



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

Faculty of Technical and
Human Sciences – Târgu-Mureș

Tic-Tac-Toe

Made by:

Dávid Arthur

Kelemen Dániel

Leading teacher:

Szántó Zoltán

Contents

1. Introduction	2
2. Goals	3
3. Requirements	3
3.1 User requirements	3
3.2 Use cases	4
3.3 System requirements	5
3.3.1 Functional requirements	6
3.3.2 Non-functional requirements	6
3.3.3 Checked minimum requirements	6
4. System Design & Architecture	7
4.1 System architecture	7
4.2 Technical Details	8
4.2.1 Gesture recognition	9
4.2.2 Mouse Tracker	10
4.2.3 Server connector module and message processor	11
4.2.4 Base game	12
4.2.5 Back-end	12
4.2.6 UI	13
4.2.7 Activity diagram	13
4.3 UI Plan	15
4.4 Management	15
5. Application walkthrough	16
6. Conclusion	23
6.1 Further development possibilities	23

1. Introduction

People observe their environment through their senses. We have five main senses: sight, hearing, smell, taste, touch. The most important of these is sight. The eye delivers 80 percent of the information we need to perceive the world around us. We recognize colors, shapes, movements with the help of our eyes. It recognizes gestures as well, so does our game, which is Tic-Tac-Toe, but not just the simple game, it is controlled with hand gestures. The game uses a camera to detect the movements of the player's hand. Tic-Tac-Toe is a simple game, but we made it more fun with this feature, and also you don't have to be close to your friend, because now you can play it through the internet.

2. Goals

Our goals for this project were to make an enjoyable, but easy to use game. We wanted it to be a game which can be played to pass time and it doesn't involve much preparation and learning the mechanics of the game, or a high-end electronic device.

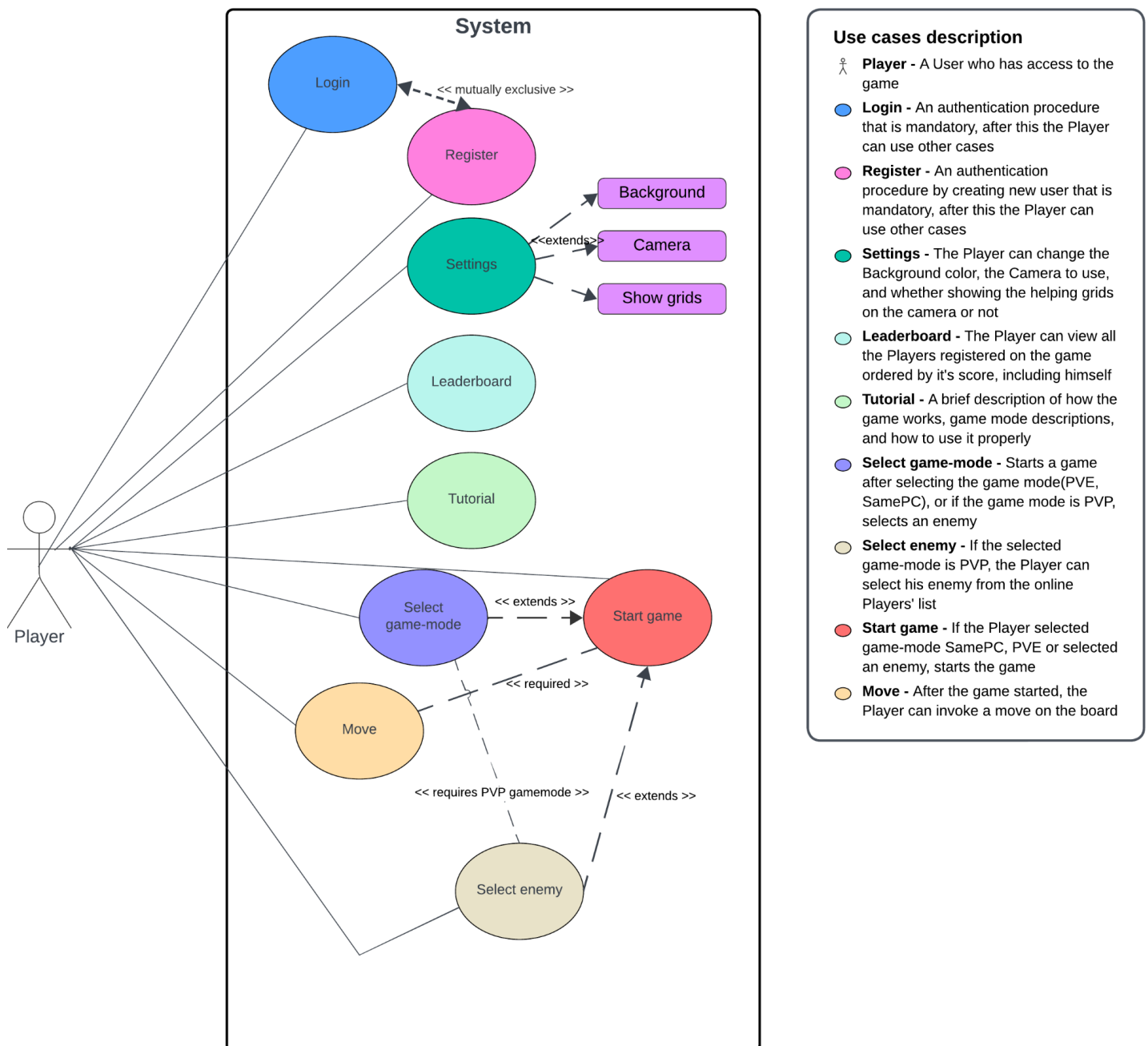
3. Requirements

3.1 User requirements

In order to play our game, the user needs to register a new account or if he has one already, he needs to log-in. To register, the user needs to fill out a registration form with a username and a password. The user must be aware that the credentials are encrypted, and his public IP address is used for logging purposes. The credentials provided by the Player should be noted down, because at the time the game doesn't have a password recovery feature.

After the Login/Registration procedure, the player can adjust the game settings in the ‘Settings’ page, and after that he can start a game, from where he should choose the game mode he wants to play.

3.2 Use cases



Structural Description

Title : Login

Actor : Player

Precondition :

1. Has internet access
2. Has access to a user previously registered in

Postcondition :

1. The login credentials were correct

Normal flow :

1. Enters the username and password
2. Click login button

Alternate flow :

**// Incorrect credentials
Until correct credentials**

1. Re-enters the credentials
2. Clicks Login button

Exceptional flow :

// If the server is offline

1. Pop's up a message informing the player to retry later

Title : Start game

Actor : Player

Precondition :

1. The Player logged in
2. Has selected game-mode

Postcondition :

//If gamemode PVP

1. The enemy accepted the invitation to play

Normal flow :

//If The gamemode is PVE, or SamePC

1. The board window shows up
2. The Player is announced who starts

//If gamemode is PVP

1. The gamemode is PVP
2. The Player selects the enemy from the list
3. The board window shows up
4. The Player is announced who starts

Alternate flow :

1. Player clicks Cancel
2. Returns to main menu

Exceptional flow :

// If the enemy refuses

1. Pop's up a message informing the player that the enemy refused
2. Returns to main-menu

Title : Settings

Actor : Player

Precondition :

1. The Player logged in
2. The Player is not in game

Postcondition :

-

Normal flow :

1. The Player changes background
2. The Player selects the camera
3. The Player choose to show the Grid's or not

Alternate flow :

1. The Player click's Back
2. Returns to main menu

Exceptional flow :

// If the camera is not available anymore

1. Pop's up a message informing the player that the camera is not available

3.3 System requirements

The system requirements vary according to the way the user wants to play the game. In order to play online the player needs an internet connection, if he wants to control the game with hand gestures, the system has to be connected to a camera.

