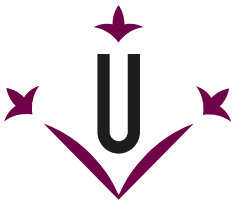
Universitat de Lleida
Enginyeria Informàtica
Embedded and Ubiquitous Systems
Prof. Fernando Guirado Fernández



# e-Game Board

## Meeple Showdown

### Engineering Book

Jordi García Ventura
Christian López García

January 10, 2025

# Contents

# 1 Project Milestones

The project needs to be done between the 19th of October 2024 and the 13th of January 2025. We identify five main features that need to be developed:

- Game Design
- MQTT Broker
- Meeple
- Operation base
- Master

All the tasks, subtasks, time estimations and dependencies are shown in the Gantt chart in Figure 1.

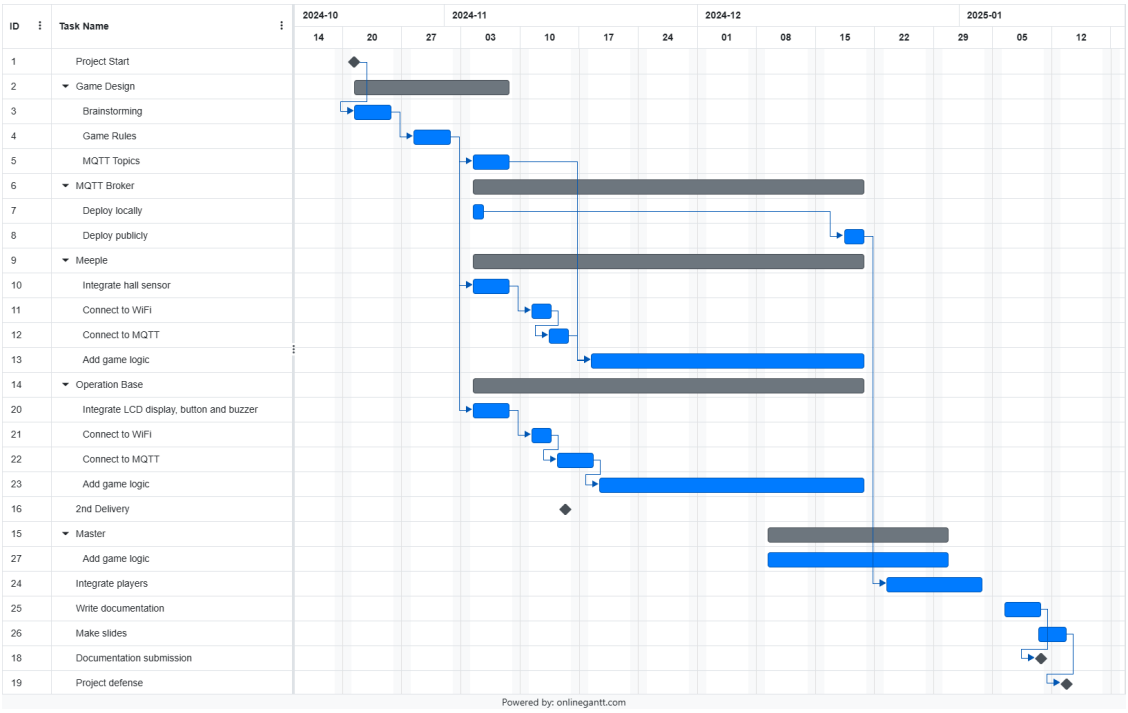



Figure 1: Gantt chart of the project

Regarding the asignations, the *Game Design*, *Master* and *MQTT Broker* are going to be done by all members of the different teams. Then, each team is responsible of programming their *Meeple* and *Operation base*. In our case, Christian will do the *Meeple* and Jordi will do the *Operation base*.

Finally, we are going to expose the MQTT Broker in a public server in order to test the communication between the devices of all the groups remotely.

# 2 Component Design

## 2.1 Meeple

The meeple is the component that represents the player's position on the board and provides feedback about player movement, death, and turn roles. It achieves this using two LEDs (a green one and a yellow one) and a hall sensor. Communication with the game controller is facilitated through the MQTT protocol.

The logic governing the meeple is depicted in Figure 2.

