UltiMaze: Ultimate territory game

Project realized by group $8B$, composed of:

Juliette FLORIN
Aurelien BRUN
Romain GEORGENTHUM
Haouran ZENG
Clement RICHARD

Project supervised by:

Massimiliano TODISCO,
Chiara GALDI,
Pasquale LISENA,
Arun PADAKANDLA

2023-2024

Contents

Abstract. In this semester project we are creating an interactive game. The game has to take every aspect of a game into consideration: the way it looks, how it is played, and against whom the game is played.

1 Objective

The objective is to create an Ultimate Tic Tac Toe game. We have to create a game using the Ultimate Tic-Tac-Toe rules. The requirements for the game are that it has to have:

- An Interface
- A way to play against other Raspberry Pi: Communication protocol between Raspberry Pi for one-to-one play
- A Bot or AI to play against
- A camera to play without a Mouse: Object tracking to control the game

The material at disposal is:

- Raspberry Pi
- Raspberry Pi Camera Module 3
- Raspberry Pi Battery
- Mouse
- Screen

To go toward this objective there are many challenges that should still be detailed.

2 Challenges

Game Rules: As the game has to respect the rules of the Ultimate Tic Tac Toe we need to code functions to respect the game's rules. Those functions must be used for all modes of play (online, offline, as an AI or as a Human).

Communication: As we need to establish a connection between two Raspberry Pi, it is needed to choose a protocol on both the Transport layer (TCP, Bluetooth, UDP...) and the Application layer (Command to send and the way it is structured). The connection protocol must ensure that the messages are well send and that it is secure so no one can steal another persons game.

The Bot or AI: The AI must take into account the time to compute and the fact that players want to play against an AI that plays fast. However, since we are working on a Rasberry Pi it is important to understand that when testing the AI it will be faster on a PC then on a Rasberry Pi. This is because the CPU and the RAM of the PC are more efficient than the ones of the Rasberry Pi.

Possible game play: Since there will be multiply types of players and as later, we will see that the communication pillar decided to take Guest and Host communication we have multiple scenarios possible:

- Local
  - AI vs Human
  - Human vs Human
- Online
  - Host AI vs Guest Human
  - Host AI vs Guest AI
  - Host Human vs Guest AI
  - Host Human vs Guest Human

3 Rules of the game

To compute a game it is essential to understand the basic rules of the Ultimate Tic Tac Toe.

- Ultimate Tic Tac Toe is an extension of Tic Tac Toe, a two-player strategy game in which players take turns marking cells with their own symbol into a 3X3 board. The objective of the game is to have 3 aligned cells with your mark - vertically, horizontally, or diagonally.
- The game ends when one player successfully gets 3 aligned cells or when the game board is full and no more moves can be made (draw).
- In the Ultimate version, in each cell is nested another Tic Tac Toe board. You need to win the board to mark the cell.
- You may only play in the big cell that corresponds to the last small cell your opponent played. When you are sent to a field that is already decided, you can choose freely.

Understanding how to model the rules in a code is essential.

4 Code General Structure

Cutting the work down: To code this game it is impotent to devise the work into pillars. We had 5 pillars: the game representation, the Interface, the Hand Gestures, the AI (Bot), and the Communication part (so communicate between two RasberryPi). These pillars all need to be interconnected through a general code that puts them together.