3.3.1 Functional requirements

- User authentication: The Game allows players to save their scores, and use them on multiple devices(not simultaneously). Also saves their unique username that the Player can select.
- Personalized game experience: Is allowed to change the Background color of the game in the ‘Settings’ menu, and also if the player has multiple Webcams connected, they can use which they want. If using a camera, the grid view can be enabled in the view.
- Multiple game mode types: The game provides multiple game modes that you can choose from, and play it: PVE(Player vs. Environment), SamePC(2 players on the same device), PVP(Player vs Player)
- Player selection: The Player can choose its enemy if he plays PVP. He can choose one from the list of Online-Players
- Leaderboard: The game provides possibility to check the all-time all-player leaderboard
- Tutorial: The new Players can read a brief walkthrough of the game explaining the base mechanics, and game modes.

3.3.2 Non-functional requirements

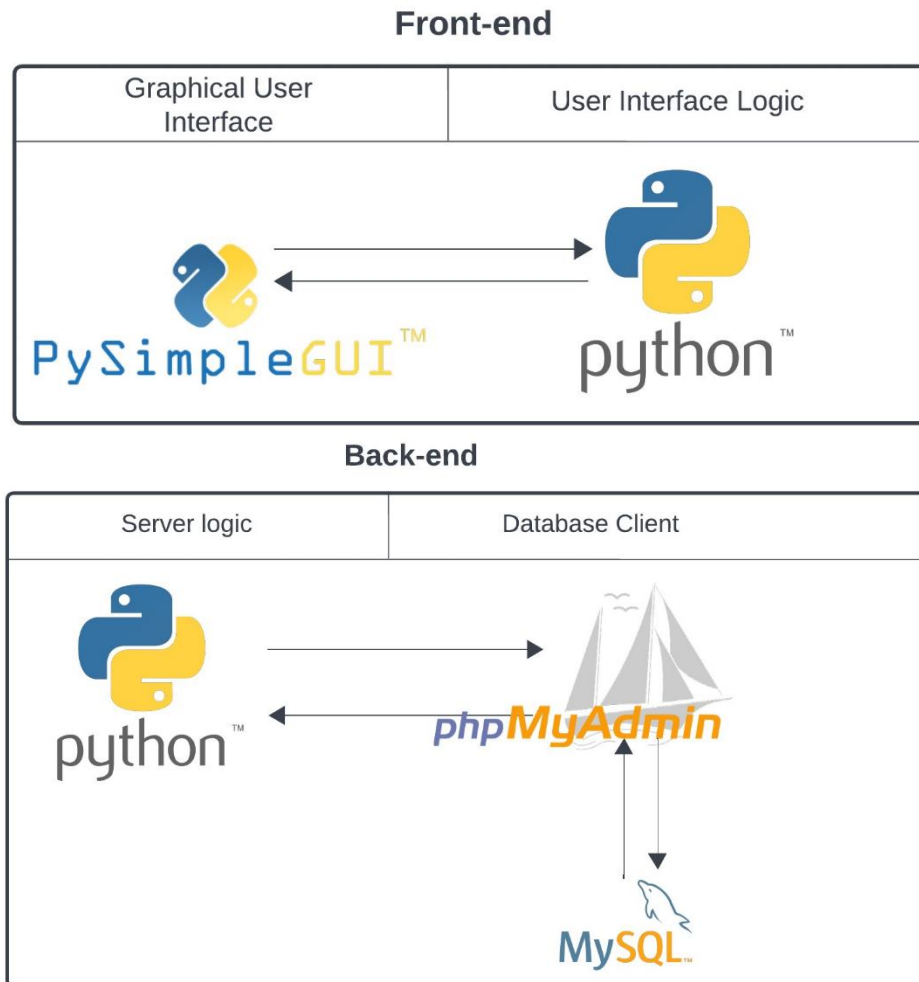
- PC or Laptop running Linux/Windows OS
- Stable internet-connection
- Python 3.10 release or higher
- Storage space: 450 MB at least
- Memory: 512 MB RAM at least
- Dedicated or Integrated Webcam with a minimum resolution of 480x640
- Dedicated or Integrated Graphics Card

3.3.3 Checked minimum requirements

- OS: Windows 10 Pro x64
- CPU: Intel i3-10110U
- GPU: Integrated Intel UHD Graphics
- Memory: 8 GB RAM
- Integrated Webcam with a resolution of 1280x720
- Internet speed: 60 Mbps

4. System Design & Architecture

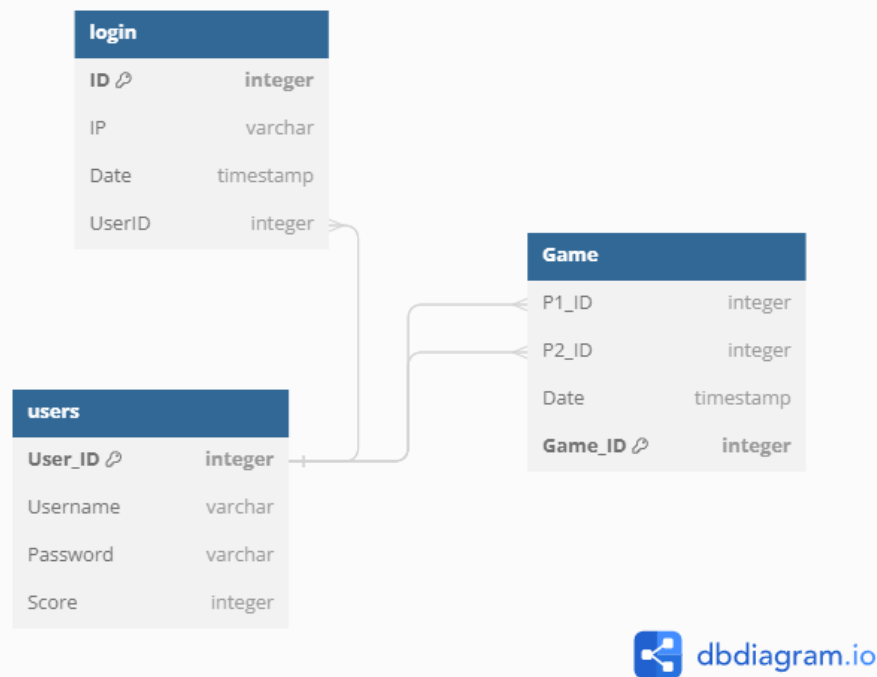
4.1 System architecture



As the figure shows above, we used Python as our main programming language, because it had the most libraries we needed to create our game, and we didn't have to go on low-level programming.

