Project S6: Final Report
Super Retro Tic Tac Toe

Group members:

Manon GALTIER
Lilian VARINOT
Sean ROLLY
Youri HALMAERT
Nils DEGHILAGE

Supervisors:

Massimiliano TODISCO,
Pasquale LISENA,
Ludovic APVRILLE

2023-2024

Contents

1 Introduction

1.1 Project's goal

Ultimate tic tac toe The goal of this project was to make an ultimate tic-tac-toe, which is a tic-tac-toe whose squares are tic-tac-toe boards. Whenever the player places a symbol in a small board's square, the opponent can only play in the board whose position corresponds to the square, meaning if the player plays in the top left square, the opponent will have to play in the top left board. If a board is full and a player has to play in it, they can play anywhere.

Additional requirements Our game needs to be able to be played online with other teams, meaning we need a communication protocol. It needs to work using object tracking to select where to play, and it needs to have an AI which can play the game the way a player would.

Pillars The game's creation was divided up into four pillars: interface development, AI creation, object tracking and communication. Each of our group members was assigned a pillar to work on.

1.2 Project's structure

Agile We were asked to work using the agile methodology, more specifically a sprint structure. This means we had meetings to discuss progress and decide what to do for a week or two until the next meeting, then worked on our respective tasks.

2 Work distribution: Pillars

2.1 Pillar 1: Game interface development - Manon GALTIER

We chose an arcade, pixel art theme for our game, which we called Super Retro Tic Tac Toe.

Menus Our game works using a menu system: depending on variables changed by the choices made by the player, a different menu is shown. (see figure 3)

Start menu The start menu presents the rules to the player.

Main menu Through this menu, the player can enter the sprite menu, the connection menu or the game menu.

Sprite menu Here, the player can choose which player plays using which sprite, and whether or not one of the players is an AI.

Connection menu This menu is entered when trying to play online. It allows the player to connect to another player's device, whether as a host or as a guest, and enter the game menu in online mode.

Game menu Here, the player plays the game and can enter the settings menu.

Settings menu This menu allows the user to change settings in game.

Game over menus These menus appear when someone wins or when the game is tied or an error occurs. They can bring the player back to the main menu or let them close the game.

Buttons Our game has clickable buttons made up of hitboxes hidden under images. They react when the player's mouse hovers over them by changing their appearance, and when they're clicked.

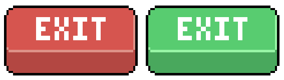

Figure 1: The exit button (left: regular, right: hovered)

Animations Some animations have been added to the game to make it more interesting to look at.

Sprite animations In the sprite select menu, if the player hovers the mouse over a sprite, it plays an animation.

Menu transitions When switching between some menus, an animation plays to make the transitions fun!

Extra animations Sometimes, in some menus, a bird flies over the screen and the cat sprite runs across it.

Sprites The player can select the image which will be displayed when they click on a box on the game board. There are several options for this, but both players have to have different images. A lock replaces the images which are already selected by one of them.

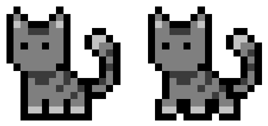

Figure 2: The cat sprite (left: regular, right: running)

Music and sounds We made musics and added sounds to the game.

Sprite selection When selecting a sprite or clicking on the lock, specific sounds play.

Transition A sound plays during the transition between menus.

Music The music in our game was made using a software which replicates how music was made for arcade games. Our music can therefore be played on an arcade machine, adding more accuracy to our arcade theme.

Sound effects Our game has a few sound effects, which are played during the transition between menus and when selecting a sprite.

Options

Sound on/off With this option, the user can choose whether or not they want the game to play sounds and music.

Light and dark modes The light and dark modes give the user greater control over how much the game fits their personal preferences.

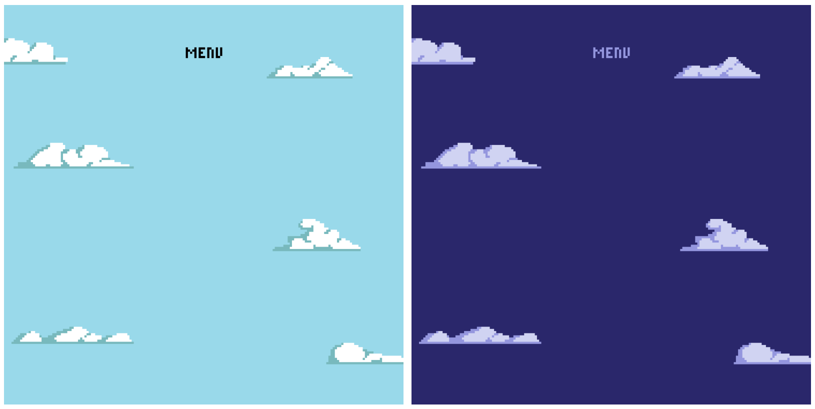

Figure 3: The main menu's background (left: light mode, right: dark mode)

Fullscreen The fullscreen mode has a background which adapts its size depending on the user's monitor.