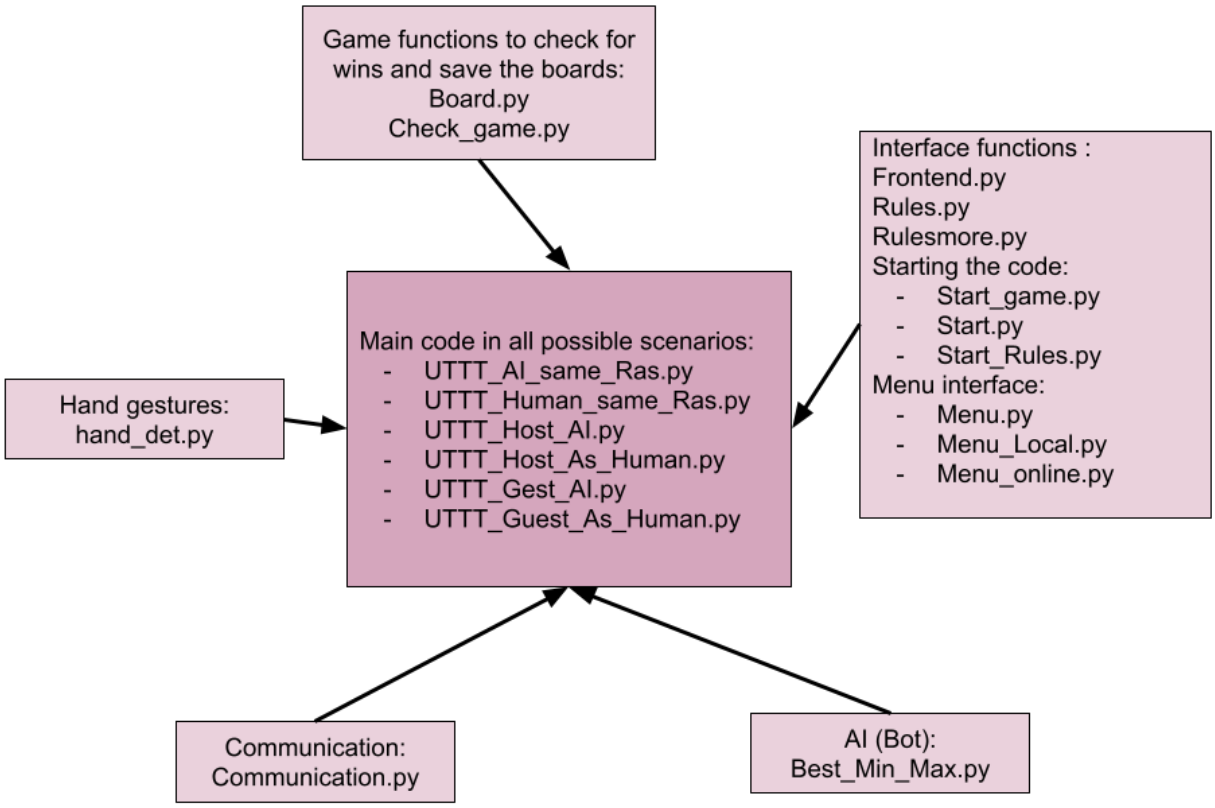


Figure 1: General structure of the code

5 Game representation

Interpretation: We saw the game as two level boards. The first one is a 9x9 matrix that is all the possible moves that can be played and the second one is the board seen as the big Tic-Tac-Toe that needs to be won in order to win the whole game: a 3x3 matrix.

Keeping track of the game: During the game we check at each move if the player that played has won. We also keep track of where the player can play in a list of coordinates of the move in the 9x9 matrix.

6 Interface

6.1 Introduction

So the player does not have to play through commands in the terminal and with a basic idea of just playing an Ultimate-Tic-Tac-Toe, we have decided to create a world and an interface that dives the player in a unforgettable world.

The Context: The game we created is the ULTIMAZE world! You are a royal governor that wants to conquer the ULTIMAZE world against another royal.

Figure 2: Image of the introduction of the game that enrolls out when you start the game

The World of Royals: The game has a pink theme around roses. In a lost world where monarchies are still popular, two matriarchy are in battle. The red rose and the pink rose countries. The era is the Middle Ages. The game is seen as a way to conquer the most territory. That is why it is represented from above as if you are sending troupes on a map.

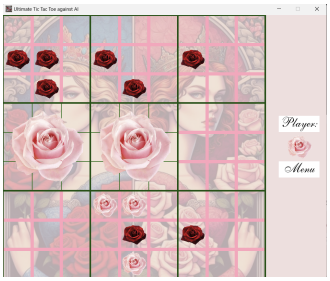


Figure 3: The interface of where to play, representation of a battle field

The Characters: The Royals are two players: the red rose royal and the pink rose royal. They both play against each other and they are looking at the game from above. As the two countries are matriarchy the royals are women.

Figure 4: The royals of the game

6.2 Rules

Since we are in a different world the way to explain the rules must be shown and the user must be able to consult them at any moments. The user must feel in the context of the world to amplify the user experience. On the Menu the player can click on a button and will be sent to the rules interface.

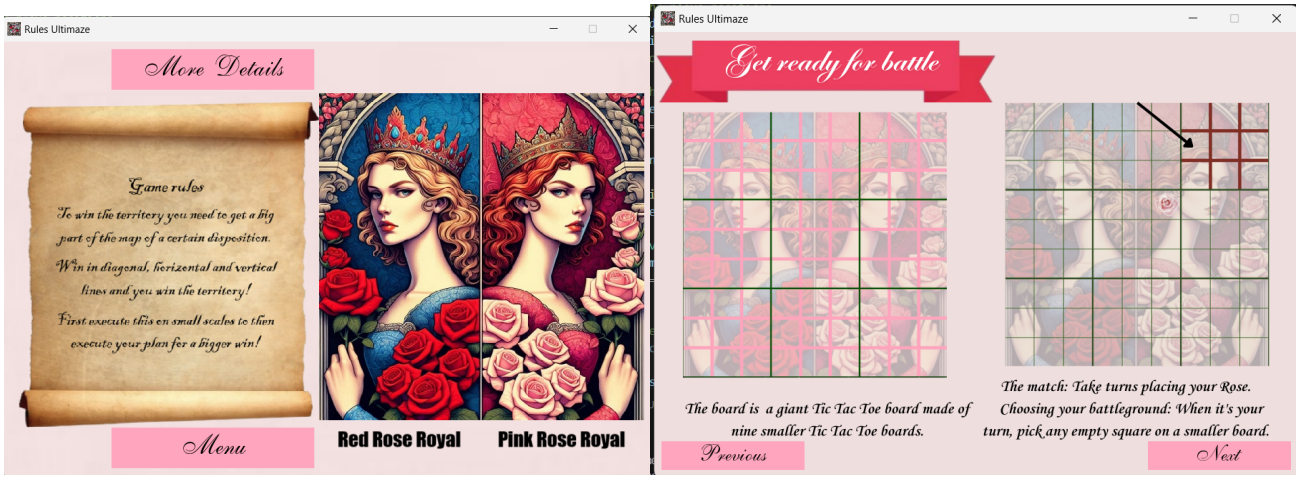


Figure 5: First and second interface of the rules

Figure 6: Third and fourth interface of the rules

Figure 7: Fifth interface of the rules

6.3 Modes of play

As seen in the challenges there are many configurations and the interface should be able to make the user choose their game mode easily. We were inspired by menu pages of current games like Minecraft.

Menu: The user will choose in the Menu page what to play. To do so, the user will choose to play online, as a guest or as a host (to understand what is a guest and a host please read the communication part), or locally.

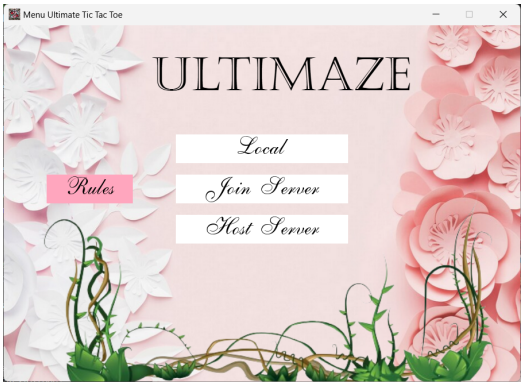
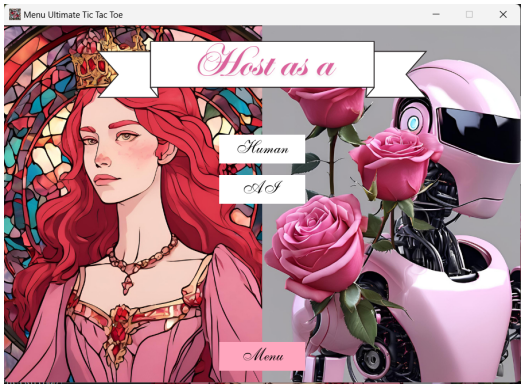


Figure 8: Menu page to select the mode (local or online)

Online: In both cases, as a Guest or a Host the user must choose if we will play online as a AI or as a Human.