For realizing the GUI(Graphical User interface) we used PySimpleGUI, which was easy to use and enough for what we needed it to be.

For creating the database we used MySQL. It is a fast database management system and has good connectivity, easy to use, which was needed for us, because the game communicates with a server through the internet. You can see our database plan below:



To integrate the database with Python, we created an ORM(Object–relational mapping) using SQLAlchemy.

4.2 Technical Details

When the game starts, it starts to initialize the modules, and establish the connection to the server in the following order:

1. Connecting to the server(mandatory)
2. Operating Gesture Recognition(Optional if camera available)
3. Mouse position tracker(mandatory)
4. Graphical User Interface(mandatory)

The initialization of these modules is optimized with parallel computing, each of these modules running on different threads, but the GUI which is running on the main thread. When a program stop signal is received, the main thread calls a function that sends out for each Thread a stop signal, and wait for them to stop.

```

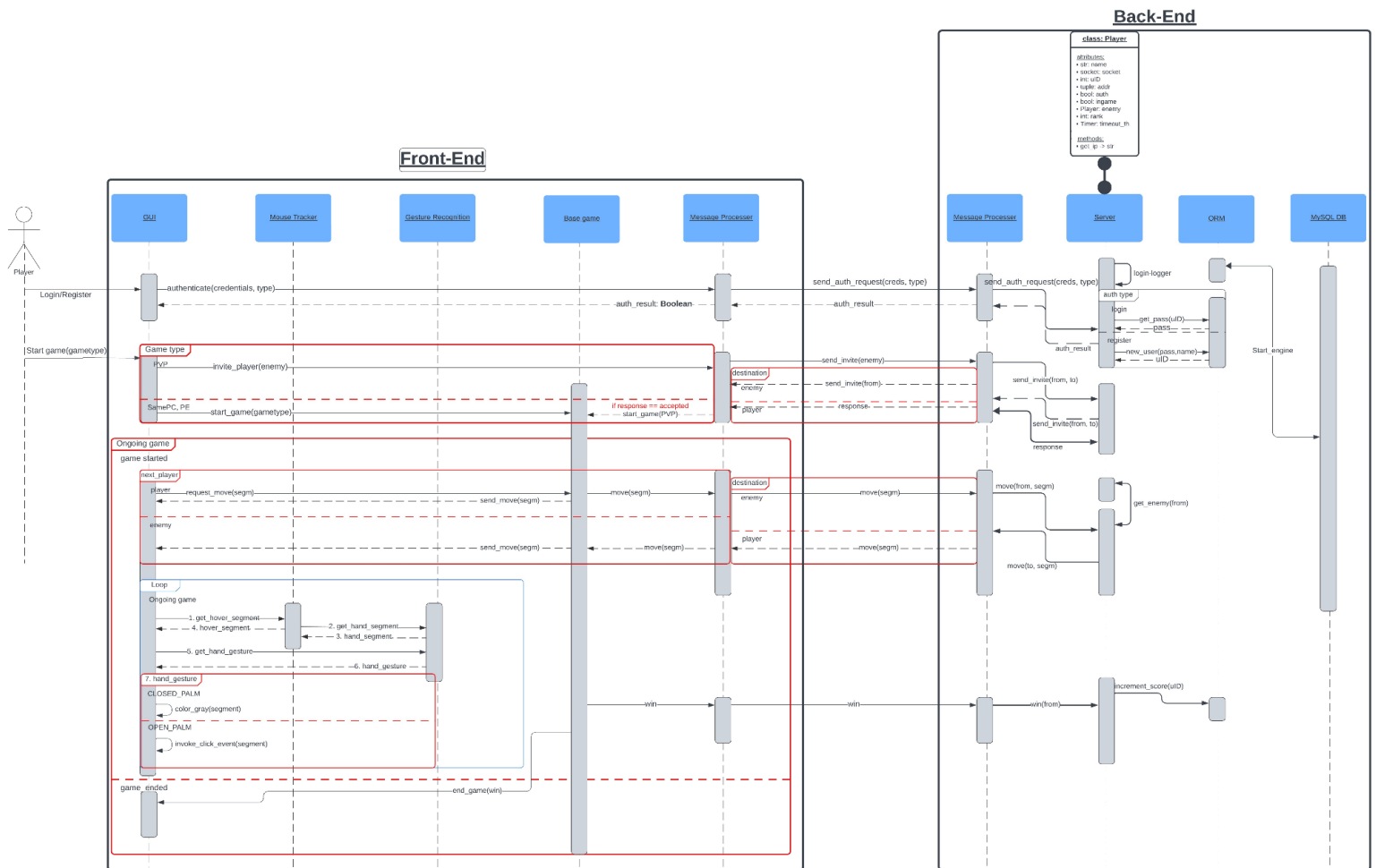
def stop_program(): # The main function to stop the entire game. Called when exiting a window, or an error
    global program_stop_called
    if program_stop_called:
        return

    program_stop_called = True
    client.stop_connection()
    stop_recognition()
    mouse_tracker.stop_hover_segment()

    if cam_t is not None:
        cam_t.join()
    if mouse_t is not None:
        mouse_t.join()
    if server_t is not None:
        server_t.join()

    print('The program shut down!')
    exit(0)
  
```


In the picture below we a Sequence diagram of the modules is presented:



4.2.1 Gesture recognition

This module makes the game special. The hand recognition is based on a library called 'mediapipe' developed by Google, that is based on Tensorflow image recognition, and trained with a lot of data. This module recognizes the landmarks of the hand, and returns it as an array, which is later used to get the gesture used for controlling our game. The Gesture detection is using calculations based on observations to define which gesture is more appropriate to the real one. The video frames are provided in the correct format with the OpenCV library. Here we can see the main process:

```

cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame = cv2.flip(frame, flipCode: 1) # Mirroring the camera
if not show_grid:
    raw_frame = frame.copy() # Making a copy for the visualized frame

rec_img = hands_detector.process(frame) # Initiate hand recognizer on the frame

if rec_img.multi_hand_landmarks: # If hand was detected
    hand_landmarks = rec_img.multi_hand_landmarks[0] # Take the first hand
    curr_gesture, hand_segm = recognize_gesture(hand_landmarks, frame) # Run the gesture recognizer on the frame

