CS6106 – DATABASE MANAGEMENT SYSTEM

Feb - May 2021

Alternity

Shivakumar.N - 2019103059

Neeraj.G – 2019103041

Praveen Raj – 2019103593

Index

Project Abstract

‘Aternity’ is a game developed using python . The application allows users to login and choose the game of their choice in the Main Menu. The games that a user can play in the application are Flappy Bird, Snake game and Space Invader.

Users can play the games and their scores are updated in the database . A leaderboard of the high scorers of a particular game is displayed. Players can add their friends , play games to compete with their friend’s high scores and try to beat them.

The games and the front end of the application were made using the “pygame module” (python) and python was used for the back end scripting.

The game is for entertainment purposes , can be used to stay connected and compete with friends and enjoy from home during these difficult COVID times.

Introduction

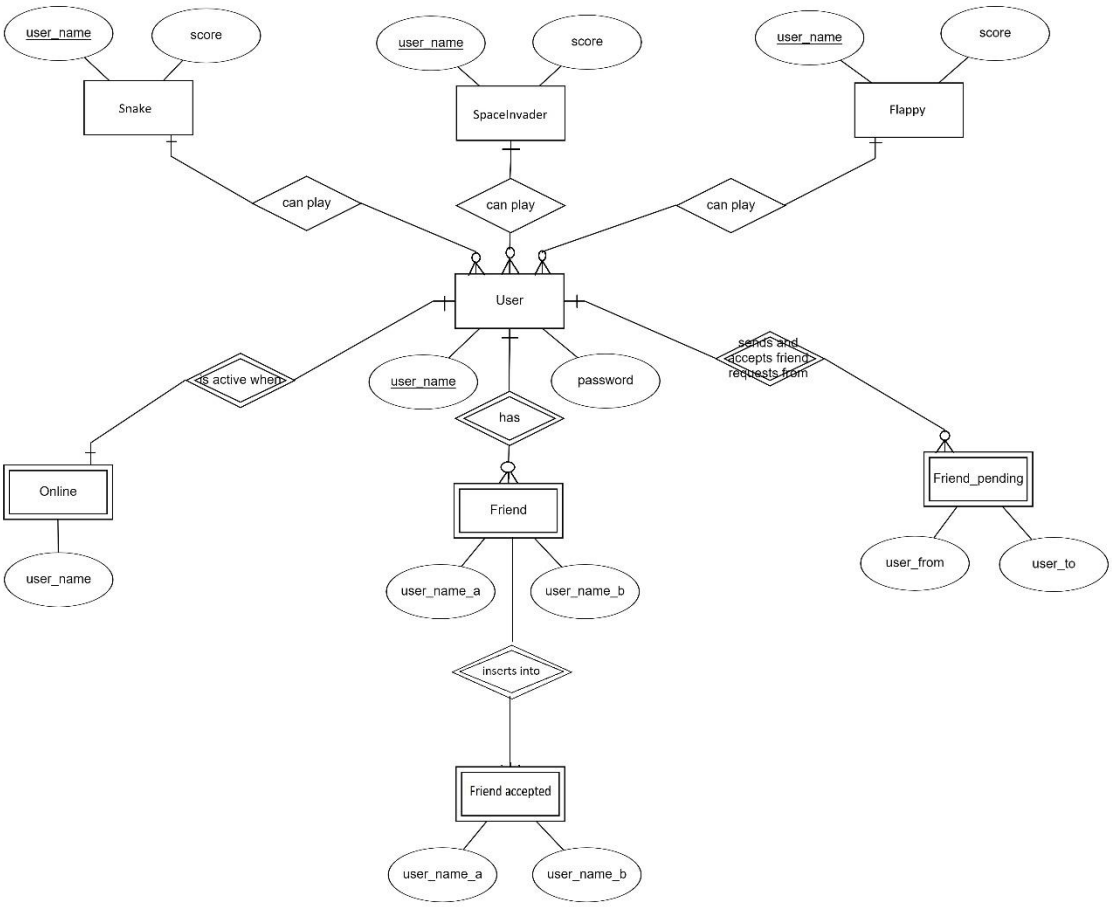
Alternity is a game application that aims at providing entertainment and pleasure to users during these difficult COVID times. With a massive increase in the number of COVID cases it is very important that people stay at home to help curb the spread of the disease .

The game merely provides users with a platform to play games with their friends and compete for high scores .

The application provides multiple functionalities each serving a unique purpose. With a fast and ostentatious user interface it provides users with the best gaming experience.

The application consists of a login page where users enter their login details and are redirected to a main menu screen. In the main menu screen users can select a game of their choice and also add friends . After playing a game users can check their score and also check their friend’s leaderboard (with scores of all their friends in descending order) or the global leaderboard for that particular game.

Entity Relationship Diagram



Front End

The front end, ie the application window is coded using python using the pygame module which provides an easily usable API for doing so.

Implementation and Screenshots:

Login Screen:

When the game is launched, the **'Login page'** of the game appears. It prompts for the player's username and password.

Figure 1:Login Screen

If the username doesn't exist, a **'User doesn't exist'** page appears.

This page allows the player to sign up with the entered username and password.

Figure 2:User Does Not Exist Screen

If the user exists and the entered password is wrong, **'Password doesn't match username'** message appears.

Figure 3:Password Error Screen

Main Menu:

If the username exists and the password matches username, the **'Main Menu'** page appears. This page allows the user to choose a game **(SELECT GAME),** manage friends and friend requests **(FRIENDS)** and quit game **(QUIT)**.

Figure 4:Main Menu Screen

If the **'Select game'** option in Main Menu is selected, **'Select Game'** page appears.

This page displays a list of games from which the user can choose a game to play.

Figure 5:Select Game Screen

Game Screens:

On selecting the **'snake'** option the snake game is launched.

This is a screenshot of the **'Snake'** game.