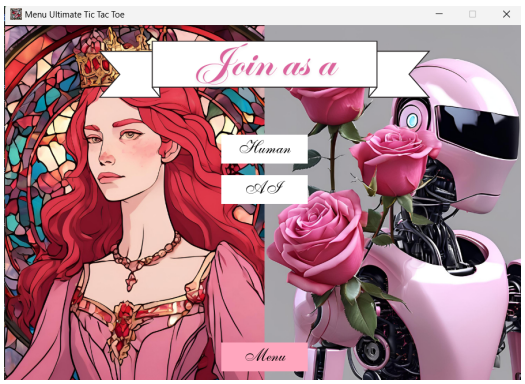


Figure 9: Join or Host as an AI or as a Human

Local: Once the player has selected to play locally he can choose to play against an AI or a Human.

Figure 10: Select if the user will play against an AI or a Human

6.4 Structure Code

General idea: After imagining our world and what we wanted to implement, we needed to find the right software to design the images and the python library to show it. The code is made so that when we will use the interface in the game it will be easy to generate any instance of the game.

Library and used software: To code this interface, we used Pygame. This enables us to change the colors, keep track of what the player choose to play (what pixels where clicked) and create sequences of images that look like things are moving or that something happened when the users mouse is above a button. To generate the images of the Royals and the images of the AI and the human player (see in the modes of play the images of the AI and the human player), drawings were made using a generative AI like bard or the AI of Canva. Canva was used to draw the different backgrounds.

Pygame: Through pygame we can create buttons that enables us to go interact with the user, display information, and get the events that the user created.

The possible events: A player may want to quit the game by clicking on the x button on the upper right of the game. The Interface program must quit the game if this happens. A player might have its mouse over a button. The interface must display a color over the button to amplify it for the user's eyes. The user might click on a button. The interface must be able to display another interface if this happens.

Challenge of coding the buttons: To go back to an interface that was already seen we could not just import the code, otherwise, when the code was executed, it would go into an infinite loop of importation. That is why when the right conditions were met we had to execute certain files that call the function that shows the already seen interface. Here is the code to execute a file without needing to import it at the beginning:

```
with open('Start.py')as new:
    exec(new.read())
```

In the Start.py doc there is only where we want to redirect to:

```
from Main_code.Menu import main
# This file is for to go back to the menu page
main()
```

Since the only buttons, that call an already called interface, call the menu interface or the rules (go back to old rules), the file Start and Start_Rules.py are the only ones that are coded this way.

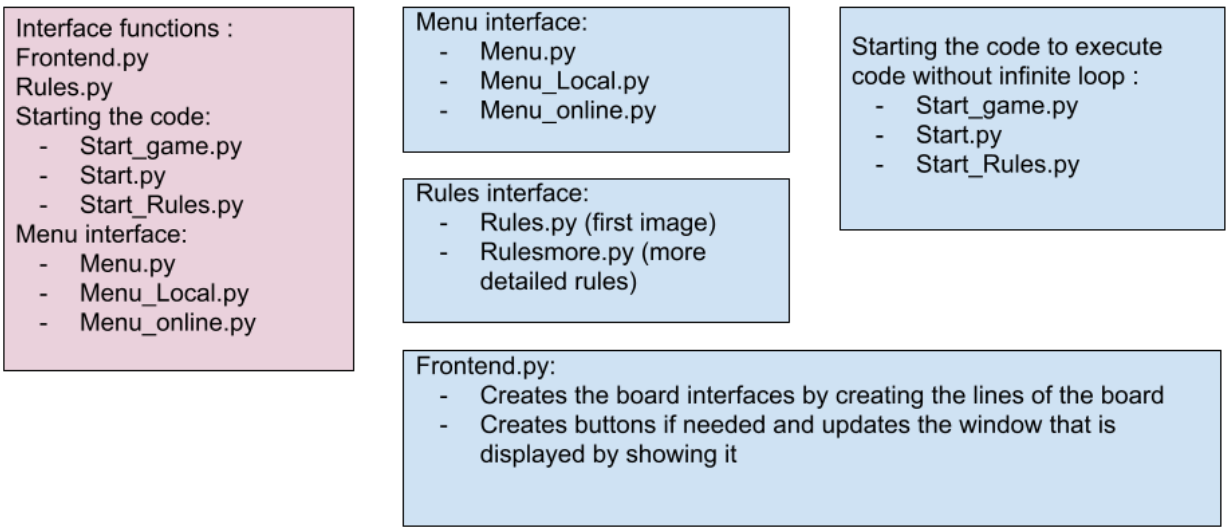


Figure 11: Interface code structure

7 Motion Detection

Challenge and Objective: The goal is, with a camera, to enable the user to have a fun new way of playing the game. The user must be able to play with only their hands in front of the camera. There are a few ways to use the camera. You could detect an object and follow it, you could react to the color of an object, you could follow a hand, or react to the positions of a hand. Using an object will force the user to have to buy it thus increasing the price of the game. An object can also be lost or damaged. For those reasons we decided to work with the hand. The first idea was to use the hand as a Mouse.

Libraries: The libraries that we used were openCv, mediapipe, and pyautogui.