```

And also the gesture recognition part:

```

def recognize_gesture(hand_landmarks, frame):
    global previous_gesture

    pos_x, pos_y = calc_median_pos(hand_landmarks, frame) # Calculates the midpoint of the hand

    segm = calc_hand_segment(pos_x, pos_y) # Calculates in which segment the hand is located

    draw_frame(frame, pos_x, pos_y, hand_landmarks=hand_landmarks, show_grid=True, show_dot=True, show_segment=True)

    landmarks = hand_landmarks.landmark

    # Determines whether a hand is closed, or open
    if get_angle_between_fingers(landmarks[0], landmarks[4], landmarks[8]) < 19 and \
        get_angle_between_fingers(landmarks[0], landmarks[8], landmarks[12]) < 8 and \
        get_angle_between_fingers(landmarks[0], landmarks[12], landmarks[16]) < 6 and \
        get_angle_between_fingers(landmarks[0], landmarks[16], landmarks[20]) < 8:
        gesture = CLOSED_PALM
        previous_gesture = CLOSED_PALM

    elif get_angle_between_fingers(landmarks[0], landmarks[4], landmarks[8]) > 23 and \
        get_angle_between_fingers(landmarks[0], landmarks[8], landmarks[12]) > 9 and \
        get_angle_between_fingers(landmarks[0], landmarks[12], landmarks[16]) > 7 and \
        get_angle_between_fingers(landmarks[0], landmarks[16], landmarks[20]) > 10:
        previous_gesture = OPEN_PALM
        gesture = OPEN_PALM

    else:
        gesture = previous_gesture

    return gesture, segm

```

4.2.2 Mouse Tracker

The Mouse Tracker module is constantly looking for the position of the GUI window, and the absolute mouse position on the screen, and with some calculations it returns a segment number, which determines on which cell is the mouse located on the 3x3 grid of the game. This connects with the Hand Recognizer, and combines the hand positions too. The Hand position has the higher priority.

```

button_positions = [ # The positions of the 3x3 grid buttons on the game, ABSOLUTE VALUES
    {'top_left': {'x': 0, 'y': 0}, 'bottom_right': {'x': 0, 'y': 0}}, # 0 - Only for indexing
    {'top_left': {'x': 30, 'y': 192}, 'bottom_right': {'x': 161, 'y': 291}}, # 1
    {'top_left': {'x': 181, 'y': 192}, 'bottom_right': {'x': 313, 'y': 291}}, # 2
    {'top_left': {'x': 334, 'y': 192}, 'bottom_right': {'x': 465, 'y': 291}}, # 3
    {'top_left': {'x': 30, 'y': 299}, 'bottom_right': {'x': 161, 'y': 399}}, # 4
    {'top_left': {'x': 181, 'y': 299}, 'bottom_right': {'x': 313, 'y': 399}}, # 5
    {'top_left': {'x': 334, 'y': 299}, 'bottom_right': {'x': 465, 'y': 399}}, # 6
    {'top_left': {'x': 30, 'y': 409}, 'bottom_right': {'x': 161, 'y': 509}}, # 7
    {'top_left': {'x': 181, 'y': 409}, 'bottom_right': {'x': 313, 'y': 509}}, # 8
    {'top_left': {'x': 334, 'y': 409}, 'bottom_right': {'x': 465, 'y': 509}} # 9
]

for i in range(1, 10): # Determining the boundaries of each cell/button, and checking on which the mouse is hovering
    x1 = button_positions[i]['top_left']['x']
    y1 = button_positions[i]['top_left']['y']
    x2 = button_positions[i]['bottom_right']['x']
    y2 = button_positions[i]['bottom_right']['y']
    if x1 <= x <= x2 and y1 <= y <= y2:
        return i

return 0

```

4.2.3 Server connector module and message processor

In the current version of the game this module is mandatory, because it provides the communication between the Game and the Server, so that 2 players can play from different devices, and the authentication of a player is also working with this. It has 4 main components:

- A method that establishes a connection to the server
- A Listener thread that listens for incoming messages
- A message processor, that processes the incoming messages to usable data, and signals
- A message sender

```

try:
    server_socket.connect((SERVER_IP, SERVER_PORT))
    print(f"Connection to the server({SERVER_IP}:{SERVER_PORT}) was successful!")
    # threading.Thread(target=check_internet_connection).start()
    Connected_To_Server.set()
    send_message('get-online-player-nb')
except ConnectionRefusedError as e:
    sendError( header: 'An error occurred in client.py/connect_to_server', 'Connection refused by the server! ' + str(e))
    return None
except Exception as e:
    if str(e) != 'timed out':
        sendError( header: 'An error occurred in client.py/connect_to_server', str(e))
    return None

listen_th = threading.Thread(target=listen_to_server) # Starting the listener function when the server is connected
listen_th.start()

```

4.2.4 Base game

This module is in the same Thread as the UI, but a different component. This guides the UI and all of the other components during the gameplay on which moves are next, or how they are executed. For example if a player clicks on the screen, it determines if the click is allowed(he's the next in line, and is empty) and after the move if the game ends, and how, and also starts the new round, and handles the main game board.

```
res = put(pos, LETTER_X if starting_player == current_player else LETTER_O) # Calls the actual putting method

win, _ = check_win() # Checking the state of the game

if game_type == GAME_PVE:
    if current_player == PLAYER_ME:
        if res == 1: # Successful, so PC turn
            current_player = switch_player(current_player)
            threading.Thread(target=request_random_step).start()
            return 1
        elif res == 2 and win == TIE: # Successful, game ended with TIE
            t = threading.Timer(interval=2, next_round, args=(True,))
            t.start()
        elif res == 2: # Successful, and ended, so comes the next round
            round_list[current_round] = current_player
            roundend_event.set()
            threading.Timer(interval=2, next_round).start()
            return 2
    return res
```

4.2.5 Back-end

The Back-end consists of a message processor that processes the requests from the clients(for example authentication, player communication), and if needed, forwards it to another player(in our case, enemy). The server keeps track of the in and out connections of each player, and stores them paired with their IP address. The storing is implemented in a MySQL database , which is connected with the Back-end through an Object Relational Mapping for easy-to-use coding. Every connected Player receives a Listener on a separate Thread for faster processing, and they are tracked by their socket, then mapped to their Player instance.

```

while True:
    try:
        client_socket, addr = server_socket.accept()
        print("Got a connection from {}".format(addr))

        send_msg(client_socket, msg: "Authenticate first!\n")
        threading.Thread(target=listen_client, args=(client_socket, addr,)).start()
    except socket.timeout:
        pass

