S6 Semester Project Report

Group 4A :

SARR Serigne Khadim, BALESTRA Alexis, WANG Jiacheng,
KIRI Khalil, MEKASS Aya

Interface

Early goals

The main goal of this interface was to provide a user-friendly way to play the Ultimate Tic-Tac-Toe game using python and a Raspberry PI 4. That said, several considerations were to take into account if we wanted a pleasant result, both on user friendliness and visuals:

- First, we needed smooth animations and sprite handling. However, taking into account the fact that there should be object tracking gameplay, AI moves processing, the communication protocol and the main program displaying everything at the same time, we thought it was relevant to “dispatch” all the three
- The object tracking gameplay has to be intuitive and accessible to the user in order to not frustrate him while he plays, so he should have the choice between playing with a mouse if he doesn’t enjoy the hand tracking utilities for instance
- The player should have a variety of choices in the parameters, allowing him to change the AI’s algorithm depth, control the game music and other relevant parameters

And so we embarked on this thrilling journey !

First promising steps

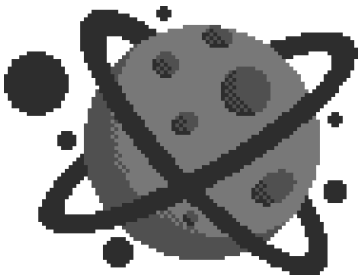
To answer those concerns, we first thought that multithreading was the best way to achieve what we wanted, but well, we knew that python doesn’t have real multithreading because of the GIL (Global Interpreter Lock) that only allows a single thread to execute bytecode at once… Thus, the additional “threads” are in fact a single thread switching very fast to the different bytecodes, which increases performance on I/O bound tasks, but that is not efficient at all on more demanding tasks (such as image processing).

Above you can see our buttons design, from where everything started

Fortunately, after some quite intense research on the subject we decided to use python in order to call subprocesses - that runs on real threads this time - which are “monitored” by python threads (or the main program itself depending on cases), that is to say the python threads pipes their standard output and error streams and then notify (using data locks and shared data structures) to the main program what they actually get from them. That approach allows the main interface program to continue its tasks (drawing and updating the sprites on the screen, getting user input…) all that without being significantly slowed down by the other game assets. See `subprocesses_utils.py` in the source code.

Problems encountered and solution

At that stage, the interface was a big machinery of python threads, subprocesses calls and shared data handling, but as we were progressing we faced one last big problem we hadn’t noticed: we were naive enough to think that the standard output of a subprocess was so kind he would allow us to access it before the subprocess’ end. Nay ! All that big machinery fell apart as the functions to monitor the subprocesses were *all* waiting for them to end before acting.