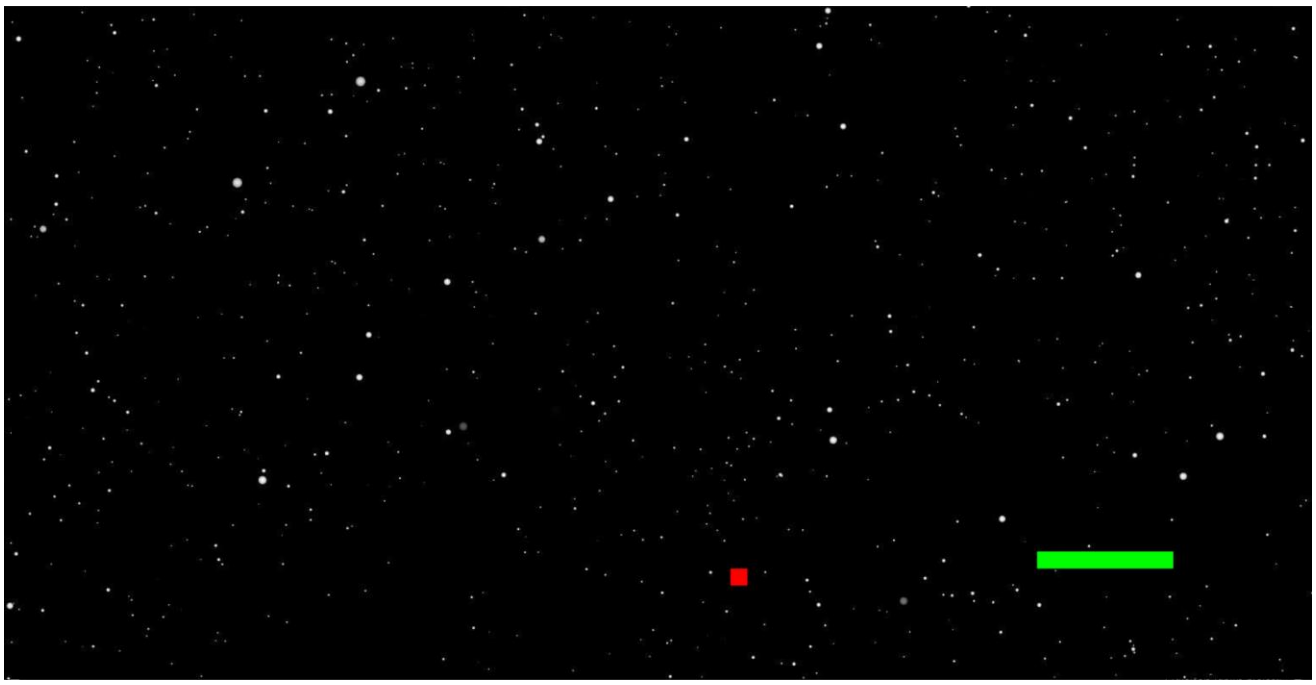


Figure 6:Snake Game

On selecting the **'Space game'** option the space game is launched.

This is a screenshot of the **'Space game'** game.

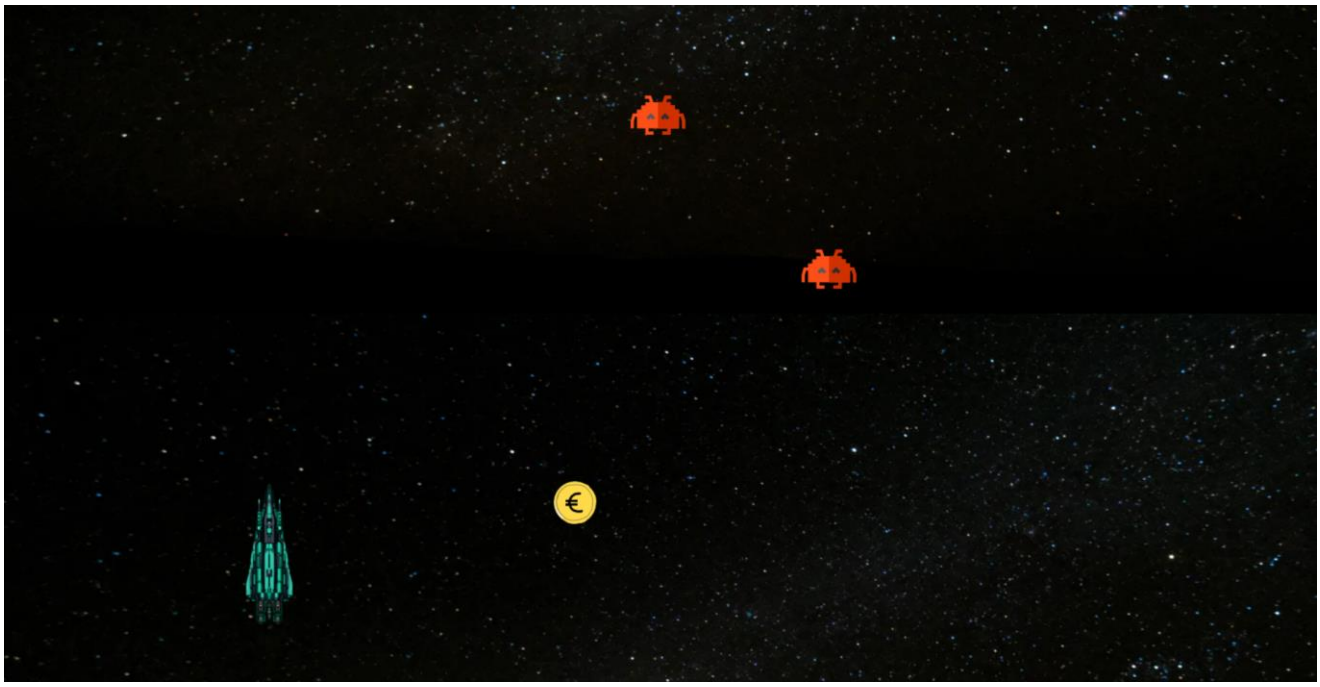


Figure 7:Space Game

On selecting the **'Flappy bird'** option the flappy bird game is launched.

This is a screenshot of the **'Flappy Bird'** game.