```

4.2.6 UI

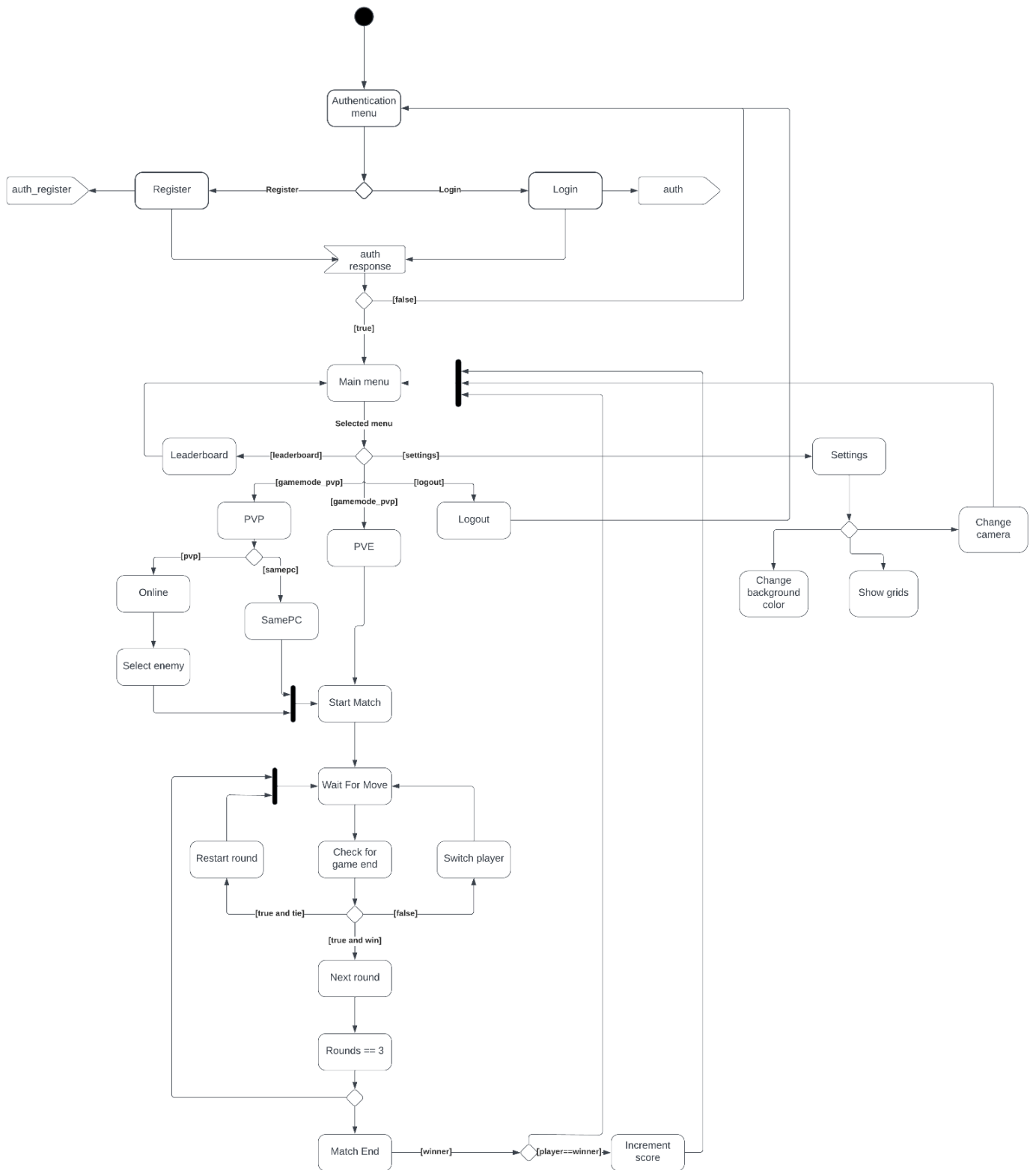
Our User Interface is implemented with a simple Graphic, using PySimpleGUI. This decides the flow of the application, depending on the Player's choice of what to do. A simple code fragment of a casual window can be seen below, and on the `4.3 UI Plan` we can see a full wireframe of the GUI.

```

def firstpage():
    global window, position
    Logout.clear()
    b1 = psg.Button(button_text: "Login", size=(30, 2))
    b2 = psg.Button(button_text: "Register", size=(30, 2))
    b3 = psg.Button(button_text: "Play as Guest", size=(30, 2))
    text1 = psg.Text(text='Tic-Tac-Toe', font=('Algerian', 50), text_color='black', background_color=bgclr)
    text2 = psg.Text(text='Players online: ', font=('Algerian', 15), text_color='black', background_color=bgclr)
    timeout = 0
    while online_players == -1 and timeout < 100:
        time.sleep(0.1)
        timeout += 1
    text3 = psg.Text(text=f'{online_players}', font=('Algerian', 15), text_color='black', background_color=bgclr)
    space = psg.Text(text='', size=(30, 8), background_color=bgclr)
    space1 = psg.Text(text='', size=(30, 1), background_color=bgclr)
    space2 = psg.Text(text='', size=(30, 1), background_color=bgclr)
    space3 = psg.Text(text='', size=(30, 8), background_color=bgclr)
    space4 = psg.Text(text='', size=(30, 8), background_color=bgclr)
    col1 = [[text1]]
    col2 = [[b1], [space1], [b2], [space2], [space4]]
    col3 = [[text2]]
    layout = [[psg.Column(col1, background_color=bgclr, justification='center')],
               [space],
               [psg.Column(col2, background_color=bgclr, justification='center')],
               [space3],
               [psg.Column(col3, background_color=bgclr, justification='r'), text3]]
    window = psg.Window(title: 'Tic-Tac-Toe', layout, size=(480, 640), background_color=bgclr,
                        element_justification='c', finalize=True)

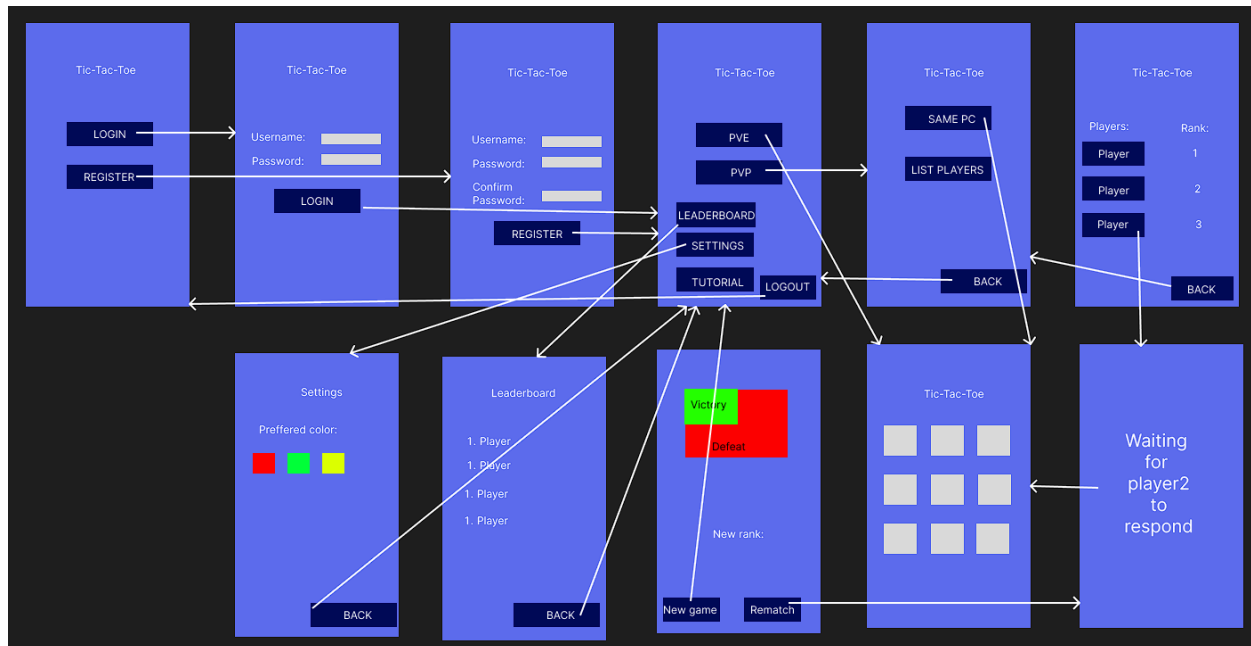
```

