verify your Neon Rush account
N
neonrushcargame@gmail.com
To: Samer Barakat (Student)
This message was identified as junk.
Welcome to Neon Rush!
Your verification code is: 775922
Please enter this code to complete your registration.
Reply
Forward

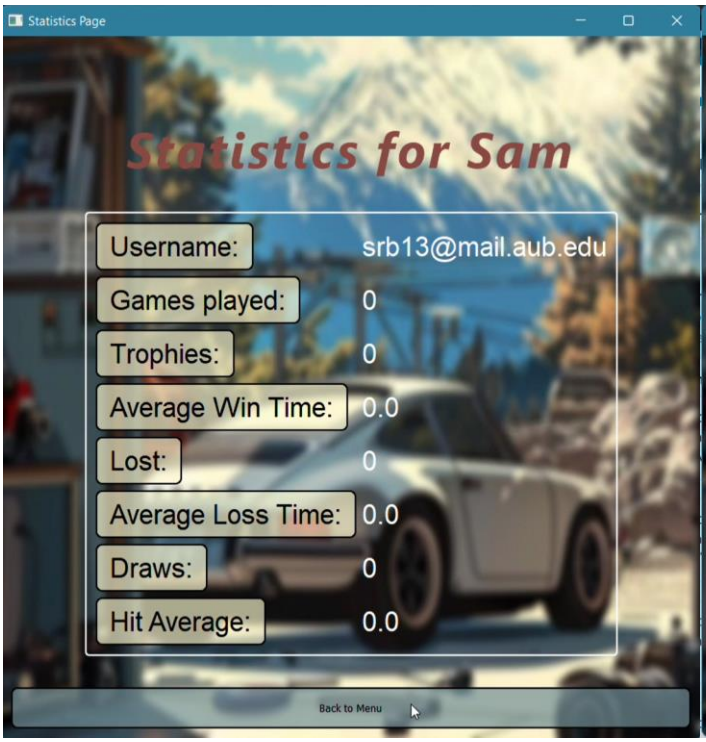

Email Verification
A verification code has been sent to srb13@mail.aub.edu
Success
Your account has been created successfully!
OK

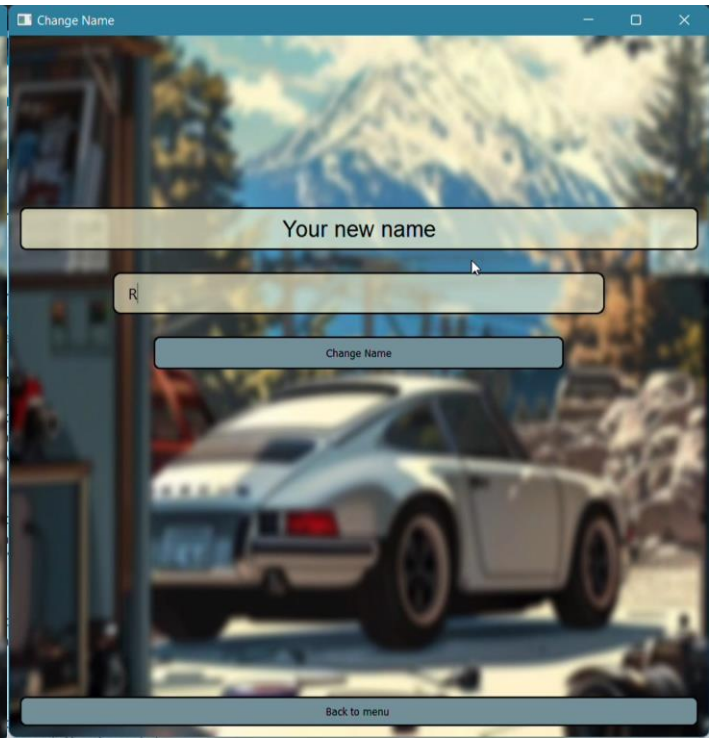

Log-in Page
Email:
srb13@mail.aub.edu
Password:
•••
Login
New ? Sign up !