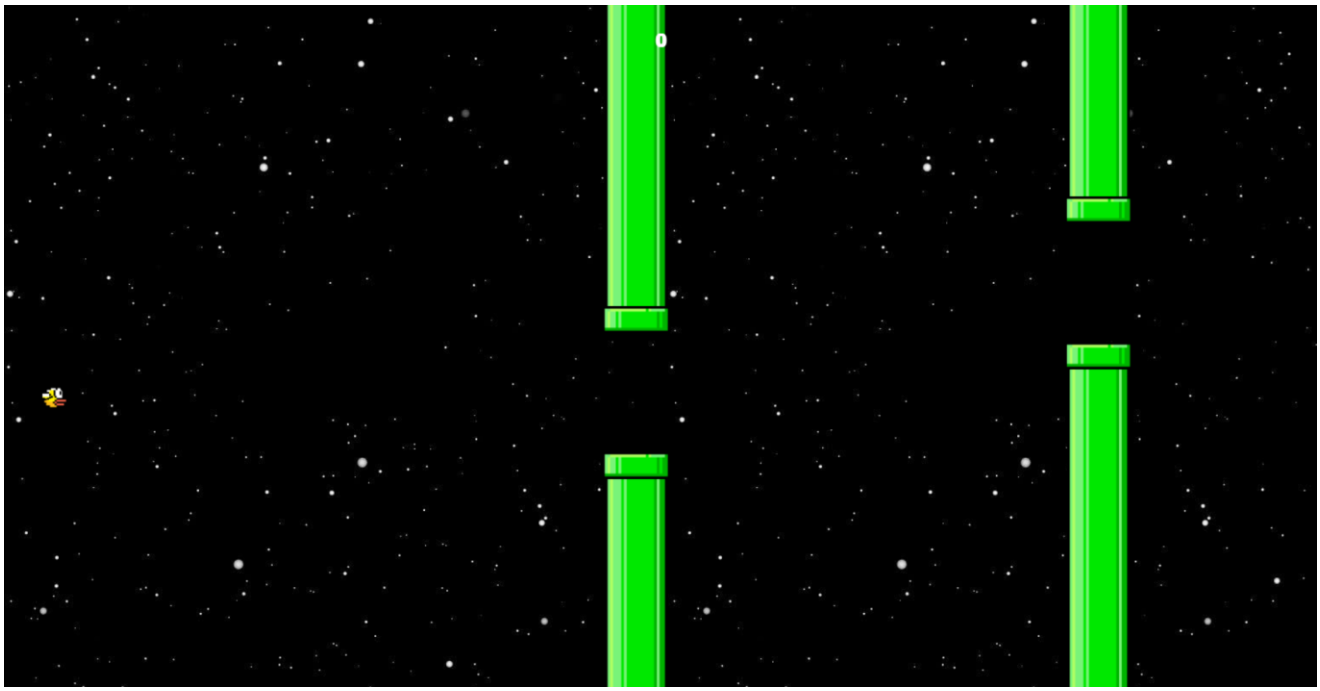


Figure 8:Flappy Bird Game

Post Game Screens:

After a game is played, the **'Post Game'** screen appears.

This page displays your score, the game's high scorer with the player's name, play again option, leader board option. It also allows the player to go to the main menu or to quit the game.

Figure 9:Post Game Screen

The **'Leader Board'** screen appears on selecting the leader board option of the postgame screen. This page allows the player to select the **Global** and **Friend's** leader board and to go back to the post screen.

Figure 10:Leaderboard Select Screen

The **'Global Leader Board'** screen appears on selecting the global leader board option of the leader board page. This page displays the high score of every player who has played the game.

Figure 11:Global Leaderboard

The **'Friends Leader Board'** screen appears on selecting the friends leader board option of the leader board page.

This page displays the high score of the player's friends who has played the game.

Figure 12:Friends Leaderboard Screen

Friends Screen:

On selecting the '**Friends'** option of the Main Menu, the **'Manage Friends'** page appears

Figure 13:Friends Screen Select

This page allows the user to **view friends, add friends**, **send and accept friend requests, remove friends** and to go back to the main menu.

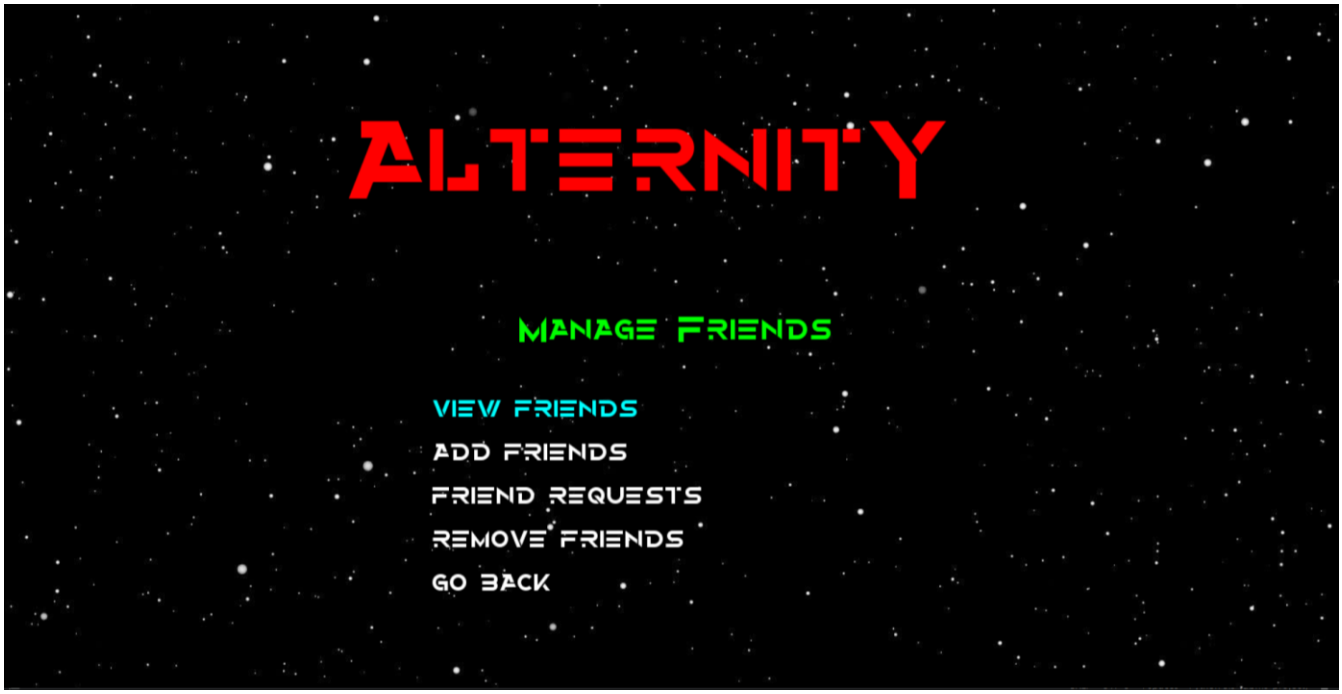


Figure 14:Manage Friends Screen

On selecting the **'View friends'** option of the Manage Friends page, the **'Your Friends'** page appears. This page displays all the friends of the player and if they are online or offline.

Figure 15:View Friends Screen

On selecting the **'Add Friends'** option of the Manage Friends page, the '**Add Friend'** page appears.

This prompts the player to enter the friend's username.

Figure 16:Add Friends Screen

If the username doesn't exist, **'Player doesn't exist'** message appears.

Figure 17:Player does not exist error

If the username exists, a friend request is sent to the user and the message **'Friend request sent'** appears.