4.2.7 Activity diagram



4.3 UI Plan

We made the UI plan at the start of the project. It helped us so much, we had something to follow throughout the whole process of creating the UI. The design and some parts of this plan changed along the way, because we came up with new ideas and better solutions. You can see the plan below:



4.4 Management

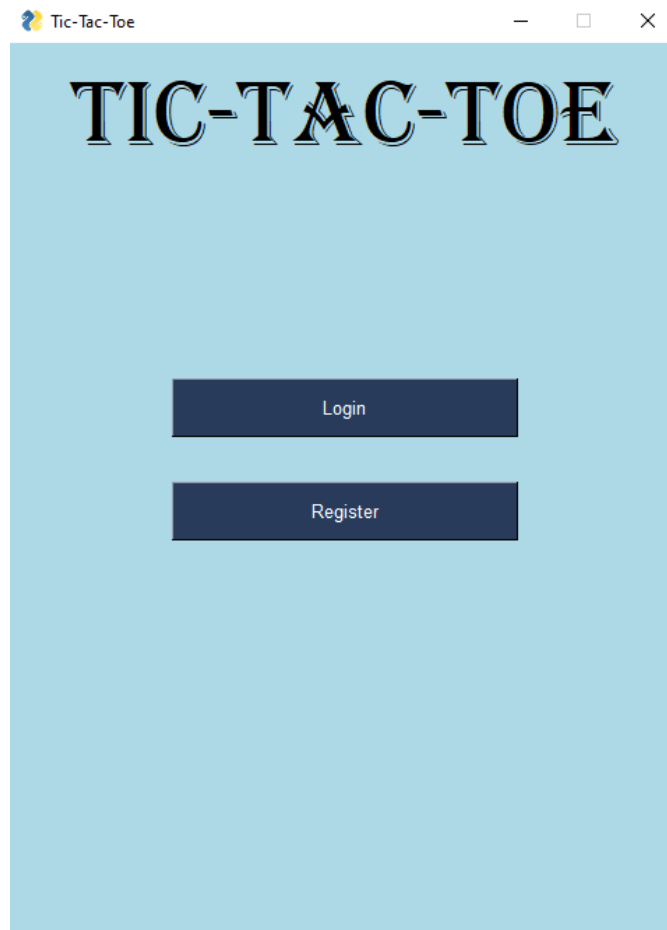
For keeping track of our work we used Trello, where we initially added the main functionalities to do, after we started to develop certain parts of the initially added functions, we added more detailed descriptions of what we needed to do. Trello board was a good way to see how we work, what we have to do and what we have done already, so the tracking of the project could go straight-forward without needing to think about what we should do next. It was also a good place to divide the work among us.

For version control we used GitHub. It is a good platform for working simultaneously on the same project. Our strategy here was to create branches for every part of our project or feature. We divided the work so that we could work at

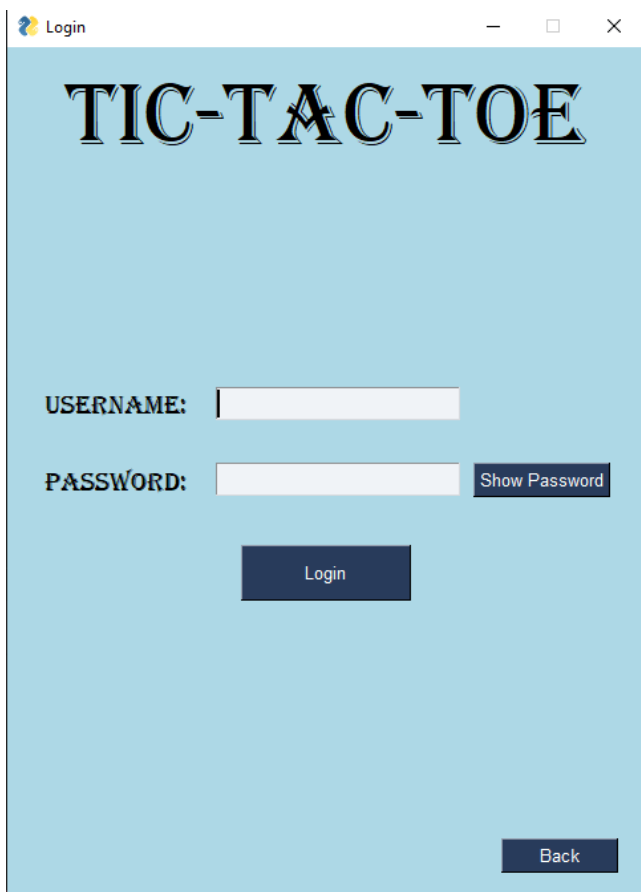
the same time. We developed two separate parts of the project at the same time so we didn't have to wait for each other.

Also, GitHub is very safe from a data-loss perspective, so we didn't have to worry about overwriting each other's codes', and also we could track every version of the project, and return to a safe version if something bad happened.

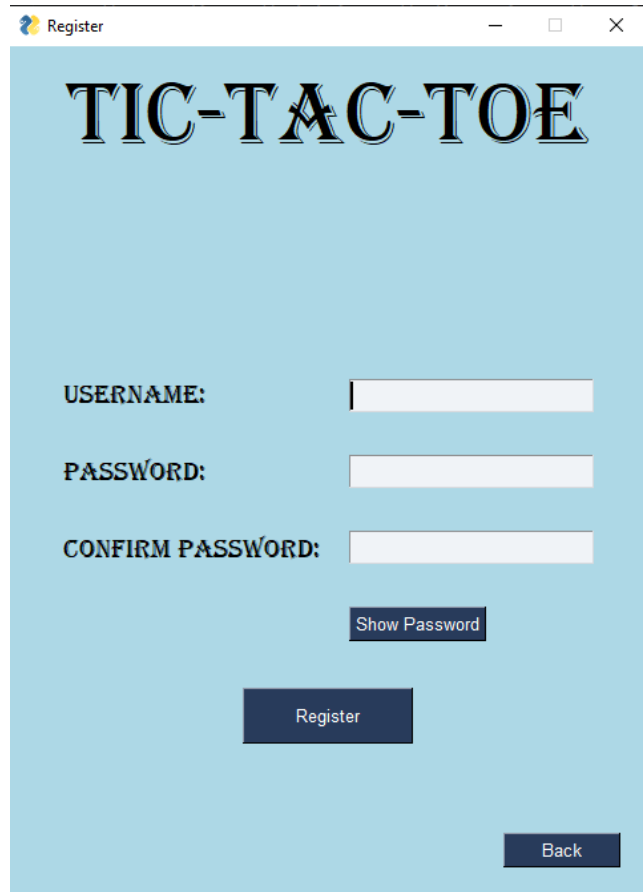
5. Application walkthrough



This is what the user sees when he starts the application. On this page he can select to register a new account or log into an already existing one. After the user selects either of these options he lands on one of the following pages:



The login page features a light blue background with the title "TIC-TAC-TOE" at the top. Below the title, there are two input fields: "USERNAME:" and "PASSWORD:". The "PASSWORD:" field has a "Show Password" button next to it. A "Login" button is positioned below the password field. At the bottom right, there is a "Back" button.