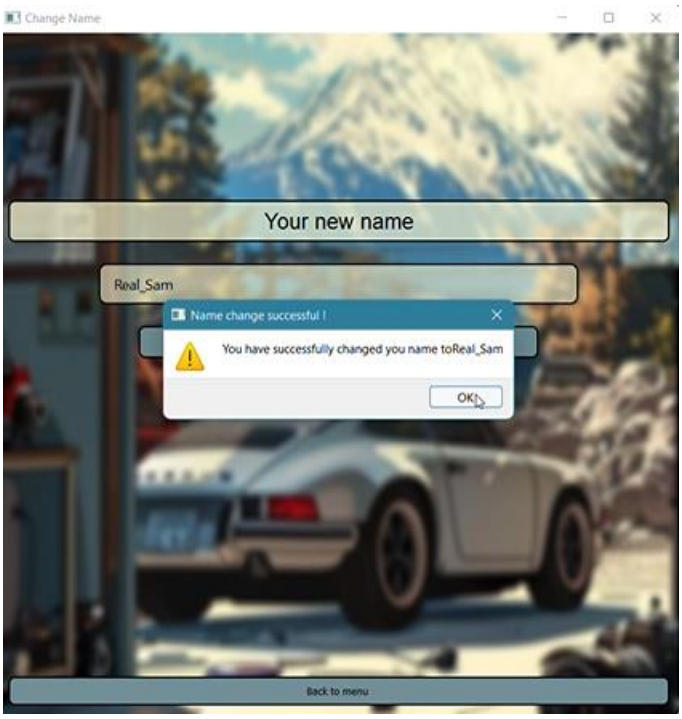

Main Menu Page
Garage
Leadership Board
View My statistics
Settings

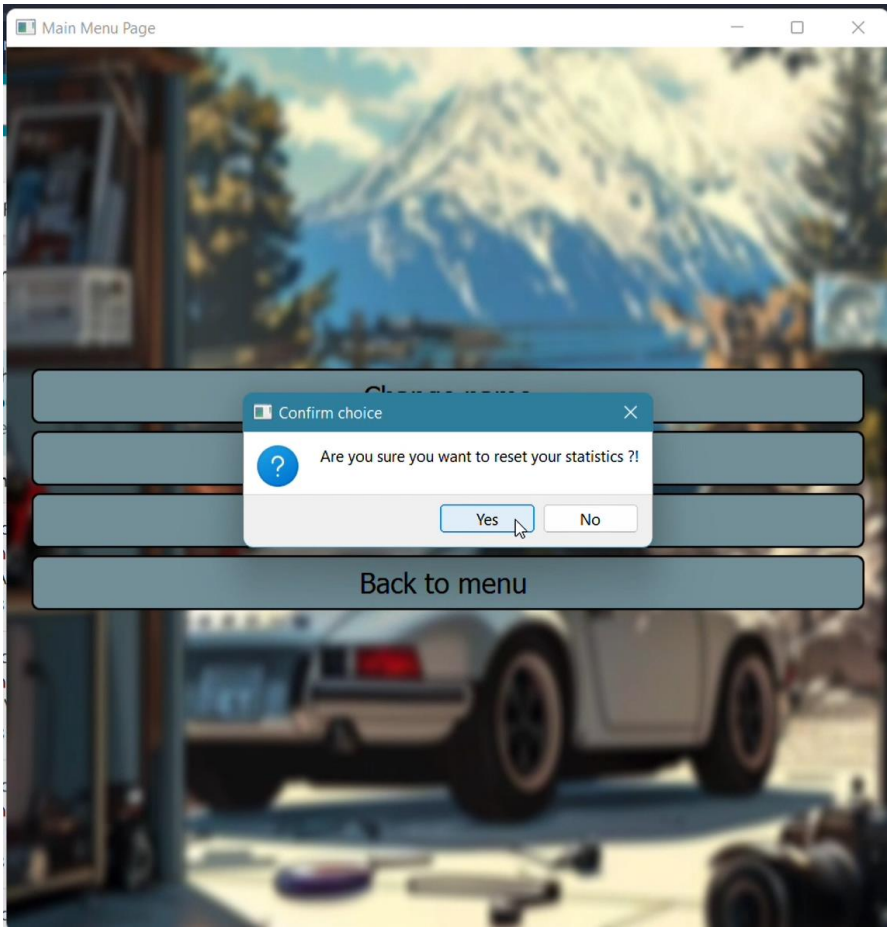

Leaderboard
Select your country:
WorldWide
Sam:0
Get Leaderboard
Back to Menu


Leaderboard
Select your country:
Albania