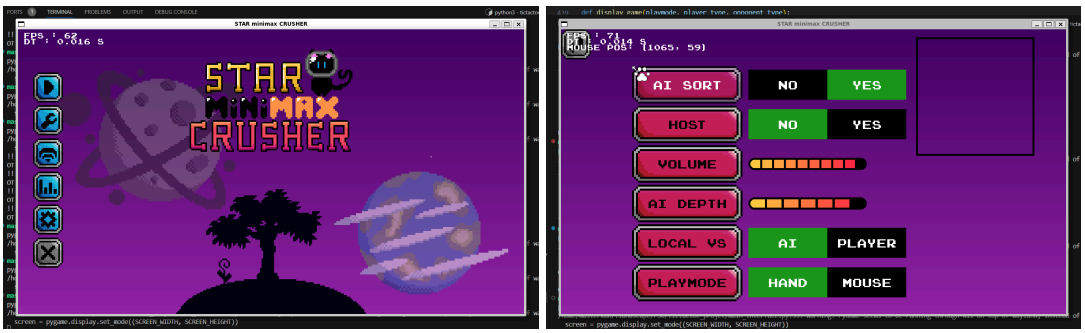


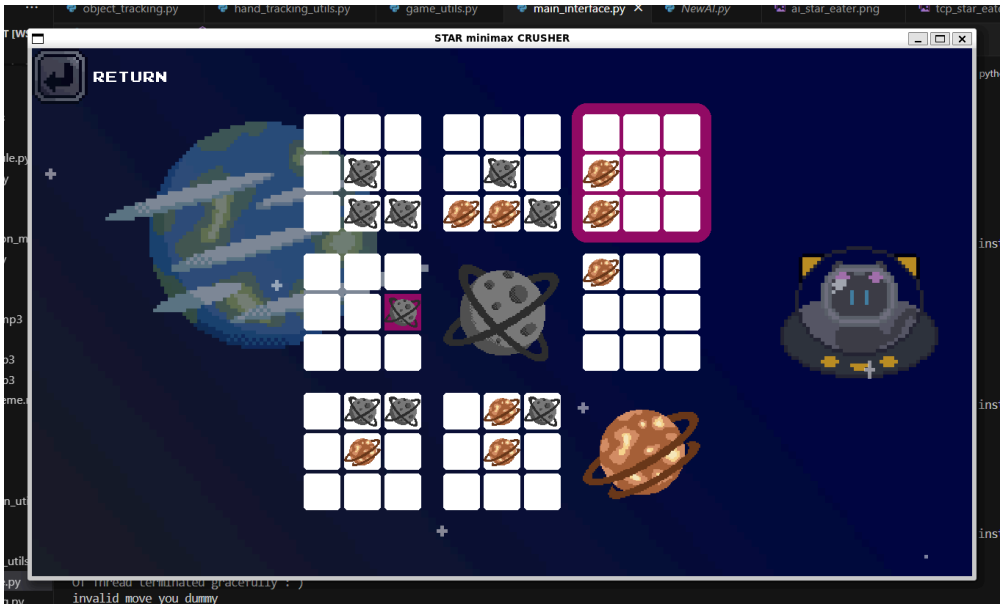
Gravitating around the problem like the pixels around this planet

Discovering that while running tests on the object tracking was harsh, but we came out with an excellent fix: allowing the OS to set the subprocess’ stdout as non blocking through their file descriptors whenever we run one (works on Linux). This allowed us to perform as we planned, and the only trade off was harder error handling when accessing one subprocess’ stdout as it is really unstable.

Game utilities, assets and last word

We made a subsequent effort in designing everything ourselves, starting with the pixel art. Instead of using modules such as pygame_gui to draw buttons and options we designed our own classes to do so. We also have classes for designing game objects such as particles (stars in the background) or animated sprites. However, as the time dedicated to build the interface was shifted at the end, we could not implement all we wanted on time. However, we plan to continue working on that project even after the deadlines, in order to see this game run flawlessly. You can follow our progress on our gitlab if you want !





Some game screenshots

see: `main_interface.py`, `game_utils.py`, `object_tracking.py`

Minimax AI

The Minimax algorithm is a recursive decision-making tool used in two-player games, where it aims to maximize the player's minimum expected outcome while minimizing the opponent's maximum expected gain. It does this by exploring all possible moves, predicting opponent responses, and choosing the optimal strategy based on the assumption that the opponent is playing optimally. We decided to apply this algorithm for our AI.

beware... the AI star eater !

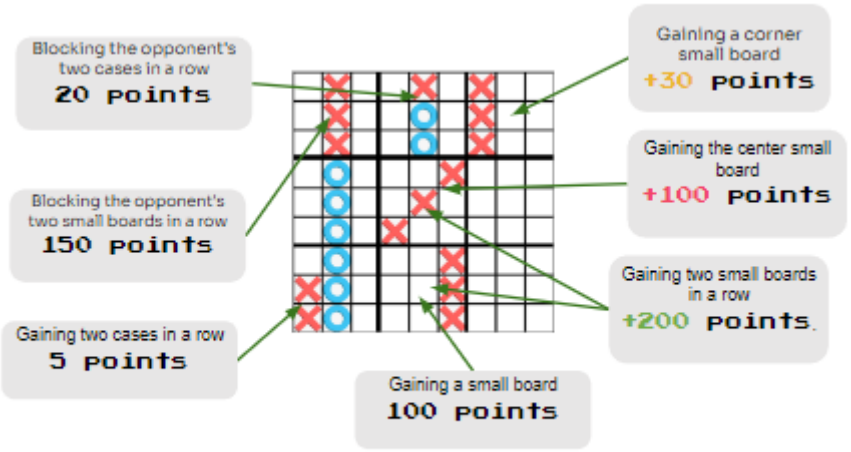
Alpha-Beta pruning

We are using Alpha-Beta pruning. It is an optimization technique for the Minimax algorithm that reduces the number of nodes evaluated in the game tree by eliminating branches that cannot possibly influence the final decision. It does this by maintaining two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of, respectively; branches are pruned when it is determined they cannot improve the outcome.