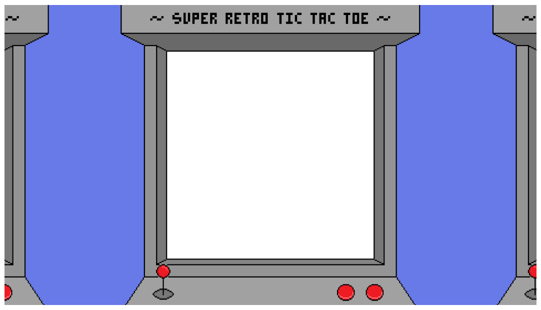

Figure 4: The main fullscreen background (light mode)

Easter eggs Be careful, there will be spoilers in this part!

Hidden sprites Each member of our team has their own personal pixel art. These pixel arts are placed both in the transition animation in the rainbow and on the different game over screens. Our logo is also our tutor's pixel art holding a joystick.

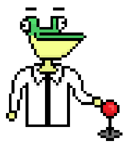

Figure 5: Our logo

Hidden music When clicking on two of our personal pixel arts in the corresponding game over screens, they play songs: the snake plays a small part of Look What You Made Me Do by Taylor Swift and the sushi plays Megalovania by Toby Fox.

Dog The cat sprite has a small chance of being replaced by a dog sprite.

2.2 Pillar 2: Artificial intelligence for the gameplay - Youri HALMAERT & Nils DEGHILAGE

Coding workflow The first option we explored was the Monte Carlo method. After some research, it seemed to be the most efficient. However, too many libraries (difficult to install on the Raspberry Pi) were necessary, which led us to a more classical method: the minimax algorithm.

How does the AI understand the game and try to win it? The AI code works by analyzing a 81 characters string storing numbers assigned to each square on the play grid.

0	1	2	9	10	11	18	19	20
3	4	5	12	13	14	21	22	23
6	7	8	15	16	17	24	25	26
27	28	29	36	37	38	45	46	47
30	31	32	39	40	41	48	49	50
33	34	35	42	43	44	51	52	53
54	55	56	63	64	65	72	73	74
57	58	59	66	67	68	75	76	77
60	61	62	69	70	71	78	79	80

Figure 6: The game grid

The number of each square represents the position of the square in the string. These numbers can be equal to 0 if the corresponding square is free or 1 or 2 if a player has already played in it.

Minimax algorithm The minimax algorithm operates using a decision tree that represents all possible moves currently available and the subsequent moves for each possibility, projecting n moves into the future. Scores are assigned to different potential moves based on their likelihood of leading to a win for each player. The more a move increases the chances of victory for a player, referred to as the maximizing player, the higher its score will be. This is achieved by assigning scores to the terminal nodes in the tree, which reflect how advantageous a move is for a player. Then, the algorithm works its way up the tree, each layer accounting for the fact that each player will choose the optimal move from their perspective. This process estimates the moves likely to be made at each layer. The goal for the maximizing player is to ensure the lowest possible score they might receive is as high as possible, while the minimizing player aims to keep the highest possible score they could concede as low as possible. In essence, each player strives to minimize their potential losses. When the top of the tree is reached, the AI selects the best move based on these calculations, assuming the opponent will employ the same strategy.

We can run the minimax function with a depth of 6 on our computers, which is very strong and can easily win against an untrained human. On the Raspberry Pi, we run the minimax with a depth of 5.