No players in this country!!
Get Leaderboard
Back to Menu

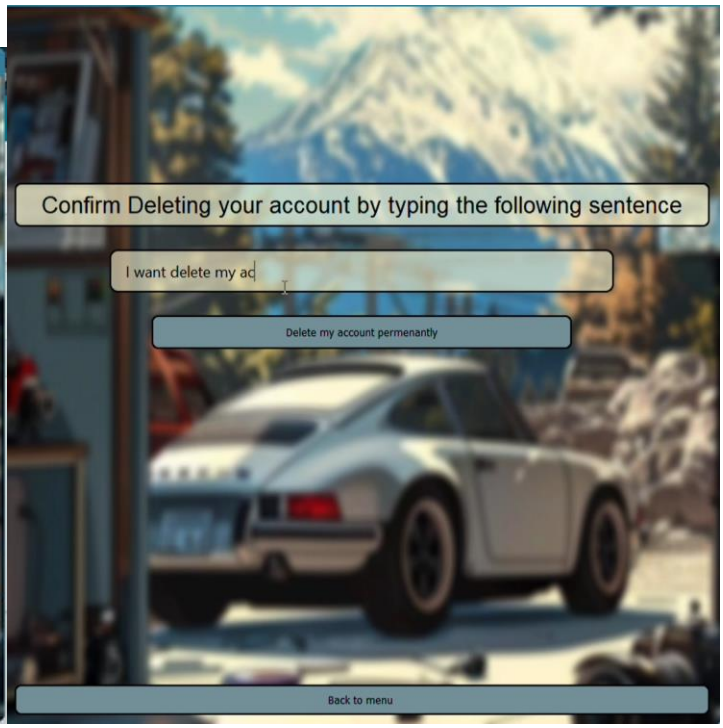

Statistics Page
Statistics for Sam
Username:
srb13@mail.aub.edu
Games played:
0
Trophies:
0
Average Win Time:
0.0
Lost:
0
Average Loss Time:
0.0
Draws:
0
Hit Average:
0.0
Back to Menu