Move evaluation

We use a ``move_evaluation()`` function to attribute a score to the move for the minimax. Several evaluation function were tested and the following is the one that was the most effective:

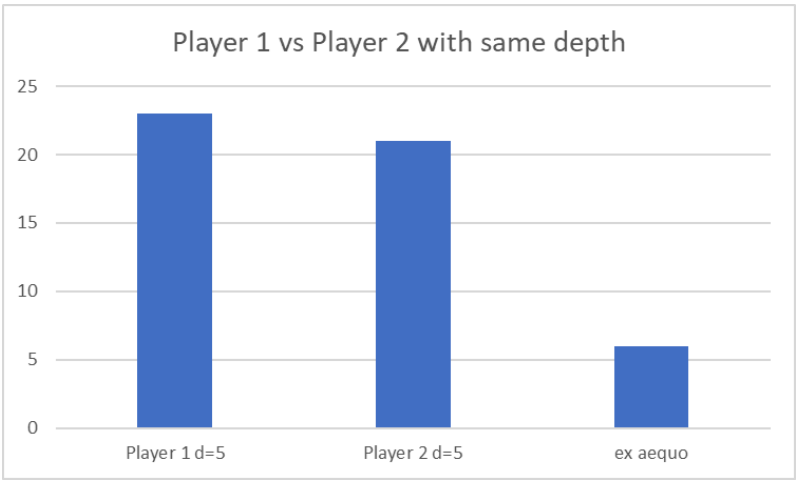
```
Winning the game gives infinite points.
Gaining a small board gives 100 points.
Gaining the center small board gives an additional 100 points.
Gaining a corner small board gives an additional 30 points.
Gaining two small boards in a row gives you 200 points.
Blocking the opponent's two small boards in a row gives you 150
points.
Gaining two cases in a row gives you 5 points.
Blocking the opponent's two cases in a row gives you 20 points.
```



Statistics

In order to test the effectiveness of the algorithm, we wrote a code that makes two versions of the AI play against each other 50 times and then display the wins of each AI as well as the game ending in ex aequo.

We first tested to see if starting gave a significant advantage to the starting player. Player 1 always starts, we see that starting does not give a significant advantage to the AI.



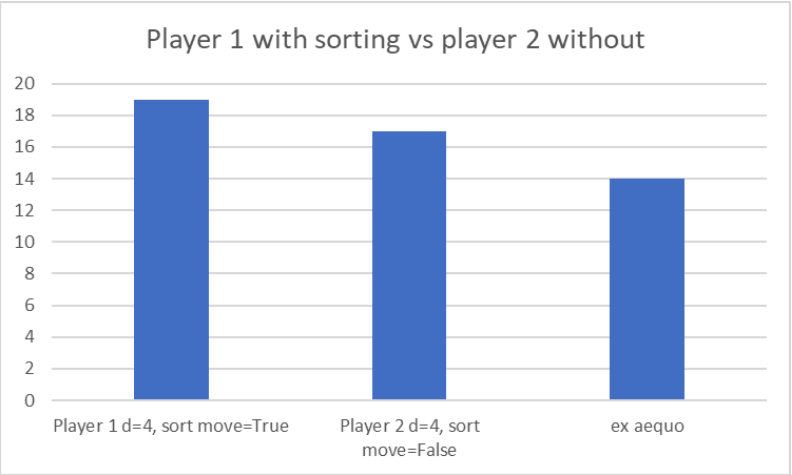
Move ordering and heuristic

In this implementation of an AI for ultimate tic-tac-toe, move ordering is a strategy used to prioritize moves that are likely to be more beneficial for the AI. This is done before the moves are evaluated in the minimax algorithm. The `move_heuristic()` function plays a crucial role in move ordering by assigning scores to potential moves based on how advantageous they are likely to be. For example, winning a small board or securing a center or corner position is given a higher score, reflecting their strategic importance in the game.