OpenCV: Imported as cv2, OpenCV (Open Source Computer Vision Library) is an open-source library used for computer vision and image processing tasks, providing tools for real-time image and video analysis. We used it for video capturing.

Mediapipe: MediaPipe is an open-source framework developed by Google for building multimodal, cross-platform machine learning pipelines, particularly for real-time perception tasks like face detection, hand tracking, and pose estimation. We used MediaPipe Hand landmarks to detect the hand.

Pyautogui: PyAutoGUI is a Python library used for automating mouse and keyboard actions, enabling control of GUI applications programmatically for tasks such as clicking, typing, and moving the cursor. We used it to change the position of the mouse.

7.1 Hand Landmarker detection

To both, follow the hand as a mouse and to detect the gesticulations of the hands, we needed to have hand landmarkers. Mediapipe Hand Landmarker takes a frame given by OpenCV and analyses it. The result object contains hand landmarks in image coordinates (in pixels), hand landmarks in world coordinates and handedness(left/right hand) of the detected hands. This is how it is structured:

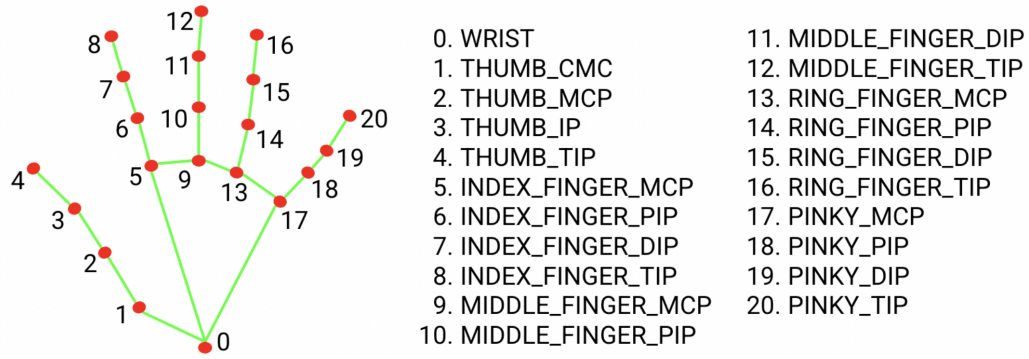


Figure 12: Landmarks references, positions

With the positions of each landmarks, we can display around each of them a circle to check that a point was detected.

As we are detecting frames of a video in time we need to read the frames at certain intervals that is why we define a frame per second parameter. This parameter was adapted to the Raspberry Pi as the Raspberry Pi is slower than a computer. We put the FPS to 60.

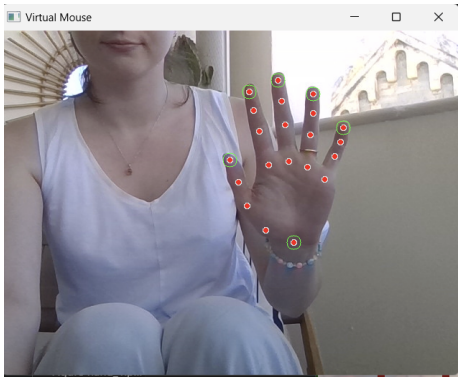


Figure 13: Hand detection through the camera

Basic code: Here is the basic code to get to the point of showing the landmarks using the model:

```
# ¨Parameter of the camera
cap = cv2.VideoCapture(0)
hand_detector = mp.solutions.hands.Hands()
drawing_utils = mp.solutions.drawing_utils
screen_width, screen_height = pyautogui.size()
frame_width, frame_height = (640,480)
hand_y = 0
#to select the frame rate
clock = pygame.time.Clock()
FPS = 60
#We get a new frame at the frame per second rate of 60s
while game_not_over:
    # Limit frame rate for smooth gameplay
    clock.tick(FPS)
    _, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frame=cv2.resize(frame,(frame_width, frame_height))
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    # The line cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
        #is in the code because OpenCV (cv2) reads
        #images in BGR (Blue, Green, Red) format by default,
        #whereas the hand detection model from MediaPipe
        #expects the image to be in RGB format (Red, Green, Blue).
        output = hand_detector.process(rgb_frame)
        hands = output.multi_hand_landmarks
        #This is where we find the landmarks of all the hands we detected
        #If you detect hands
        if hands:
            #Check only one hand and draw it
            hand = hands[0]
            drawing_utils.draw_landmarks(frame, hand) #Drawing the landmarks
            landmarks = hand.landmark

        #Show the camera
        cv2.imshow('Virtual Mouse', frame)
        cv2.waitKey(1)
```

In this code we only display the landmarks, we do not do anything with it yet. Now that we have the location of the hand and of the different fingers we must use this information.

7.2 Mouse Following

7.2.1 Implementation

Design: We would like to use the location of the landmark number 0 to follow as a mouse location and click when landmarks number 16 and 0 are close in distance. When clicking we will use the pyautogui library.

Code structure: After implementing the frames we add a code inside the frame detection that acts according to the location of the landmarks.

```
if hands:
    hand = hands[0]
    drawing_utils.draw_landmarks(frame, hand)
    landmarks = hand.landmark
    for id, landmark in enumerate(landmarks):
        x = int(landmark.x*frame_width)
        y = int(landmark.y*frame_height)
        #Here we add the code needed
        #We take the coordinates of number 16th
        #We draw a green circle around the landmarks
        #we will be using for the user to understand better.
        if id == 16:
            cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 0))
            hand_y = screen_height/frame_height*y
        #Get the coordinates of the 0 landmarks
        #Compare inside the coordinates with the 16th
        if id == 0:
            cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 0))
            x = screen_width/frame_width*x
            y = screen_height/frame_height*y

            #If there is a click
            if abs(hand_y - y) < 200:
                pyautogui.click()
                pyautogui.sleep(1)
            #Always follow the hand
```

```
        elif abs(hand_y - y) < 600:
            pyautogui.moveTo(x, y)
cv2.imshow('Virtual Mouse', frame)
cv2.waitKey(1)
```

7.2.2 Issues

Latency: After testing this code on the computer we realized it was efficient. However, when testing on the Raspberry Pi it was slow and particularly impossible to use even with a slow frame rate.