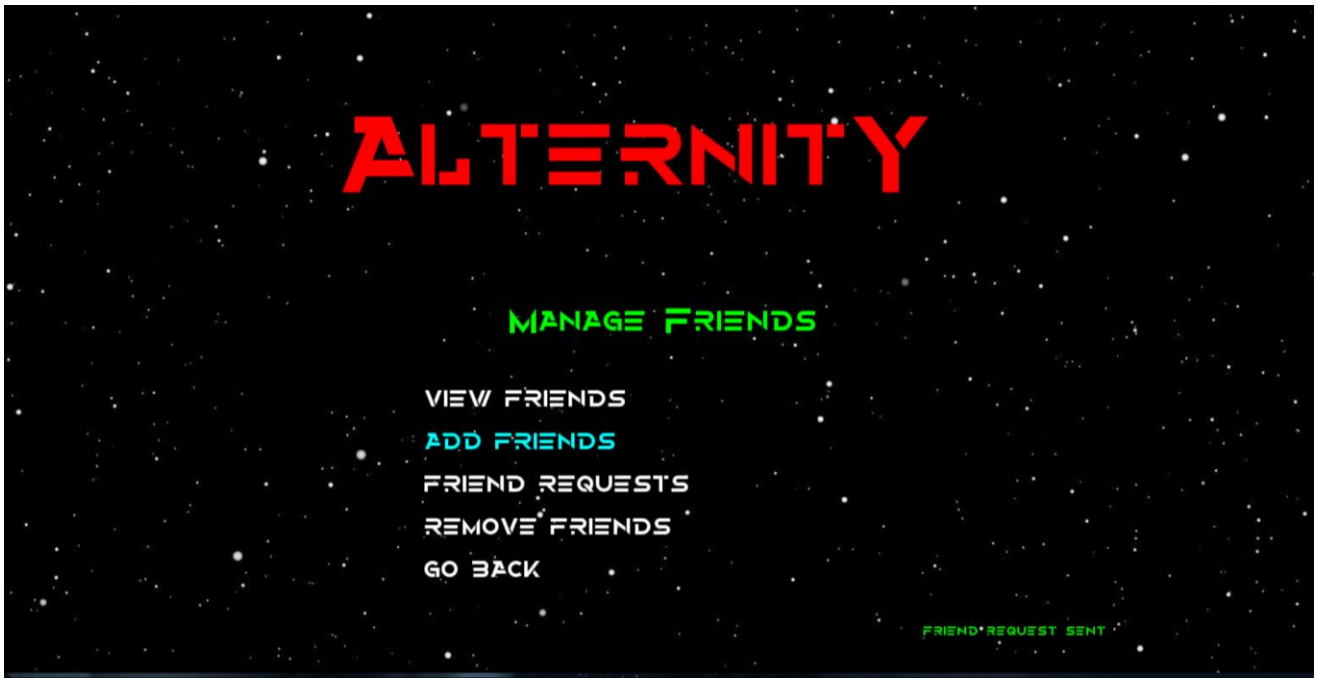


Figure 18:Friend Request Sent

On selecting the **'Friend requests'** option of the manage friends page, the **'Friend Request'** page appears.

This page allows the user to **accept (YES)** or **reject (NO)** friend requests.

Figure 19:Friend Requests Screen

If there are no friend requests for the user, **'No Pending Requests'** message appears.

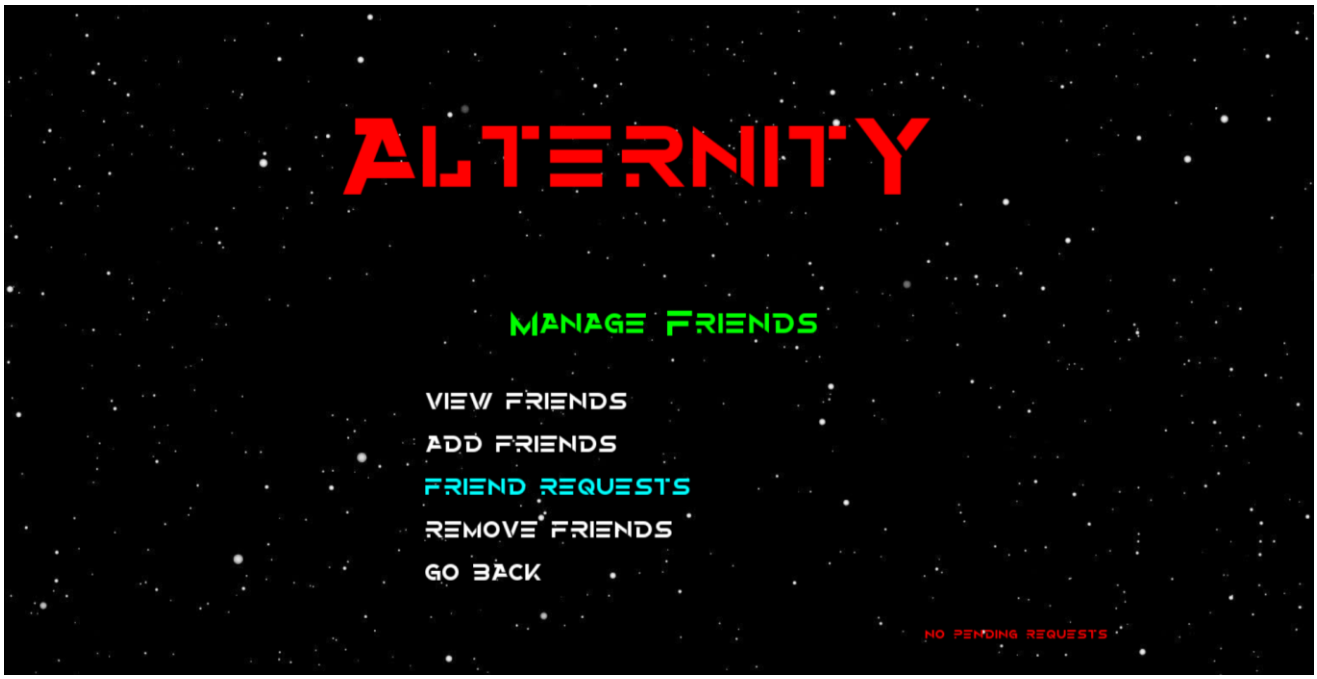


Figure 20:No Pending Requests

On selecting the **'Remove friends'** option of the manage friends page, the **'Remove Friends'** page appears.

This page allows the user to remove friends.

Figure 21:Remove Friends Screen

If a friend is removed, **‘friend removed’** message appears.