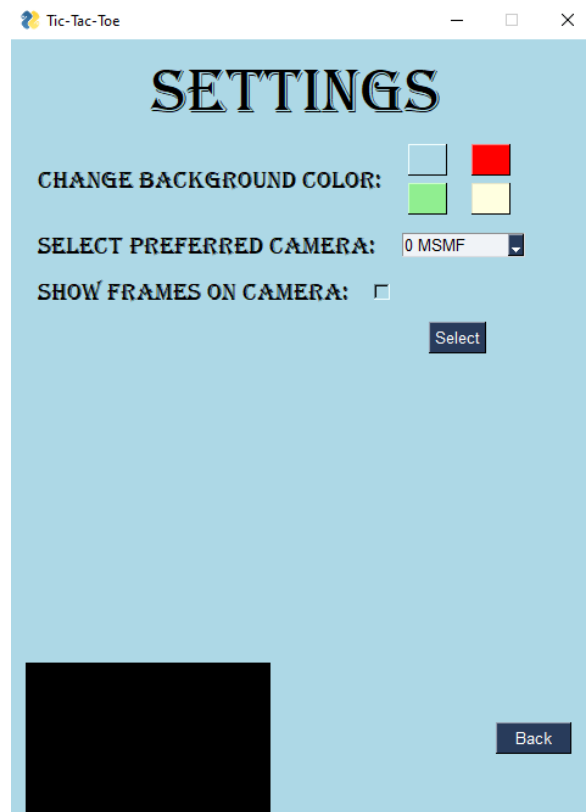
The register page features a light blue background with the title "TIC-TAC-TOE" at the top. Below the title, there are three input fields: "USERNAME:", "PASSWORD:", and "CONFIRM PASSWORD:". The "PASSWORD:" field has a "Show Password" button next to it. A "Register" button is positioned below the "CONFIRM PASSWORD:" field. At the bottom right, there is a "Back" button.

On the left you can see the login page, the user needs to give a username and a password to authenticate himself. If the user chose the register option he landed on the page on the right side. After a successful log in/register he goes to our main page:

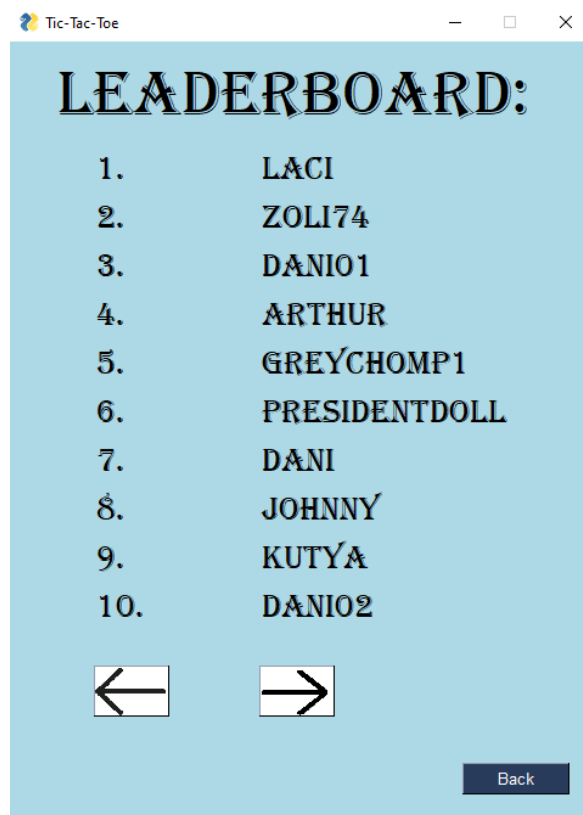


The main page features a light blue background with the title "TIC-TAC-TOE" at the top. Below the title, there is a "RANKING: 3" label. A "MODE:" label is followed by two buttons: "PVE" and "PVP". At the bottom, there are three buttons: "Settings", "Leaderboard", and "Tutorial". A "Logout" button is located at the bottom right.

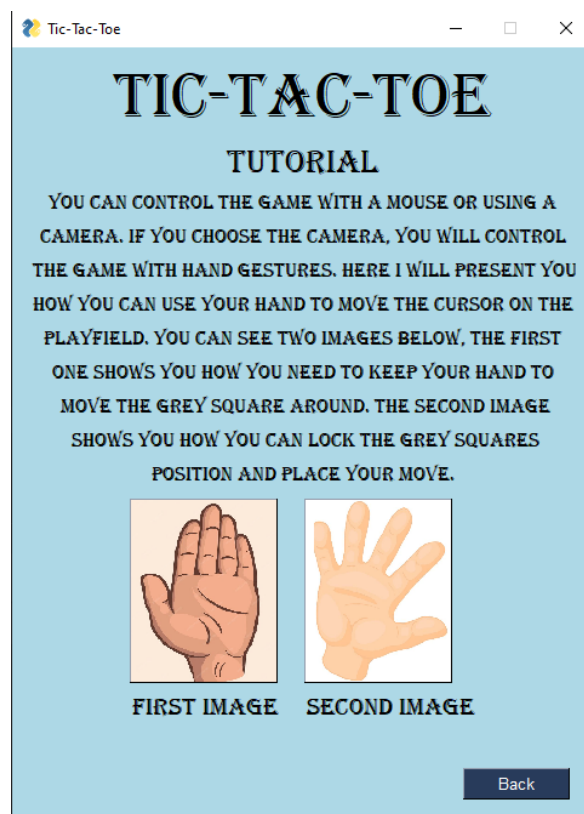
This page has the most options. The player can see his rank in the upper right corner of the screen. He can select either to play against other players or if he wants to play against the environment. The player can go into the settings page to change some options, he can choose to see the leaderboard or he can go to our tutorial to learn how to play the game. If he wants to play in another account he can log out to change his account.