Detection of distance: When testing it was essential to understand that the distance of the hand to the screen had a huge impact on the distance detected by the hand. We could see that a far away hand could be detected as a click.

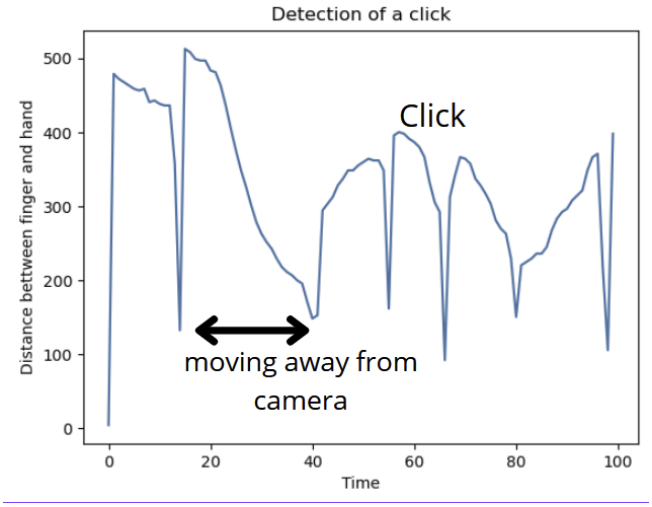


Figure 14: Effect of the distance to the camera on the distance detected between fingers

7.3 Hand Gestures

To find a solution to the issues we detected, we decided to change the way we used the motion detection. Instead of seeing a hand as a mouse we decided to communicate through gestures. The gestures will translate where to go and what actions to take.

Landmarks used: The positions (x,y) of landmarks 0,4,8,12,16,20 are known. They are the points which are the most visible. We then implement a code to find out whether the hand is executing different gestures by calculating the distance between the different landmarks on the hand. Then we take actions according to the gestures seen.

Purple square: The square where you want to play is preselected (highlighted in purple). At the start of the game, the preselected square is the center square. It was important to add a purple square to know where the user was going to play for the user to know what he is doing.

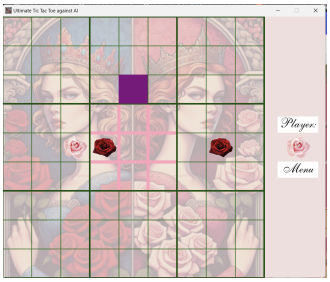


Figure 15: Interface to follow the position you will be playing

Move by one the purple square (the selection): To change the preselected square, execute the following commands: To move to the left square, the thumb must touch the index finger(8). To move to the right-hand square, the thumb must touch the middle finger(12). To move to the top square, the thumb must touch the ring finger(16). To move to the bottom square, the thumb must touch the little finger(20).

Teleport the purple square (the selection): For greater playability, we've added a command that allows you to move closer, i.e. to the first playable square given by the box function. You need to move your middle finger (12), little finger (20) and ring finger (16) towards your wrist (0), keeping your index finger (4) raised.

Select the position: Then, to validate the square where you want to play (the square highlighted in purple), you need to move the little finger(20), ring finger(16), middle finger(12) and index finger(8) closer to the wrist(0).

Go back to the Menu: Bringing the middle finger(12) and ring finger(16) closer to the wrist(0) while keeping the index finger(4) and little finger(20) raised(photo) allows you to leave the game.

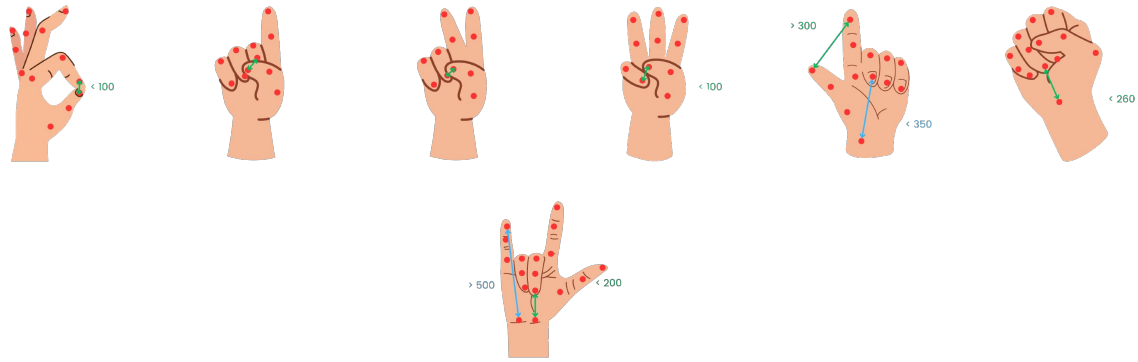


Figure 16: Hand gestures: moving left, right, up, down, teleport, select, menu

Code structure: The code has the same structure and layout as the one for the mouse detection but we changed the reaction of the code to the different distances between the new landmarks we implemented.

Final production and result: This code was tested on the Raspberry Pi. The camera is still slow to detect but it works well with this implementation and a player would be able to quickly change hand gesture with having a smooth game. On the computer this implementation has the issue of having a too quick detection. This means that if your fingers are touching for more than a second, the game will react as if you made the gesture many times. As this game is to be implemented on a Raspberry Pi we decided to choose the hand gesture detection method that is better for such a use.

8 AI

Challenge and Objective: The goal is to create an AI that the player can play against. The challenge is that it should be able to win against other AI, from other teams.