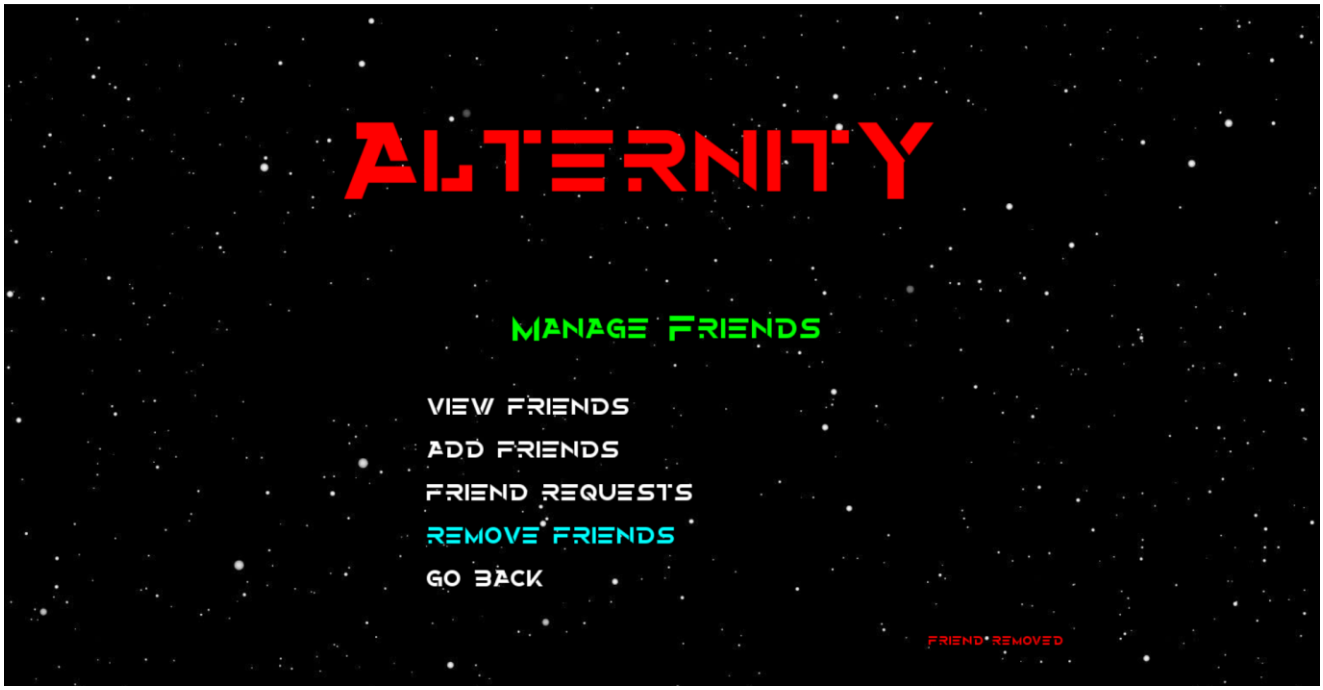


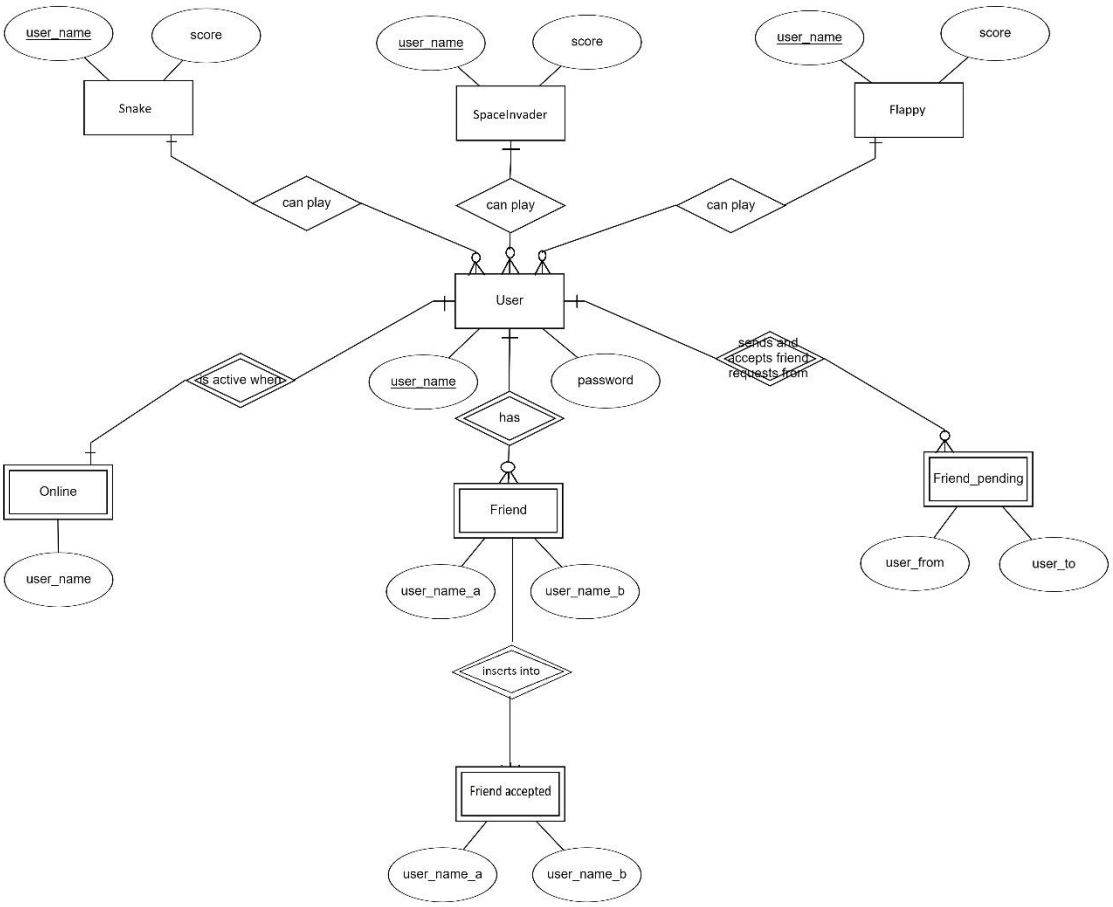
Figure 22:Friend Removed Popup Screen

The **‘Notification screen’** automatically pops up (if there are any notifications) whenever the player goes to the main menu page.

This page displays information if the user’s friend request was accepted or if the user has a pending request or if the user’s high score gets beat by someone else.