The difference between the `evaluate()` function and the `move_heuristic()` function lies in their purpose and application within the AI's decision-making process. The evaluate function assesses the overall game state to determine the AI's standing in the game,

considering factors like board control, potential lines of victory, and the state of each small and large board. On the other hand, the move_heuristic function focuses on the immediate value of individual moves, helping to sort and prioritize them for the minimax algorithm to explore.

This new algorithm doesn't seem to give a big advantage to the player who uses it, this means that we must keep searching for ways to improve the AI.



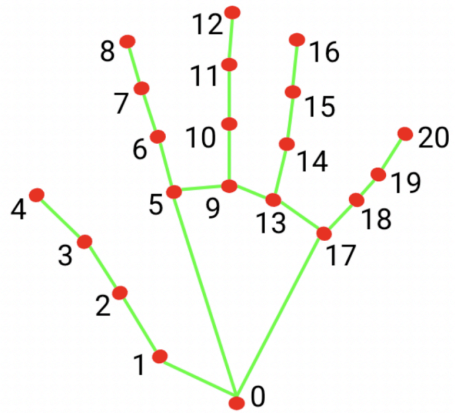
Python code optimisation

The interface played a crucial role in optimizing the AI calculations by allowing the subprocesses to run a specific python interpreter to manage them : pypy. That interpreter can run pure python code that runs for a long time much faster than the classic python3 interpreter built upon Cpython, and with that, a move that would cost 228 seconds with move ordering was reduced to 43~ seconds for instance, see the `test_subprocess_move()` function in `subprocesses_utils`. On average, pypy is 4.8 times faster than CPython 3.7.

Hand tracking

The hand tracking method we use runs with Python and uses the tools MediaPipe and the library OpenCV. To our convenience, we used Mediapipe's hand landmark detection feature to set the foundation of our method, and OpenCV is used to simply capture the frames on which we are doing the hand tracking.

With MediaPipe's AI-trained model, it can easily recognize 21 landmarks of our hands

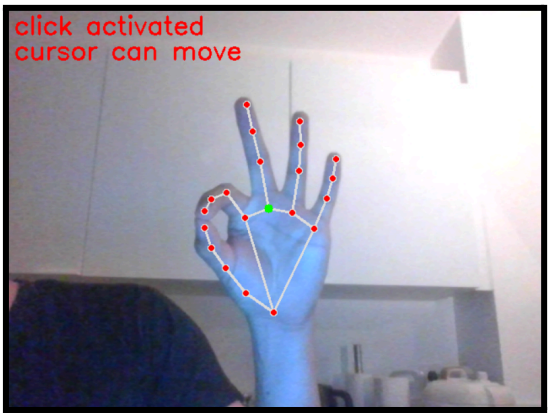


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

The job was simply to put these landmarks in use, as we can recover the x,y coordinates of each landmark on the screen. We decided that only a hand with all the fingers minus the thumb raised could make the cursor "move", whose coordinates are the ones from the landmark 9 of the hand, highlighted below in green. This landmark is the most convenient as it doesn't move when making hand signs and is the "center" of our hand.

The "raised" status would change only if the tips of each finger were above the base of the fingers on the y axis, and then the coordinates of the green landmark would start getting stored and updated continuously.

Moreover, we decided that the "OK" motion would represent the left click of a mouse



This was done by continuously getting the coordinates of the tip of the thumb and the index, if the distance between both was close enough, we would update a boolean to true, signifying that the left click is “pressed”.

Finally, those 3 data, the coordinates x,y of the green landmark and the click boolean would be printed in an infinite loop as long as the program was running, this allows us to get its data with the stdout of this python program.

Issues faced

The most challenging part of the hand tracking is that a Raspberry Pi is not powerful enough for us to run the entire game, plus the hand tracking smoothly, this issue was resolved with two methods of ours.