8.1 MiniMax

The MiniMax algorithm, often used in decision-making and game theory, helps in determining the optimal move for a player, assuming the opponent also plays optimally. It is commonly used in two-player games such as tic-tac-toe, chess, and checkers.

8.1.1 Algorithm

The basic idea of the MiniMax algorithm is to explore all possible moves in a game to determine the best move for the player. Here's a step-by-step breakdown of the algorithm:

1. Generate the Game Tree: Construct a tree where each node represents a game state, and each edge represents a move from one state to another. The root node is the current game state, and the leaf nodes represent possible game outcomes.
2. Evaluate the Terminal States: Assign a numerical value to each terminal state (leaf node). This value reflects the desirability of the game outcome (e.g., a win might be +1, a loss -1, and a draw 0).
3. Backpropagate Values: Starting from the leaf nodes, propagate these values back up the tree to the root node. For a maximizing player, choose the maximum value among the child nodes. For a minimizing player, choose the minimum value among the child nodes.
4. Choose the Optimal Move: At the root node, select the move leading to the child node with the optimal value after backpropagation.

8.1.2 Detailed Example

Let's consider a simplified game scenario with two players: Max (trying to maximize the score) and Min (trying to minimize the score).

Generate the Game Tree:

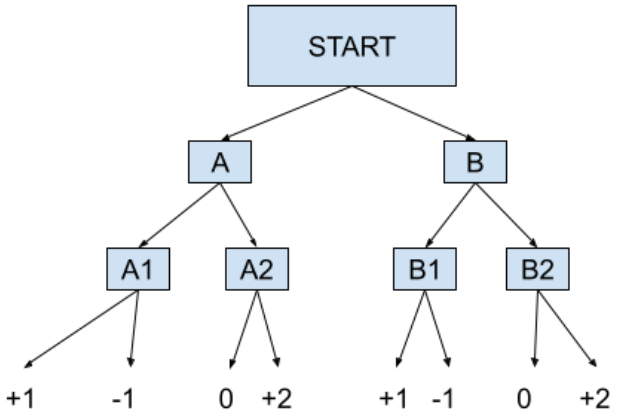


Figure 17: MiniMax example tree

1. Evaluate the Terminal States: Terminal nodes are evaluated as follows: +1, -1, 0, +2, +1, -1, 0, +2.
2. Backpropagate Values:
   (a) At Level A and B:
   - A1: Max chooses between +1 and -1 => Max(A1) = +1
   - A2: Max chooses between 0 and +2 => Max(A2) = +2
   - B1: Min chooses between +1 and -1 => Min(B1) = -1
   - B2: Min chooses between 0 and +2 => Min(B2) = 0

(b) At the Root (Start): Max chooses between the values from A and B:

- A: Min(A1, A2) => Min(+1, +2) = +1
- B: Min(B1, B2) => Min(-1, 0) = -1
- Start: Max(Start) => Max(+1, -1) = +1

8.1.3 Heuristic

General idea: Heuristics are used to estimate the value of non-terminal states when it is impractical to search to the terminal states. A common heuristic is an evaluation function, f(s), which estimates the desirability of a state s. For example, in chess, the evaluation function might consider material balance and positional factors.

Simple Heuristic: We started implementing a simple Heuristic. We used the heuristic most used in a simple Tic-Tac-Toe. It is a matrix where each element represents the "mass" or value of a game piece at that position.

Figure 18: Heuristic choosen of simple Tic-Tac-Toe

After we define a heuristic evaluation function, f(s), that uses the matrix of masses to evaluate the desirability of a game state s. The function sums up the masses of the player's pieces and subtracts the sum of the opponent's pieces. Since this Matrix is a 3x3, we use it for every boards.

Final Heuristic: To take into account both the small boards and the big board in a new way we decided to change the matrix by making is directly a 9x9.

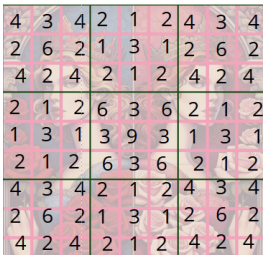


Figure 19: Final heuristic chosen

We must not forget that in the evaluation function, if there is a winner the function must go to -infinity or +infinity. This means that there is not only the mass matrix in the mass function but also a winner check that changes the score according to the wins.

8.1.4 Alpha-Beta Pruning

To make the code go faster we select only blanches that are useful. Alpha-Beta Pruning is a technique used to reduce the number of nodes evaluated by the MiniMax algorithm in its search tree. It is particularly useful in games like chess or checkers, where the number of possible moves can be extremely large. The pruning process eliminates branches in the tree that do not need to be explored because they cannot influence the final decision.

- Alpha: The best value that the maximizer currently can guarantee at any level of the tree. It starts with a very low value (negative infinity) and increases during the maximizer's turn.
- Beta: The best value that the minimizer currently can guarantee at any level of the tree. It starts with a very high value (positive infinity) and decreases during the minimizer's turn.

The main idea is to keep track of these two values while traversing the tree and prune branches that cannot possibly affect the final decision.

- Initialization: Start with

$$\alpha = -\infty$$

$$\beta = +\infty$$

- Traverse the Tree: Explore the game tree in a depth-first manner, updating alpha and beta values at each node.
- Prune Branches:
  - If the current node is a maximizing node and its value is greater than or equal to beta, prune the remaining branches because the minimizer will not allow this value to be chosen.
  - If the current node is a minimizing node and its value is less than or equal to alpha, prune the remaining branches because the maximizer will not allow this value to be chosen.

The final decision at the root node (Max) is based on the evaluated values. After pruning, the algorithm efficiently determines the optimal move without evaluating every possible state.

8.2 Other Algorithm

When developing an AI or bot for decision-making in games or other scenarios, several algorithms can be considered. The MiniMax algorithm, along with its optimization via Alpha-Beta pruning, is a commonly used and effective approach. However, there are other potential algorithms such as Monte Carlo methods and machine learning techniques. This explanation will cover why MiniMax was chosen over these alternatives and why the others might be less effective in certain contexts.