Figure 23:Notification Screen

Database Design



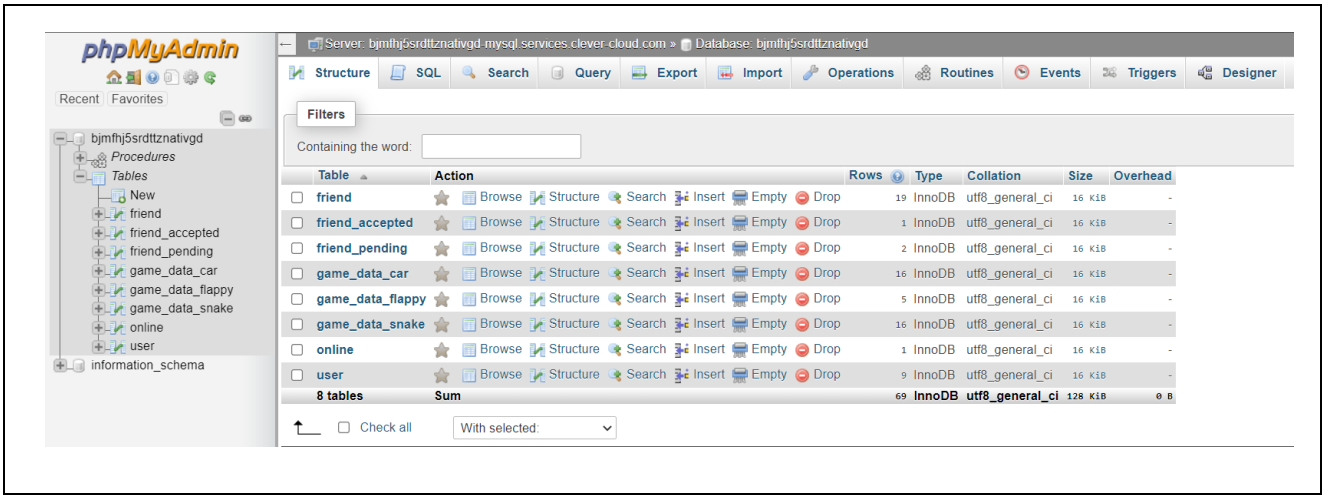
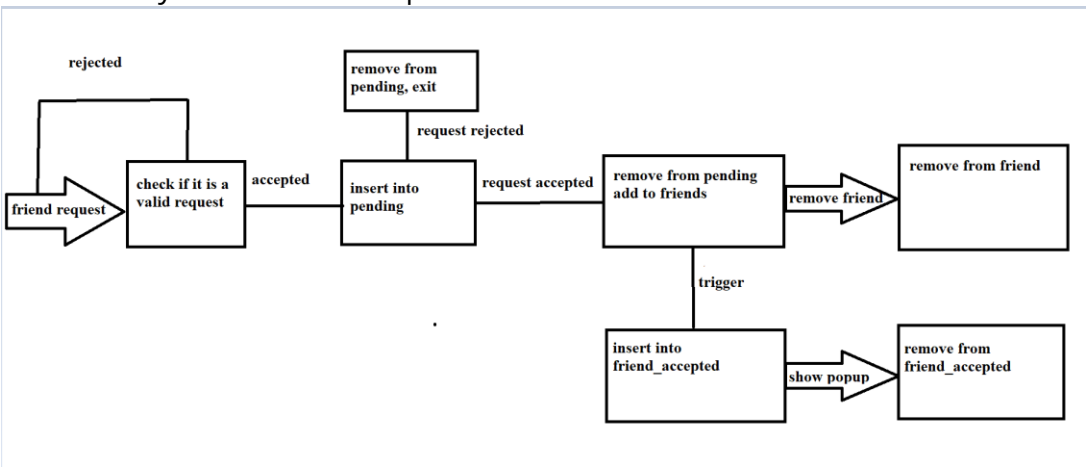


Figure 24:Database Tables

Tables:

- User : user id and password of the players.(The player names are unique like in all games, so the primary key is the user name)
- Friend : this table stores the pair user_name a and user_name_b where a is a friend of b and vice versa
- Friend_pending : this table stores all the pending requests
- For both of these tables(Friend and Friend_pending), user_name_a, user_name b is the composite key, so the pair is unique
- Friend_accepted : stores the pair user_name_a, user_name_b (b has accepted a's request)
- Whenever an entry is inserted into friend, a trigger is triggered which inserts into friend_accepted
- So the life cycle of a friend requests looks as follows :-



- Flappy , car , snake are the game tables.
- These tables store the tuple user_name, score
- The key is the tuple itself , there are no duplicate entries
- online : stores a list of active players.
- There is a foreign key between user_name in user and all of the user attributes in friend, friend_pending, friend_accepted, user_name in all the game tables and in online.

Normalizations:

1 nf :- In all the tables, all cells are atomic, ie there are no multivalued attributes. There are no duplicate rows either.

So, it is in 1 nf

2 nf :- User :- is a relation with a single attribute as a primary key, is automatically in 2 nf. This is because it is not possible to have a partial dependency on a set with just 1 element.

The friends table :- The composite key in the friends table covers all the attributes, so there are no non-prime attributes. So, there cannot be any non-prime attribute which is partially dependent on the composite key.

The game tables :- The composite key covers all the attributes, so it is in 2nf

The online table :- The composite key covers all the attributes, so it is in 2nf

3nf:-In the user table, the only functional dependency is user_name -> password . So, there are no transitive dependencies.

In all other tables the number of functional dependencies is more than one due to which there cannot be any transitive dependencies

Bcnf:-

For the functional dependency user_name --> password, the left side ie, user_name is a key. So the user table is in bcnf.

For all other tables there aren't any functional dependencies where the left side isn't the key.

So all the relations are in bcnf.

Procedures:

The procedure given below selects the high score for a particular game.

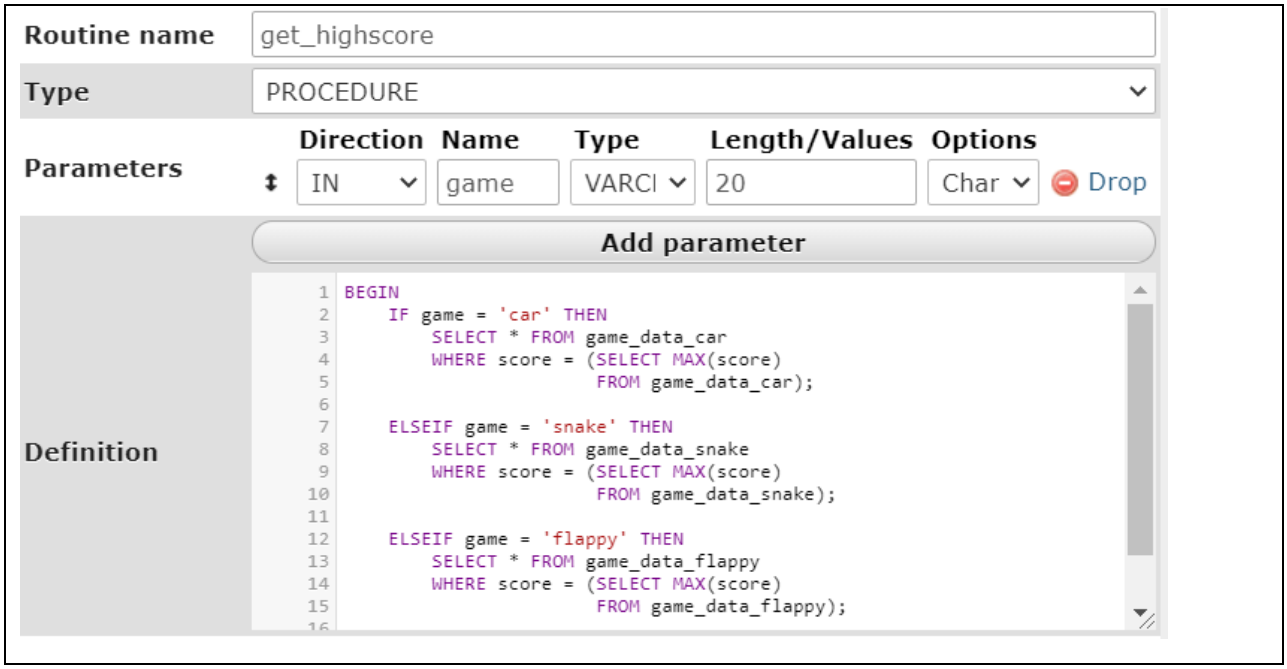


Figure 25:Procedure for printing high score

The procedure given below is used to notify a user when their friend request has been accepted by another user.