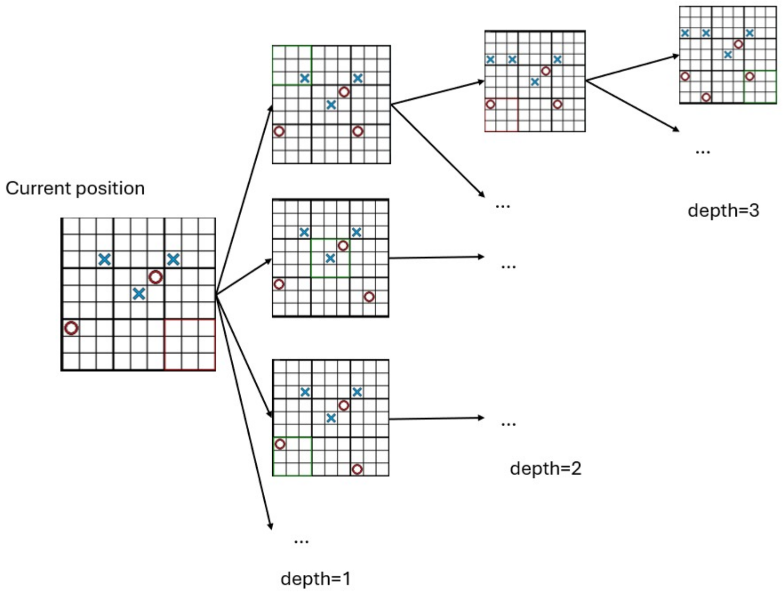

Figure 7: The minimax decision tree

Code structure The AI code is divided in three different parts.

As the game states are not described in the same way by the AI and the interface, there are several functions whose role is to ensure an exact translation from the linear index of the string to the coordinates of the array and vice versa (‘translate’ and ‘translate_other_side’). As well as translation between the different ways of representing the game (‘translate_state’ and ‘translate_state_other_side’). In the end, these state translation functions were not used, as we chose to maintain both game states simultaneously, so we only needed to translate the moves, but they were crucial to check that no mistakes had been made in our other functions.

Then there are all the functions needed to make the minimax algorithm work. The most important of these are:

- ’evaluate’, which evaluates the heuristic value of the entire game using the function ‘evaluate_small_box’ which only evaluates the small boxes.
- ’min_turn’ which tries to reduce the maximum score the opponent can achieve while ’max_turn’ tries to increase the maximum score the player can achieve.
- ‘check_small_box’ that tells the state of a small box (win, lose, tie or still possible to play in it. Then, with ’update_box_won’ a 3x3 grid is updated with check_small_box to know the state of total grid. Using check_small_box on the 3x3 grid, we are able to know if the game is over or not.
- ’successors’ gives all the possible states for a state, a last move and a player thanks to the function ‘possible_moves’ which gives all possible moves for the current position.

Finally, there is the decision-making function ’minimax’ which returns the move and state chosen by the computer depending on depth, player and current state, and the game functions:

- ’who_is_player’, which determines who plays
- ’bot’ which gives the state and the bot move and updates variables
- ’joueur’ which gives the state and the user move and updates variables

2.3 Pillar 3: Object tracking to control the game - Sean ROLLY

For this project, letting the user choose the cell they want to play in using object-tracking was imposed to us. I seized this opportunity to offer the user the best experience possible thanks to handtracking.

Basic explanation of the concept Once the player has chosen the player mode, they can decide to not use their hand instead of their mouse for the selection of the cells they will play in. If they decide to use their hand, their camera will start capturing frames. Then the user will see lines appear on the images the camera gets, imitating a grid of Tic Tac Toe.

The user will simultaneously see the grid of the game and the frames the camera gets so that they can see themselves and know how to move their hand to select the right cell.

If both the big cell and the small cell need to be chosen, then the user simply has to choose twice, starting with the big cell.

In addition to having the freedom to choose whether to use the mouse or the hand, the user also has the ability to choose which sign they want to be the signal for the selection of the cell. This is done after choosing the hand as the way to select the cell. The user chooses how many raised fingers they want to be the signal by clicking on the corresponding number on the screen.

Up to two hands can be detected at the same time, but only one hand will be taken into account when the number of raised fingers is computed. That’s why for a more enjoyable experience during the game, we ask the user to use only one hand to select the play cell (it can be the left or the right hand).

Our code is able to recognize hands and count the number of raised fingers whether it is your left or your right hand, but also independently of skin color, motion, lighting and palm orientation.

Deeper dive into the science The handtracking code is written in Python, allowing productivity and performance, the latter being a crucial point since we are working on a Raspberry Pi and its capacity in terms of memory storage and speed is lower than a usual machine’s.