Monte Carlo: Monte Carlo methods involve running many random simulations of the game to determine the most promising move based on statistical analysis. It requires many simulations to get reliable results, which can be computationally expensive.

Machine Learning : ML techniques, such as neural networks, can be used to learn optimal strategies from data. Training and running ML models can be computationally intensive. But decisions made by ML models can be difficult to interpret and explain.

Final decision: Given the context and goals of our AI or bot development, the MiniMax algorithm was chosen for the following reasons:

- Deterministic and Predictable: The MiniMax algorithm provides consistent and deterministic results, which are easier to debug and validate.
- Optimal for Our Game Type: For two-player, zero-sum games with well-defined rules, MiniMax (especially with Alpha-Beta pruning) guarantees optimal play.
- Simplicity and Understandability: MiniMax is simpler to implement and understand compared to the often opaque decision-making process of ML models.
- Resource Efficiency: While Monte Carlo methods can be effective, they require significantly more computational resources. MiniMax with Alpha-Beta pruning is more resource-efficient and suitable for real-time decision-making.

This Bot is quick on the Raspberry Pi except on the first move. That is why we implemented that the first move will always be the position $[4, 4]$.

9 Communication

Objective and Challenge: The goal is for two Raspberry Pi to play together. We thus have to define two protocols to use, one on the Transport layer and one on the Application Layer.

9.1 Transport Layer:

The transport layer is a layer in the OSI (Open Systems Interconnection) model responsible for providing end-to-end communication services for applications. It ensures that data is transferred reliably and accurately between devices over a network. Key functions of the transport layer include error detection and correction, flow control, and ensuring complete data transfer. Examples of transport layer protocols are TCP (Transmission Control Protocol), which provides reliable, ordered, and error-checked delivery of data, and UDP (User Datagram Protocol), which offers a faster but less reliable transmission.

TCP: We decided to choose the TCP connection protocol as it is safer then UDP. This enables us to communicate between two Raspberry Pi on the same Network. However, using this protocol means that there has to be a Server and a Client: a Host and a Guest. The server has to open a socket for the client to sent information too. The client initiates the connection, and the server responds, facilitating structured communication, resource management, security, and reliability in networked applications. This division of roles is fundamental to the effective operation of networked systems and the Internet. However the system of a Host and a Guest (Server and Client) is not useful for our game. We had to take into account all the different scenarios this brought up (Human Host vs AI Guest, AI Host vs AI Guest, Human Host vs Human Guest, AI Host vs Human Guest). The TCP connection creates a connection between the server and the client.

Server/Host: The Host is a server: it starts by opening a socket and listening at a given port. Once the Host gets a connection from a client the TCP connection is established and an exchange of information can happen. The TCP connection can be closed by both the Host and the Guest.

Client/Guest: The Guest needs the IP address and the port number of the Host. This means both players must communicate this information before playing. The Guest will then connect to the Host through TCP connection.

9.2 Application Layer

We need to implement a protocol for the Application Layer to know what messages we will send through the TCP connection and if there will be a protocol for error checking of the game. The goal of the Communication pillar is to make sure the board game, the move played by each player and the error are taken cared of so that both Raspberry Pi has the same information.

9.2.1 Protocol

This protocol aims to enable communication between two devices playing the Ultimate Tic-Tac-Toe. This protocol sends messages that start with UTTT/1.0 [Command] [Attribute1] [Attribute2]\n

The important part is what to put in command, attribute1 and attribute2. This has to send the errors, the move to play and the state of the whole game. There are three phases: the connection phase, the playing phase and the win phase.

9.2.2 Technical Description

Phases and Time out:

- PLAY Phase: A 10-second timeout is set during the PLAY phase. If the delay between sending and receiving the PLAY and ACK messages exceeds 10 seconds, the connection is terminated, and a FATAL_ERROR is sent.
- CONNECTION and WIN Phases: Similarly, a 10 second timeout is applied during the CONNECTION and WIN phases.

Representation of the board: We want to be able to send the whole board to check that everyone has the right information but also know how to give the coordinate of the played move. The board is represented in a different way then in the game see before. The board is represented by a first number for the general game and a number for the small game:

0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8

Table 1: Big board and small board representation

9.2.3 Initial connection

After starting the Host and getting the IP address and port number of the Host, the Guest connects with the TCP connection to the server. To make sure messages can be sent through this connection the client sends UTTT/1.0 CONNECTION [Pseudo]\n. CONNECTION is the command. [Pseudo] is the attribute1 that is the name of the game (for us it is ULTIMAZE). The Host has to reply with the same message but with their Pseudo if we are playing against another Ultimate Tic Tac Toe game with a different pseudo.

9.2.4 Playing phase:

The connection is done and the game must start. Guest is always the one that starts the first move. The request format (the message) to communicate the played position is UTTT/1.0 PLAY [Position] [State of play]\n. PLAY is the command. Position is the attribute1 that is the "number big board number small boars" without any space.

State of play is the game whole board put into a long string and hashed. The game state is the state BEFORE the moved is played. The string representation is:

- "1": First player (GUEST)
- "0": Second player (HOST)
- ".": Empty square
- "/": Line parser
- "-": Square parser

Here is an example: "010101101-.....-...../0.....1-....1....-...../.....-....1....-1...0...."

Later to put the communication pillar code into the code of the game it is mandatory to make functions to go from the communication protocol board positions to the game interpretation of the board and vice versa.

Time-out starts Once the player has sent the PLAY command, the player starts a time out waiting for the others NEW_STATE command with the new state of the game with the position sent. (UTTT/1.0 NEW_STATE [New state of play]\n).

Error position: If there is an error with the position played the player receives from the other player the command UTTT/1.0 405 BAD_REQUEST\n. The game will then end sending the player to the interface of No Winner.