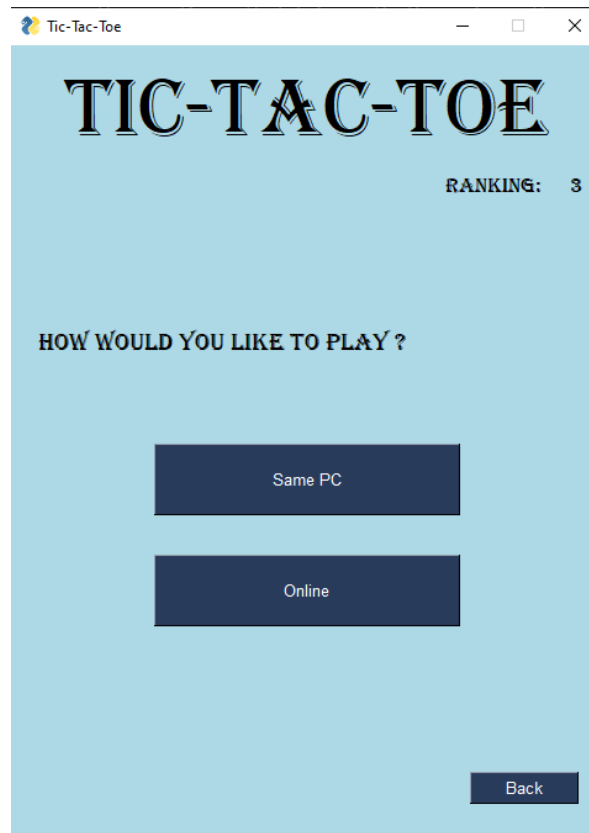
This is the settings page, the user can change his background color, he can change between cameras if he has more then one and he can enable the grid on the camera's image.



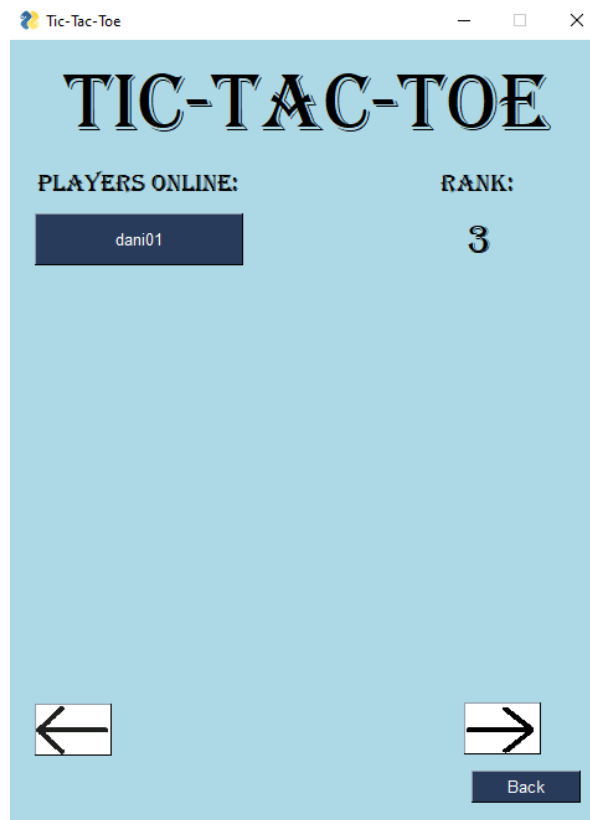
On this page the user can see the game's leaderboard, he can go to the next page by pressing the arrow pointing to the right and he can go to the previous page by pressing the arrow pointing to the left.



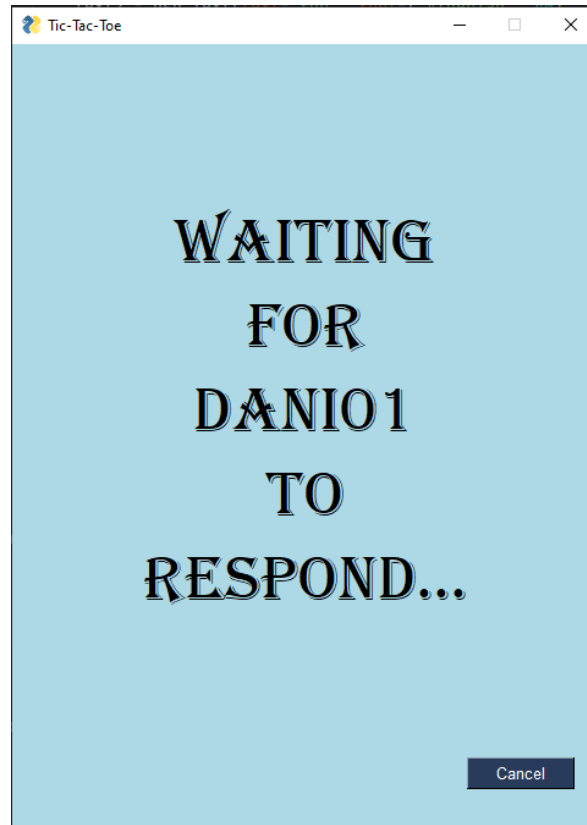
This is the tutorial page. The user can read about how the game's controls work.



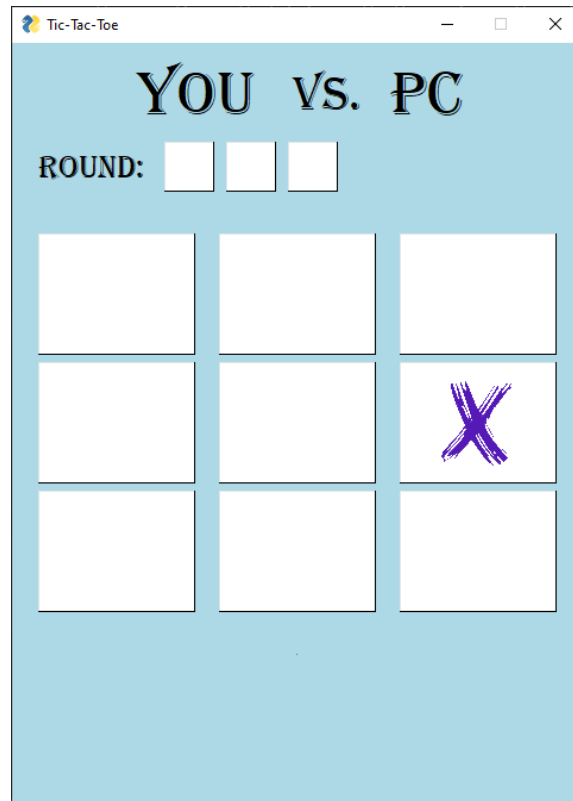
If the player chose to play PVP on the main page, this is the next window he will get to. Here he can select to play on the same pc with a friend next to him, or to play against other players online.



In order to play PVP online the player needs to select an opponent from this list. The list shows other online players.



This is the page that will appear after the player selects an opponent. He will stay here until the other player accepts his invitation. If he doesn't accept it, the player will be thrown back to the main page.



This is the page that the player will see during a game. On the upper part of the page he can see the rounds, if he wins a check mark will appear in the blank squares, if he loses an “X” will appear.



Finally this is the game outcome page. The player can see his new rank and select to play another match with the same opponent or to go back to the main page.

6. Conclusion

The application in its current version is stable, playable, so we can call it the first version. As by now the main features of the game are:

- Multiplayer, so the players can play across the internet
- Hand/gesture recognition, so the players can control the game using hand gestures

6.1 Further development possibilities

In the future we would like to extend the features of the game by offline-games, so you don't have to connect to the server, and don't need an internet connection, thus your ranks will be updated when synchronized with the server.

Another feature we would like to add is cross-platform playing. We want to make a GUI on Android so it can be played not only with a Windows/Linux operating system.

Also we would like to add in the future a password recovery feature.