Firstly, we have optimized the performance of MediaPipe’s hand tracking model by limiting the number of hands that can be recognized in the video stream to 1, then we decided to tweak those parameters to ensure that it’ll still work but to allow us to boost the performance and the speed of the program

`min_hand_detection_confidence`	The minimum confidence score for the hand detection to be considered successful in palm detection model.	`0.0 - 1.0`	`0.5`
`min_hand_presence_confidence`	The minimum confidence score for the hand presence score in the hand landmark detection model. In Video mode and Live stream mode, if the hand presence confidence score from the hand landmark model is below this threshold, Hand Landmarker triggers the palm detection model. Otherwise, a lightweight hand tracking algorithm determines the location of the hand(s) for subsequent landmark detections.	`0.0 - 1.0`	`0.5`
`min_tracking_confidence`	The minimum confidence score for the hand tracking to be considered successful. This is the bounding box IoU threshold between hands in the current frame and the last frame. In Video mode and Stream mode of Hand Landmarker, if the tracking fails, Hand Landmarker triggers hand detection. Otherwise, it skips the hand detection.	`0.0 - 1.0`	`0.5`

For example, we lowered the hand detection confidence so that if someone waves its hand rapidly, which could happen to move the cursor faster, it is still recognized as a hand even if the video stream is getting blurry.

Secondly, we are running this Python file in a subprocess with the subprocess module, in which returned data will be “monitored” by the main program as said in the interface part,. We are using multithreading to run the entire object tracking program concurrently with the main program, allowing us to gain significantly in speed and performance. Still, for that, one thread of the Raspberry CPU is dedicated solely to hand tracking and its translation into the game.

Connection and Communication Protocol

Introduction

This section outlines the activities carried out to establish a reliable connection and communication protocol with TCP between two Raspberry Pi devices for the Ultimate Tic Tac Toe game.

Development Chronology

1. Initial preparation and configuration
    - Activity: Preparation of equipment and initial configuration of Raspberry Pi devices.
    - Details: Set up two Raspberry Pi devices with Raspbian OS, configured network settings, and updated systems.

2. Meetings and protocol design
    - Activity: Held meetings with team members to establish a common protocol.
    - Details: Discussed error handling, packet structure, communication continuity, and security issues. Developed the protocol structure, including methods for connection initiation, game state transmission, and error management.

3. Initial implementation of simple scenarios
    - Activity: Implemented simple game scenarios in the code using if statements.
    - Details: Initially used basic conditional statements to handle scenarios like win detection. This method was functional but not optimal for handling complex game logic.

4. Initial testing on local machine
    - Activity: Tested server-client connection on a Chromebook (Linux integrated).
    - Details: Successfully established the server-client connection using IP addresses and ports. Resolved an issue where the client could not display messages from the server, ensuring smooth communication.

5. Transition to Raspberry Pi
    - Activity: Tested the server-client connection on Raspberry Pi devices.
    - Details: Repeated the connection tests performed on the Chromebook, confirming that the system worked reliably on different hardware.

Problem : Overcoming scenario complexity

During the development phase, handling all possible game scenarios and potential errors posed significant challenges. Initially, the game logic relied on simple conditional statements, which proved inadequate for managing the complexity of the game.

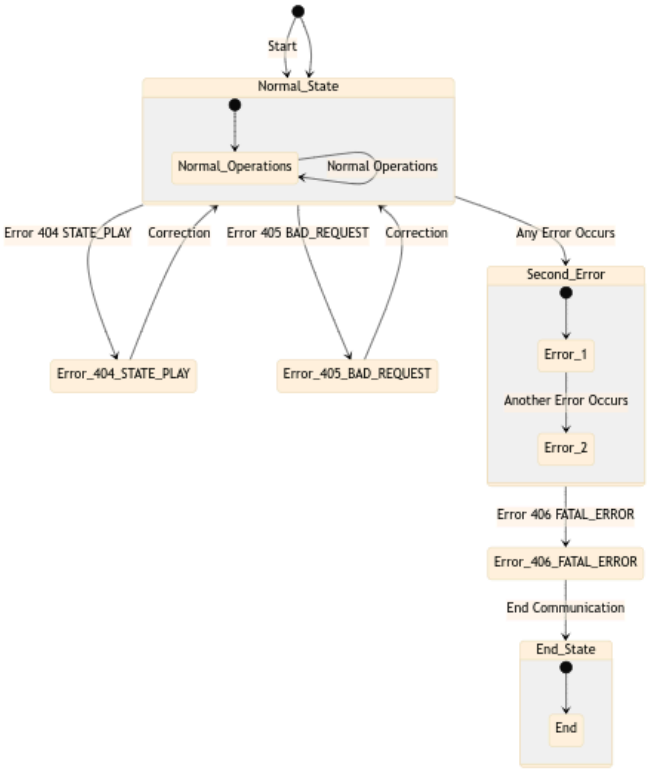
To address this, the focus shifted towards creating small, modular functions to handle specific tasks such as packet creation, sending, hashing, and error handling. These auxiliary functions significantly reduced the complexity by breaking down tasks into manageable pieces. The core game logic now primarily calls these functions to make decisions and manage the game flow, enhancing both efficiency and clarity.