Comparing the boards: The player compares that hashing of the state of the game with the one of his game after playing his/her/its move. If everything is correct the player replies with Acknowledgement: UTTT/1.0 ACK \n. And ends his/her/its time out.

Error boards: However if the state does not match the player sends back : UTTT/1.0 404 STATE_PLAY [Position] [previous state of play]\n. If the player has to send more then two times this error in a row or if the time out ends before the ASK message, the Error: UTTT/1.0 406 FATAL_ERROR\n is sent. This ends the connection and the game.

Hashing information: The communication decided it would be easy to compare the state of play and to communicate the information in a secure way if we hashed the state of play. The state of play is hashed using SHA-3 224 with Python's hashlib module to ensure consistency between devices.

9.2.5 End phase:

If any player leaves the game during the game a message with the command END is received or sent: UTTT/1.0 END\n. The player is then exited.

9.2.6 Win phase

If the player that receives the last played position detects a WIN after all the exchange messages on the position a request is sent. The command is WIN and the attribute1 is HOST if the Host wins, GUEST if the guest wins, and Nothing if it is a tie. Every time a win is sent or received a time out is started. If that win is true a END command is sent and the game ends, the connection is closed.

If there is an error, the 406 FATAL_ERROR command is sent. This ends the game and no one wins as there was an error.

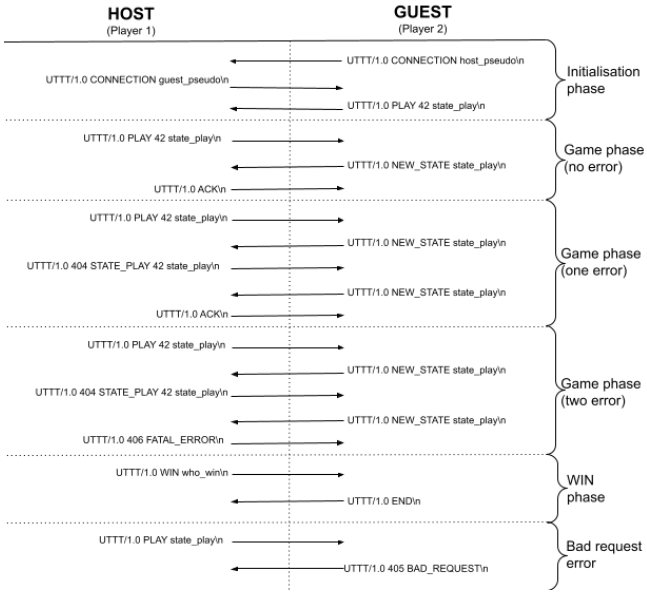


Figure 20: Diagram of Communication

10 Algorithm

Now that we have made all the pillars, it is important to make them work together. To do so we have a simple code that is the same in every scenarios. The element that changes is how to get the information of what move was played.

Initialize game settings and assets

- Define host IP and port
- Set screen dimensions and colors
- Load image assets for game pieces
- Create an instance of the game board

Define main game loop function

- Create a connection with the server
- Initialize game window and clock
- Set initial game variables
- Main game loop
  - Update game window
  - Handle events and user inputs

- Handle turn-based gameplay
  - Host plays (turn == -1)
    - Receive move data from the server
    - Update game board with the received move
    - Check for win conditions and send updates
    - Switch turns
  - Client plays (turn == 1)
    - Determine best move using the AI algorithm
    - Update game board with the best move
    - Send move data to the server
    - Check for win conditions and send updates
    - Switch turns
- Check for game state updates and win conditions
  - Check for small board wins
  - Check for main board wins
  - Handle end-game scenarios
- Send and receive data over the network

11 Potential

After realizing all this it is important to note that there can be some improvements.

Interface Enhancements To upgrade the user interface, better graphics and smooth animations will be integrated to make the game more visually appealing. Modern UI frameworks such as Unity or Unreal Engine will be employed to create a more immersive experience. Additionally, user profiles will be implemented, allowing players to log in, track their statistics, and view their game history.

Communication Enhancements The game will be enhanced with the ability to use Wi-Fi and the internet, enabling players to play online with others globally. An online ranking system featuring an Elo rating will be introduced to make the game more competitive.

Motion Detection Improvements Motion detection will be expanded to include a wider range of gestures, allowing for more complex interactions and a more intuitive gameplay experience. The system will also be enhanced to adapt to different lighting conditions and backgrounds, making it more robust in varied environments.

Artificial Intelligence Improvements More sophisticated AI techniques like Monte Carlo Tree Search (MCTS) or Reinforcement Learning will be implemented to create a more challenging and adaptive AI opponent. Different levels of AI will be developed so that players of varying skill levels can enjoy the game. Additionally, features explaining the AI's decisions and moves to players will be implemented, enhancing their understanding and improving their learning experience.

Additional Features Social features such as chat, friend lists, and leaderboards will be integrated to enhance the community aspect of the game. Anti-cheating mechanisms will be implemented to detect and prevent unfair practices, ensuring a fair gaming environment.

12 Conclusion

Our ULTIMAZE game is a complete game. Through the integration of advanced interfaces, efficient communication protocols, motion detection capabilities, and dynamic MiniMax bot algorithms, we have developed a versatile and feature-rich game that pushes the boundaries of traditional Tic-Tac-Toe gameplay.

For future projects, it will be important to work on the group dynamism and communication. Even if we were able to finish the project in due time.

13 Bibliography

References

[1] *MediaPipe Solutions provides a suite of libraries and tools for you to quickly apply artificial intelligence (AI) and machine learning (ML) techniques in your applications.*

[2] THE LANCET Global Health, V. Shashkina, *Git with code for object tracking*, December 1, 2020

[3] OpenCV-Python Documentation, eastWillow, *OpenCV Documentation*, 2016

[4] Pygame *Pygame Documentation*, 2016