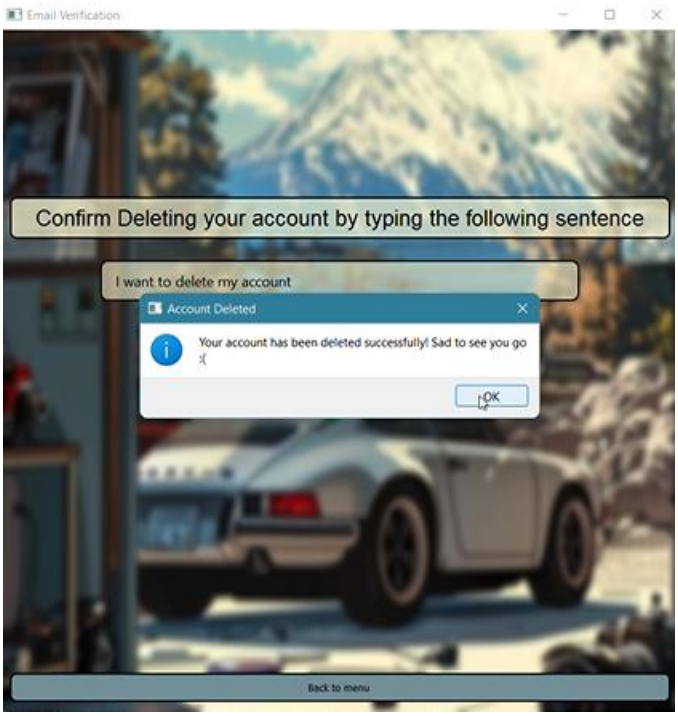

Change Name
Your new name
R
Change Name
Back to menu

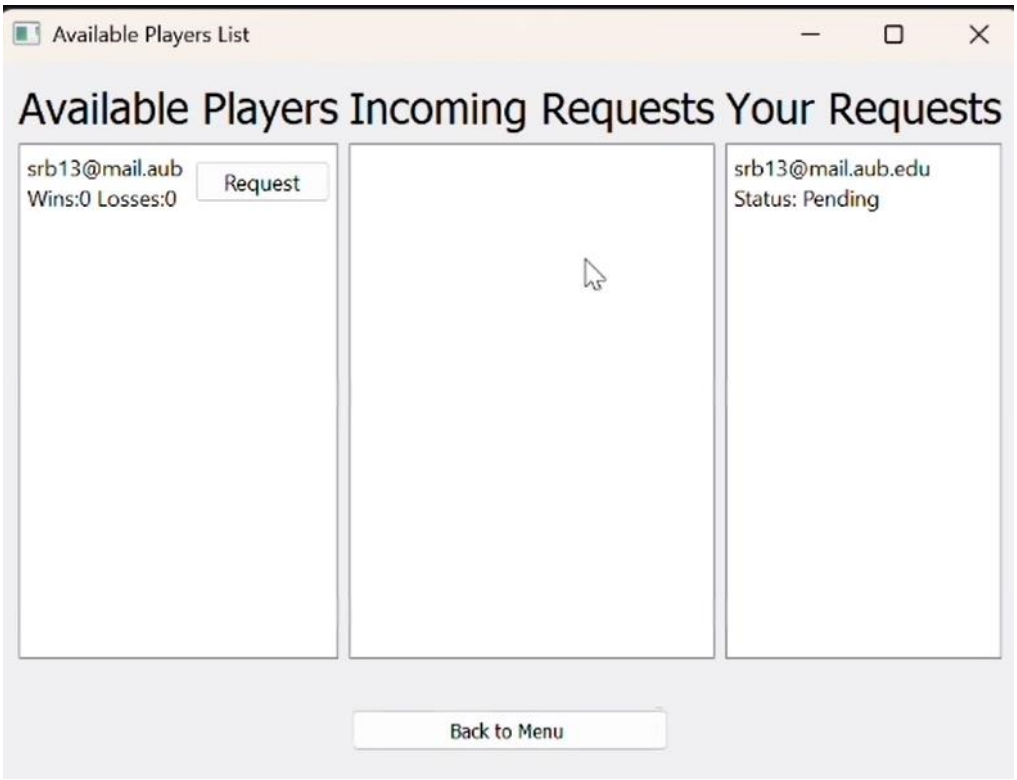

Change Name
Your new name
Real_Sam
Name change successful !
You have successfully changed you name toReal_Sam
OK
Back to menu