It uses the following Python libraries:

- mediapipe, a library dedicated to handtracking, enabling us to detect a hand, draw critical points of the hand, get their coordinates, etc.
- cv2, a library specialized in image processing that helped us draw lines on top of the frames and color the play cell for instance.

- Picamera2, a library specific to Raspberry pi use that allows us to use the camera, capture frames and visualize them.
- time in order to get a frame rate to define the number of frames per second.

The code is composed of 3 major parts:

- the import of the needed libraries
- the definition of useful variables and functions like the choose_number function which asks the user to choose a number between 0 and 4 to define the way of selecting the play cell.
- the main part, i.e. the handtracking function that is used all over the heart.py code.

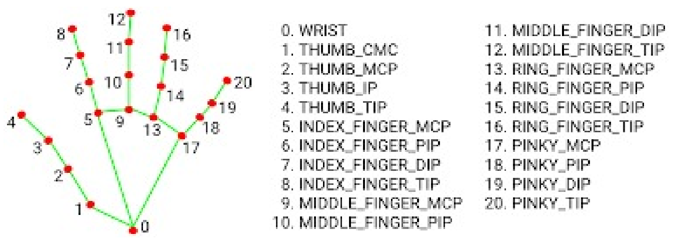

Figure 8: hand landmarks, i.e. critical points

Focus on the last part First, the user is asked to choose an integer between 0 and 4. This will define the condition for a cell to be selected as the play cell. Then, the program enters a 'while True' loop in which the following events occur:

- the camera captures frames
- lines are drawn above the frames to represent the different cells
- the hand is detected and the coordinates of critical points of the hands are located and drawn on the frames
- the coordinates of the middle of the hand are computed using the middle of the coordinates of the points 2 and 17.
- the number of raised fingers is computed by looking at the relative positions of the points (we look at the x coordinate for the thumb and the y coordinate for the other 4 fingers ; if the tip of the finger (except for the thumb) is higher (has a lower y coordinate) than the point below it, then the finger is considered not raised. The same logic applies for the thumb but for the x coordinate (hence the need to distinguish left and right hand because the symbol $<$ for the left hand will become a $>$ for the right hand))
- we check if the darkmode is activated. If the answer is yes, then the chosen cell will be highlighted in dark blue (reminding the user of the midnight sky blue of the start menu in darkmode). Otherwise, the chosen cell will be highlighted in crystal sky blue (just like the sky in the start menu in light mode)
- Finally, if the number of raised fingers equals the number the user chose at the beginning, then the code looks for the cell in which the middle of the hand is and highlights this cell (in dark or light blue depending on whether the darkmode is activated or not)

2.4 Pillar 4: Communication protocol between Raspberry Pi for one-to-one play - Lilian VARINOT

For this project, we had to create a common communication protocol with the other groups so that we could each play the game on our own devices. We have chosen to communicate via WiFi using the TCP protocol. We also wanted to keep communication simple to code and use, while guaranteeing security. That's why there are only 5 different messages, with 3 error messages, listed in this diagram:

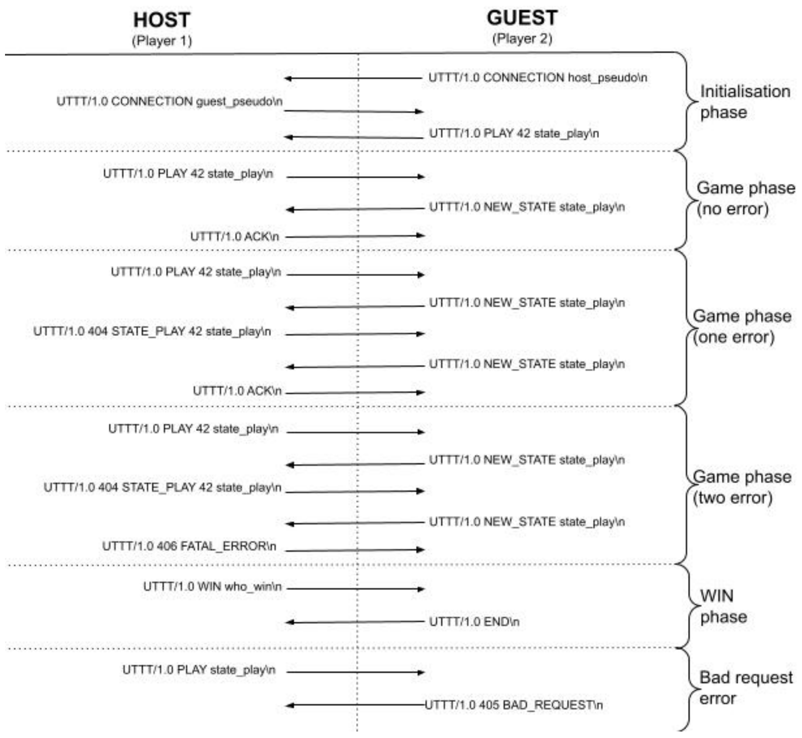
HOST
(Player 1)
GUEST
(Player 2)
UTTT/1.0 CONNECTION host_pseudo\n
UTTT/1.0 CONNECTION guest_pseudo\n
UTTT/1.0 PLAY 42 state_play\n
Initialisation phase
UTTT/1.0 PLAY 42 state_play\n
UTTT/1.0 NEW_STATE state_play\n
UTTT/1.0 ACK\n
Game phase (no error)
UTTT/1.0 PLAY 42 state_play\n
UTTT/1.0 NEW_STATE state_play\n
UTTT/1.0 404 STATE_PLAY 42 state_play\n
UTTT/1.0 NEW_STATE state_play\n
UTTT/1.0 ACK\n
Game phase (one error)
UTTT/1.0 PLAY 42 state_play\n
UTTT/1.0 NEW_STATE state_play\n
UTTT/1.0 404 STATE_PLAY 42 state_play\n
UTTT/1.0 NEW_STATE state_play\n
UTTT/1.0 406 FATAL_ERROR\n
Game phase (two error)
UTTT/1.0 WIN who_win\n
UTTT/1.0 END\n
WIN phase
UTTT/1.0 PLAY state_play\n
UTTT/1.0 405 BAD_REQUEST\n
Bad request error

Figure 9: Communication protocol

3 Major group decisions

3.1 Management choices

Heart Our tutor suggested that we make a main program, the heart of the code, in which we import and use each of our separate parts of the code. This was implemented by us a bit late so the separation between the interface and the heart isn't perfect, but we still made a main code which mainly handles logic decisions and calls to the different parts such as hand tracking or communication. This works using a main loop which calls different menus depending on variables we can change by clicking on buttons.

Work distribution Although we each have a pillar we are responsible for, there is work which isn't exactly part of a specific pillar, such as making the poster and final report or working on the heart of the code. Since the heart and the interface were developed together, the heart was made mainly by the person responsible for the pillar 1 until everyone had to import their own part of the code into it, then everyone who imported something worked on implementing it properly into the existing structure.

For the extra work outside of coding such as the carbon report we chose to do, the monthly reports and the deliverables, we tried to be fair by giving those who had done the least work extra tasks to do. This ended up not working out very well, so everyone contributed at least using the knowledge they had acquired through their work during this project.

3.2 Choice of the theme and name

The idea of a retro and pixel art style for the game came to us quite naturally. Indeed, thanks to the skills of out talented design engineer Manon Galtier, we knew that a pixel art aesthetic was going to be a huge success. Then, we played with the name of the game (Ultimate Tic Tac Toe) and gave it a little twist to remind the player of the nostalgic and simplistic design. That's how the name Super Retro Tic Tac Toe was born.

4 Difficulties encountered and potential upgrades

4.1 Library importation

Issues along the way Some issues have been encountered during the importation of the pygame library but they were fixed quite swiftly. Indeed, it was a matter of compatibility between the version of Python and the library so we just had to update the version of Python we were using. However, the main issue occurred when trying to import the libraries required for the object-tracking section of this project, the 3 main libraries being cv2, Picamera2 and mediapipe. When trying to install the libraries no major error happened but when running our code, an error message announced that the libraries were not imported. After a meticulous analysis of the code and the way we imported the libraries, we decided to install the libraries differently (using "sudo apt get install" command instead of the "pip" command) but the problem remained the same. So we decided to run a very short and basic code, using individually each library but still an error occurred in one of the importations. After several meetings with our object-detection tutor Chiara Galdi, we couldn't find the root of the problem and decided to reboot the Raspberry Pi completely, losing the importation of the libraries already installed. It turned out to be a complete success! We were finally able to import the needed libraries and start running, testing and debugging the object-tracking code we had written since.

What would we do differently if we started this project again now? Looking back at the events, I would definitely be less afraid to reboot the Raspberry Pi and start all over again because even with rebooting the Raspberry Pi, I don't start from zero; I have learnt many things along the way which make me more efficient and aware of the possible issues to come along the way.