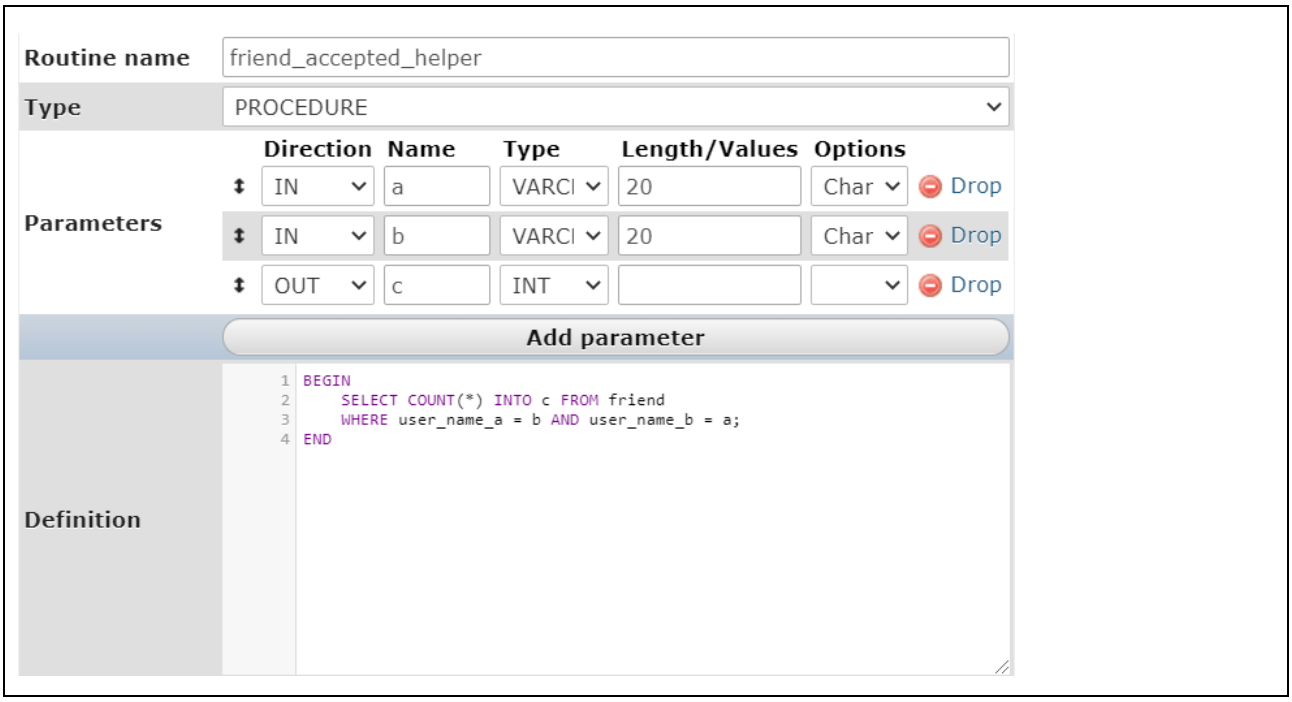


Figure 26:Procedure to notify user when friend request is accepted

Trigger:

The code snippet given below creates a trigger which notifies a user when the user's friend request has been accepted .

Code:

```
BEGIN

CALL friend_accepted_helper(NEW.user_name_a, NEW.user_name_b, @c);

IF(c = 0) THEN
INSERT INTO friend_accepted VALUES(NEW.user_name_a, NEW.user_name_b);
END IF;
END;
```

Server End

The back end, which is the crux of this project and is what provides multiplayer functionality is implemented using a mysql database and the mysql connector module, which provides an API to access the database through python.

Establishing Connection With The Database:

The function given below establishes a connection with the database and returns this connection as an object of the mysql.connector module.

```
def create_server_connection(host_name, user_name, user_password) :

    #closing any existing connections
    connection = None

    #trying to connect to mysql database with given credentials
    try :
        connection = mysql.connector.connect(
            host = host_name,
            user = user_name,
            passwd = user_password
        )
        print("MySQL Database connection successful")

    # cannot connect
    except Error as err :
        print(f"Error : '{err}'")

    #returning a connection object
    return connection
```

Figure 27:Establishing connection with the database

Inserting into USER Table:

The function given below inserts the username and password of a new user into the USER table.

```
def insert_into_user(connection, user_name, password) :

    string1 = "INSERT INTO user VALUES('"
    string2 = "', '"
    string3 = "')"
    query = string1 + user_name + string2 + password + string3

    execute_query(connection, query)

    insert_into_friend(connection, user_name, user_name)
```

Figure 28:Inserting into user Table

Inserting into GAME DATA Table:

The function given below inserts the username and the score for a particular game into the GAME DATA table.

```
def insert_into_game_data(connection, user_name, score, game) :

    flag = check_tuple_game_data(connection, user_name, score, game)

    if flag == True:
        return

    string1 = "INSERT INTO game_data_" + game + " VALUES('"
    string2 = "', '"
    string3 = "')"
    query = string1 + user_name + string2 + str(score) + string3;

    execute_query(connection, query)
```

Figure 29:Inserting into game data table

Inserting into Friend Table:

The function given below checks if a tuple is already present(if the user is already a friend) and if not it inserts the usernames of the two users as a tuple (user1,user2) into the FRIEND table.

```
def insert_into_friend(connection, user_name_a, user_name_b) :

    flag = check_tuple_friend(connection, user_name_a, user_name_b)

    if flag == True :
        return

    string1 = "INSERT IGNORE INTO friend VALUES('"
    string2 = "', '"
    string3 = "')"

    query1 = string1 + user_name_a + string2 + user_name_b + string3
    query2 = string1 + user_name_b + string2 + user_name_a + string3

    execute_query(connection, query1)
    if not user_name_a == user_name_b :
        execute_query(connection, query2)
```

Figure 30:Insert into friend table

Inserting into Pending Table:

The function given below checks if certain conditions are satisfied (user is already a friend , user is trying to add himself as a friend , user has already sent a friend request) and if none of the conditions are satisfied , the two usernames are inserted into the PENDING table.

```
def insert_into_pending(connection, user_name_a, user_name_b) :

    #same name
    if user_name_a == user_name_b :
        return -3

    flag = check_tuple_friend(connection, user_name_a, user_name_b)

    #friend already added
    if flag :
        return -1

    #alerady requested
    flag = check_tuple_pending(connection, user_name_a, user_name_b)

    if flag :
        return -2

    string1 = "INSERT IGNORE INTO friend_pending VALUES('"
    string2 = "', '"
    string3 = "')"

    query1 = string1 + user_name_a + string2 + user_name_b + string3

    execute_query(connection, query1)

    return 1


    try :
        cursor.execute(query)
        print("Database created successfully")
    except Error as err :
        print(f"Error : '{err}'")
```

Figure 31:Insert into Pending table

Creating the database:

The function given below creates a database.

```
def create_database(connection, query) :
    cursor = connection.cursor(buffered = True)

    try :
        cursor.execute(query)
        print("Database created successfully")
    except Error as err :
        print(f"Error : '{err}'")
```

Figure 32:Create database

Execute a Query:

The function given below executes the query passed as an argument to the same.

```
def execute_query(connection, query) :
    cursor = connection.cursor(buffered = True)

    try :
        cursor.execute(query)
        connection.commit()
        #print("Query executed successfully")
    except Error as err :
        print(f"Error : '{err}'")
```

Figure 33:Execute Query

Read a Query:

The function given below reads the query passed as an argument to the same and returns the result.

```
def read_query(connection, query) :
    cursor = connection.cursor(buffered = True)
    result = None

    insert_into_online(connection, "dummy42069")

    try :
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as err :
        print(f"Error : '{err}'")
```

Figure 34:Read A Query

Validate Password:

The function given below verifies whether the password entered by a user is correct and returns an error if there is no match in the database

.

```
def validate_password(connection, username, password) :
    # return 0 on match
    # return -1 on wrong password
    # return 1 on usernmae not found

    query1 = "SELECT * FROM user WHERE user_name = '" + username + "'"
    result = read_query(connection, query1)

    if not len(result) == 1 :
        return 1

    query2 = "SELECT * FROM user WHERE user_name = '" + username + "' AND passwor
d = '" + password + "'"
    result = read_query(connection, query2)

    if not len(result) == 1 :
        return -1

    return 0
```

Figure 35:Validate a password

Validate Friend:

The function given below verifies whether the friend(to be added) is a valid user(present in the user table).

```
def validate_friend(connection, username):
    # return 0 on match
    # return -1 on friend not found

    query1 = "SELECT * FROM user WHERE user_name = '" + username + "'"
    result = read_query(connection, query1)

    if len(result) == 1:
        return 0

    else :
        return -1
```

Figure 36:Validate Friend

Friends LeaderBoard:

The function given below gets the leaderboard with the user's friends.

```
def get_leaderboard_friends(connection, user_name, game) :

    query = """SELECT DISTINCT user_name, MAX(score) as score FROM game_data_"""
+ game +  """ g
               INNER JOIN friend f ON g.user_name = f.user_name_b and f.user_name
_a = '"""
    query += user_name + "' GROUP BY user_name ORDER BY score DESC"

    result = read_query(connection, query)

    return result
```

Figure 37:Friends Leaderboard

Global LeaderBoard:

The function given below gets the leaderboard with all players.

```
def get_leaderboard_global(connection, game) :

    query = """SELECT DISTINCT user_name, MAX(score) as score FROM game_data_"""
+ game + """ GROUP BY user_name ORDER BY score DESC"""
    result = read_query(connection, query)

    return result
```

Figure 38:Global Leaderboard

Get Friends List:

The function given below gets the friends list of a user in batches of 5 from the FRIEND table.

```
def get_friends_list(connection, username, start):

    query = "SELECT user_name_b FROM friend WHERE user_name_a = '" + username
    query += "' and user_name_b != '" + username
    query += "' ORDER BY user_name_b ASC LIMIT 5 OFFSET " + str(start)

    result = read_query(connection, query)

    return result
```

Figure 39:Friends List

Delete From Friends:

The function given below deletes a tuple from the FRIEND table .

```
def delete_from_friends(connection, user_name_a, user_name_b) :

    query = "DELETE FROM friend WHERE user_name_a = '" + user_name_a + "' and use
r_name_b = '" + user_name_b + "'"
    execute_query(connection, query)
```

Figure 40:Delete From Friends

Get Pending Requests:

The function given below gets the pending friend requests for a user from the PENDING table.

```
def get_pending_requests(connection, username):

    query = "SELECT user_from FROM friend_pending WHERE user_to = '" + username + "'"

    result = read_query(connection, query)

    return result
```

Figure 41:Get Pending Requests

Remove From Pending:

The function given below gets removes a user from the PENDING table when the request is either accepted or rejected.

```
def remove_from_pending(connection, user_from, user_to):
    query = "DELETE FROM friend_pending WHERE user_from = '" + user_from + "' and
 user_to = '" + user_to + "'"

    execute_query(connection, query)
```

Figure 42:Remove From Pending

Insert into ONLINE:

The function below inserts a user into the online table after logging in.

```
def insert_into_online(connection, user_name) :
    string1 = "INSERT IGNORE INTO online VALUES('"
    string2 = "')"

    query = string1 + user_name + string2
    execute_query(connection, query)
```

Figure 43:Insert Into Online

Delete From ONLINE:

The function below removes a user from the online table on logging off.

```
def delete_from_online(connection, user_name) :
    query = "DELETE FROM online WHERE user_name = '" + user_name + "'"

    execute_query(connection, query)

    print(user_name, " signing out")
```

Figure 44:Delete From Online

Check if ONLINE:

The function given below checks if the given user is online.

```
def check_if_online(connection, user_name) :

    query = "SELECT * FROM `online` WHERE user_name = '" + user_name + "'"

    result = read_query(connection, query)

    if result == [] or result == None :
        return False

    return True
```

Figure 45:Check If online

Call Procedure:

The function given below is used to call a procedure.

```
def call_procedure(connection, procedure, args) :
    cursor = connection.cursor()
    cursor.callproc(procedure, args)

    result = []

    for row in cursor.stored_results():
        result.append(row.fetchall())

    return result
```

Figure 46:Call Procedure

Conclusion

Since everyone are stuck at home, video games are one of the only sources of entertainment. And this app will also allow users to connect and play with friends.

It is absolutely essential that people realise the situation we are in and stay at home to help curb the spread of the virus.

People need to stay at home, stay safe as much as possible and get vaccinated as soon as possible.

References

https://dev.mysql.com/doc/connector-python/en/

-mysql.connector documentation

https://www.pygame.org/docs/

-pygame documentation

https://stackoverflow.com/

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