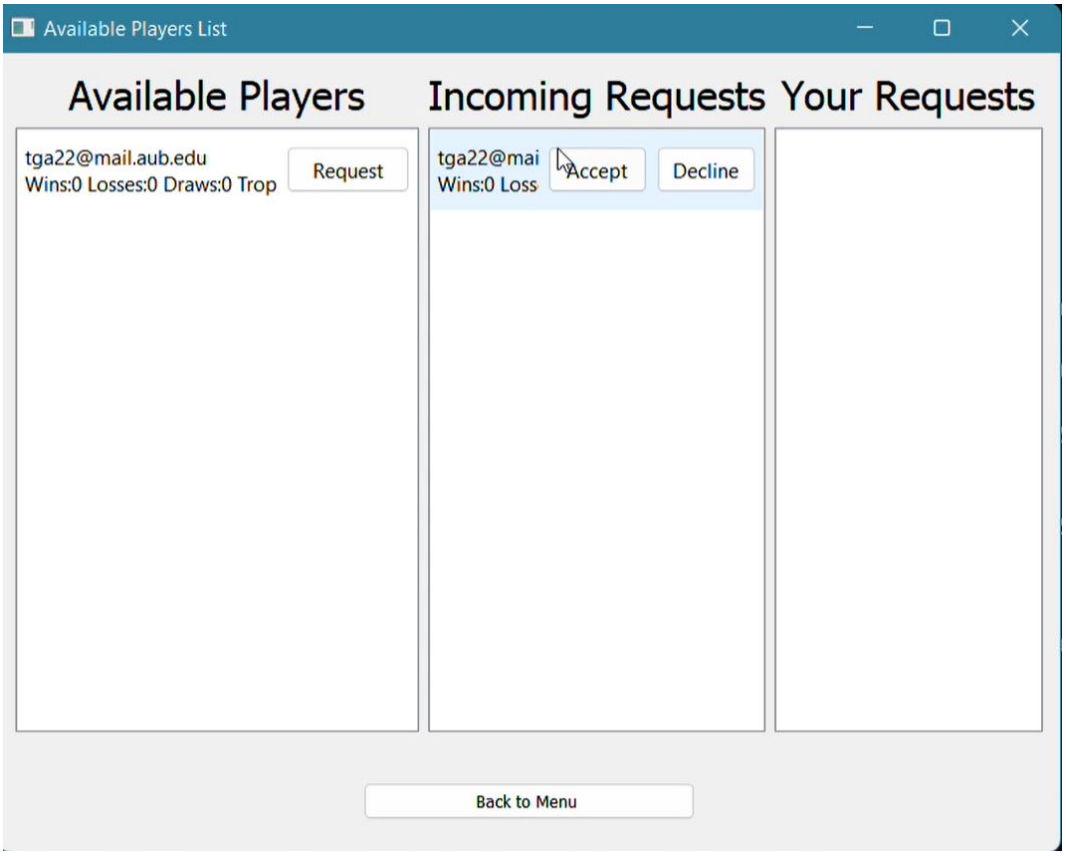

Main Menu Page
Confirm choice
Are you sure you want to reset your statistics ?!
Yes
No
Back to menu


Garage
Garage - Choose Your Car
Previous
Next
Look for opponent
Back to Menu


Confirm Deleting your account by typing the following sentence
I want delete my ac
Delete my account permenantly
Back to menu


Email Verification
Confirm Deleting your account by typing the following sentence
I want to delete my account
Account Deleted
Your account has been deleted successfully! Sad to see you go :(
OK
Back to menu


Available Players List
Available Players
Incoming Requests
Your Requests
srb13@mail.aub
Wins:0 Losses:0
Request
srb13@mail.aub.edu
Status: Pending
Back to Menu


Available Players List
Available Players
Incoming Requests
Your Requests
tga22@mail.aub.edu
Wins:0 Losses:0 Draws:0 Trop
Request
tga22@mai
Wins:0 Loss
Accept
Decline
Back to Menu


Nego Rush
srb13@mail.aub.edu 35
t9a22@mail.aub.edu 35
Time: 99


Nego Rush
srb13@mail.aub.edu 25
t9a22@mail.aub.edu 20
Time: 94

Note: This is Samer’r screen not Tony’s !