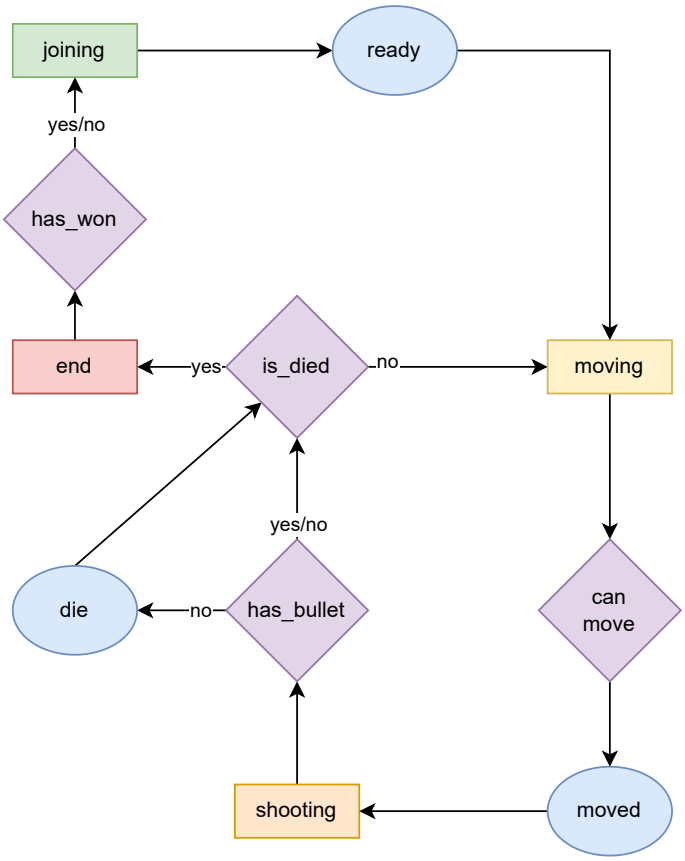



Figure 2: Meeple logic flow diagram

Developing the meeple required approximately 10 hours and was completed solely by Christian. While the component was relatively simple to design, my limited experience with electronics and C++ made the process more time-consuming, particularly when trying to create scalable and understandable code.

As mentioned earlier, the most challenging part was ensuring the C++ code worked as intended. Another difficulty was achieving the desired "blinking" effect for the green LED,

which required fine-tuning to function properly.

## 2.2 Operation base

The operation base is responsible for determining, when it is its turn, whether the player should shoot or not. Additionally, it provides feedback about the game state through an LCD display, as the operator cannot directly see the game board.

The logic governing the operation base, which relies on MQTT topic subscription and publication, is illustrated in Figure 3.

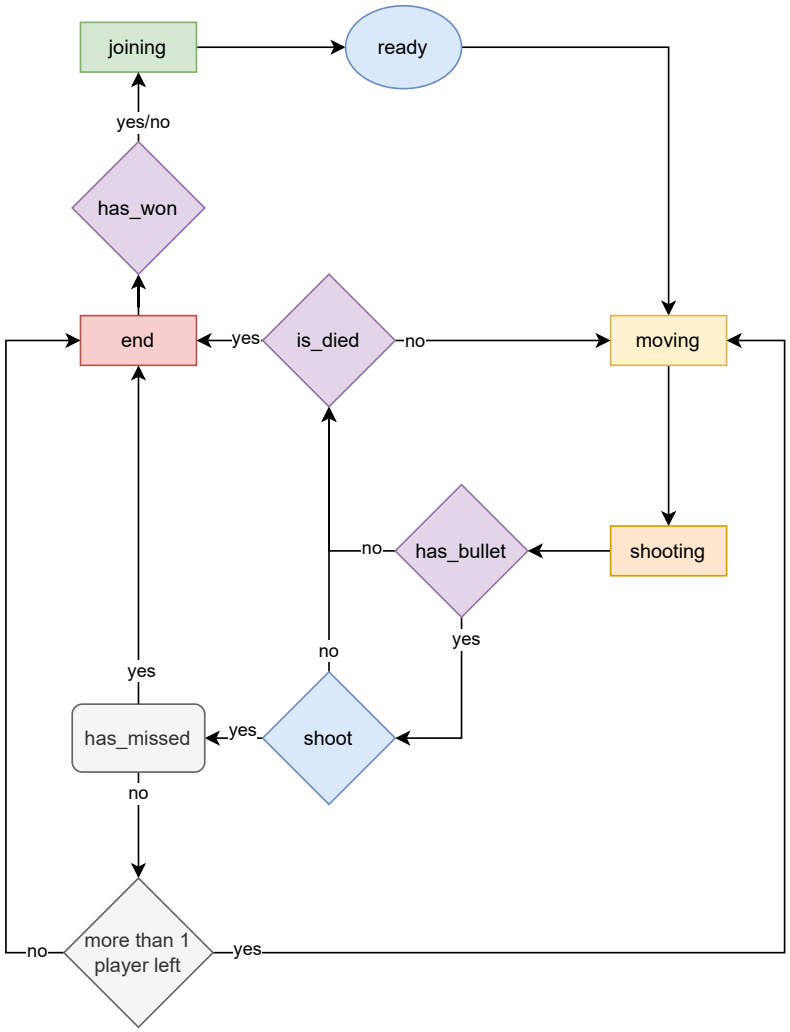



Figure 3: Operation base logic flow diagram

Developing the operation base required approximately 30 hours and was completed entirely by Jordi.