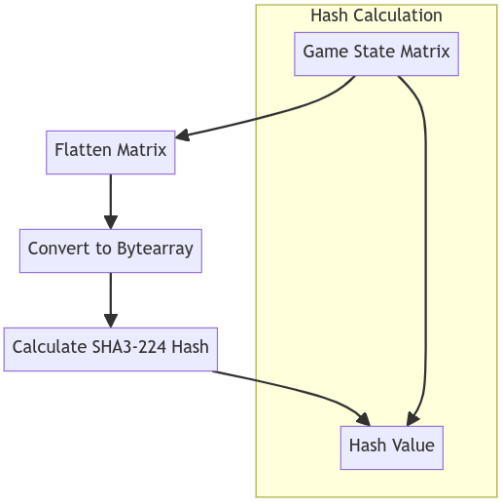
At the end, all those modular functions helped to conceive a global handler that a thread will be monitoring each time a TCP connection starts. See `communication_utils.py` .

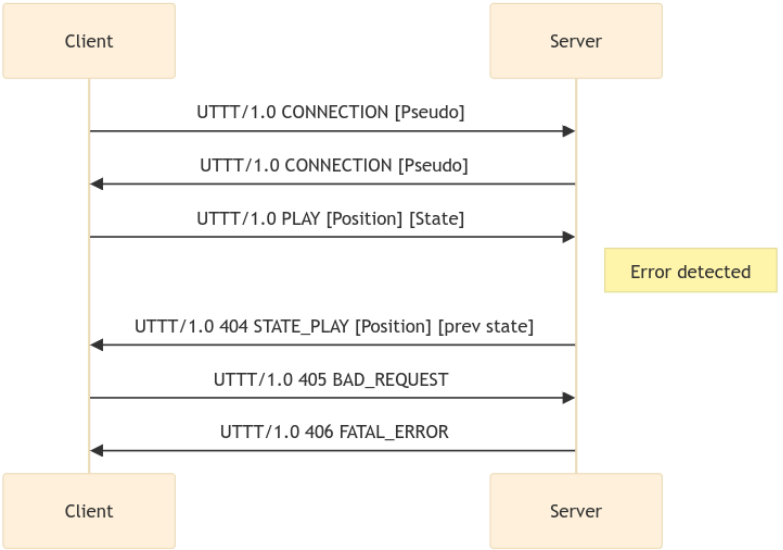
Handling Connection Robustness

During the development, a `keep_alive` function was implemented to ensure the connection remained active. This function ran in parallel and sent a “keep alive” message every 10 seconds to verify the connection's stability. However, it was eventually abandoned because TCP inherently manages connection reliability through its own keep-alive mechanisms. This simplified the protocol and reduced unnecessary traffic.

Diagrams







Quick sum up

The developed communication protocol facilitated correct interaction between the Raspberry Pi devices, allowing the Ultimate Tic Tac Toe game to be played efficiently. Future improvements could include data encryption and a reconnection procedure to enhance robustness.

That's what you are for having read this far !

Conclusion

As a conclusion, our project for Ultimate Tic Tac Toe was implemented using Python, and optimized through the use of threads and subprocesses, providing a good user experience. However, our coding structure could still be improved on the interface part by creating even more reusable structures for the different game states, and our interface is not over yet as some buttons are still useless to this date. For the AI we could experiment with convolutional neural networks or try out other reinforcement learning algorithms such as Monte Carlo Tree Search. Furthermore, we could make it possible for our Raspberry Pis to communicate through the whole internet, and we could have also added more animations to our user interface and made more options for our object tracking.

Apart from that, we really learned a lot concerning python internal functionment, multithreading, subprocesses, synchronization, and concurrent access to shared resources. It was a pleasure to make pixel art and to see our AI evolve over time.

A link to our gitlab :
https://gitlab.eurecom.fr/4a/projets6

Thank you for reading !!