4.2 AI choice and depth

The difficulty of installing libraries steered us towards the minimax algorithm. The computing speed of the Raspberry Pi also greatly limited the depth of our algorithm.

Issues along the way Teamwork was difficult to establish with those responsible for the other pillars (object tracking, interface and communication). Indeed, the game conventions were not clearly defined at the start of the project, resulting in a large gap between the interface's view of the game (9x9 table) and the AI's view of the game (81-character string). However, by updating the two game states simultaneously using translation functions, we were able to overcome this problem and reconcile the AI and the interface.

What would we do differently if we started this project again now? If we were starting the project again today, I think the most important thing before we started would have been to clearly define a game state and document our code more. Indeed, a lot of time was wasted trying to understand other people's code.

4.3 CPU capacity of the Raspberry Pi

Issues along the way We have been lucky enough to not have any direct issue caused by the CPU capacity and the performance of the Raspberry Pi. However, we always had to keep in mind that the CPU capacity of the Raspberry Pi was very limited and overflowing it with to many libraries and unoptimized codes would make it crash or at least greatly damage its performance. That's why we tried to optimize our code as much as we could when we were coding (both in terms of memory space and in computing time). This prevented us from using a code we had implemented that used the center of the hand as a mouse controller. Indeed, thanks to object-tracking and hand recognition, the user would have been able to detect their hand and make it control the mouse to click on the different buttons that appear on the screen. However, this implied another library and a memory-guzzling code, which was something we couldn't afford with our Raspberry Pi.

4.4 Communication Protocol

Issues along the way The communication group had trouble agreeing on a common protocol for the communication during a game between two different Raspberry Pi. Although they had decided quite soon that they would implement a TCP connection, how to manage the socket packets and how to handle the errors was a very controversial subject during most of the semester. Unfortunately, we waited for a common protocol to be agreed on to start coding and implementing the communication (although the main idea of the code was already written, some variables and errors handling were not implemented yet). Despite the efforts of the communication member of our team, the protocol wasn't moving forward. So the communication part of this project was put on a standstill for a certain period of time.

What would we do differently if we started this project again now? With the knowledge acquired now, I would insist more to convince the other communication members on how great my ideas were instead of letting everyone disagree on every proposition but not actually coming up with an idea or a plan that everyone could work with. That would allow us and every other group to start working on their communication code sooner and enable us to handle more errors like the loss of connection for instance.

4.5 Interface

Issues along the way The provided code for the ultimate tic tac toe wasn't ideal for our game's development. This caused a few issues both to fix bugs in the game, mostly due to the fullscreen mode, and for the other pillars' integration into the game.

What would we do differently if we started this project again now? It would be wiser to study more in detail how the provided code works and to adapt it to our needs before going forward. Trying to work more closely between the different pillars could also have helped.

5 Conclusion

5.1 How did we do during this project?

Group work There were times when we had issues with group work, including when certain group members were not available to work with the rest of the group due to personal circumstances. This hindered our progress on the project and stopped us from making a better game, which is unfortunate.

Individual work Here is a short breakdown of individual work in our group during this project:

- Lilian Varinot: Worked on communication, fixing the heart of the code, creating the connection menus, adapting our code to the Raspberry Pi and linking the different parts of the code together.
- Sean Rolly: Worked on object tracking, the presentation and the poster. He was a major part of the decision-making process in the group and supported the other group members both in their work and in their decision-making process for their respective responsibilities.
- Youri Halmaert: Made a functional AI and helped for the adaptation of the code to the Raspberry Pi.
- Manon Galtier: Worked on the interface and the heart of the code, worked on the report and its reformulation. Made the initial version of the game and helped implement the different pillars into it. Made the musics and designs for the game.
- Nils Deghilage: Made the carbon report and a first version of the AI.

5.2 Other comments

About the project's organization from our teachers Unfortunately, there were a few miscommunications which ended up preventing us from meeting the teachers' initial expectations. An example of this is the object tracking side of the project. The teachers expected a kind of object tracking unrelated to hand tracking, but most groups including ours did not understand this and ended up working on hand tracking.

Another issue we had was with the importation of the necessary libraries onto our Raspberry Pi: the expert who was supposed to help us with issues such as this one did not manage to help us quickly, which resulted in issues with the testing of our code on the Raspberry Pi.