Initially, all programming was done using Arduino. However, I later migrated most of the code to FreeRTOS. During development, I encountered several challenges. One issue was with the button, which behaved erratically. This was not solely due to switch bounce; interference was also caused by the weak internal pull-up resistorsimply bringing a finger close to the button without touching it affected the readings. Adding an external pull-up resistor resolved this issue.

Another problem involved WiFi connectivity. The ESP32 module was unable to connect to my network. After investigation, I discovered that the ESP32 does not support 5GHz networks, requiring a switch to a 2.4GHz network.

I also faced difficulties connecting to the MQTT broker. The issue stemmed from the broker's IP address changing, but the unclear error messages made diagnosing the problem challenging.

Lastly, I dedicated considerable time to refactoring the code, improving its readability and scalability for future maintenance and enhancements.

### 2.3 Game Controller

The game controller is the core component of the game, responsible for managing the game flow, player turns, game state, and communication with the player feedback system. It operates as a Docker container running a Python script that uses the MQTT protocol in a reactive manner to enable bidirectional communication with the players.

The logic governing the game controller is illustrated in Figure 4.

Developing the game controller took approximately 20 hours and was completed solely by Christian. We chose Python for the development because it didnt need to run on a microchip, so performance wasnt a major concern. Additionally, Im more familiar with Python, despite having no prior experience with the paho-mqtt library. The most challenging aspect was designing a clean and maintainable project structure. I encountered issues with circular imports while trying to make the code both modular and understandable, and I feel that I struggled with this in some areas.

I developed the meeple and the game controller in parallel, which allowed me to test communication between the two components. While this was beneficial, I was unable to test the game controller with any operation base, so Im uncertain if it functions as expected in that regard.

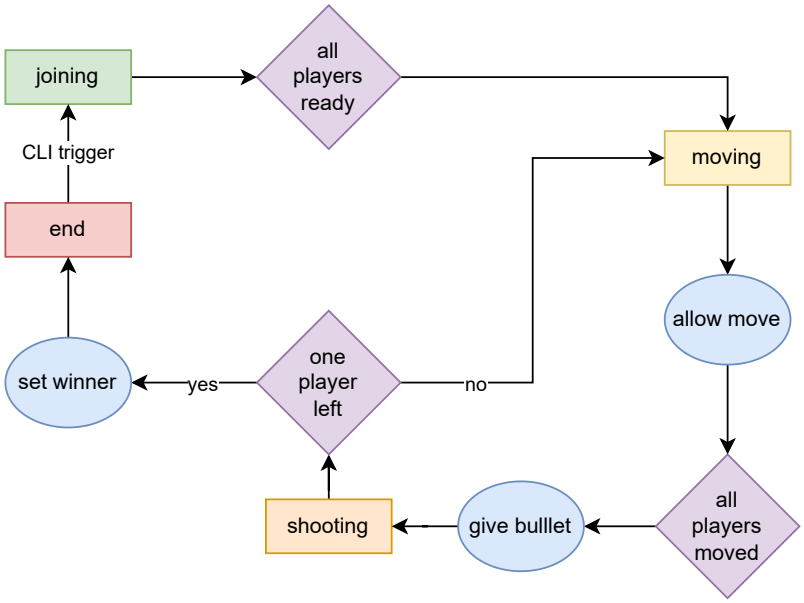



Figure 4: Master logic flow diagram

## 2.4 MQTT Broker

The MQTT broker is a critical component of the game, enabling seamless communication between all other components. It operates as a Docker container running the Mosquitto MQTT broker.

Developing the MQTT broker required approximately 4 hours and was completed solely by Christian. The most challenging aspect was implementing authentication for the broker. The team also encountered an issue where GitHub altered the line endings in configuration files from LF to CRLF, which caused unexpected failures in the broker's functionality.

Ultimately, Jordi deployed the broker to the cloud, allowing the team to test the game remotely without needing to meet in person

# 3 e-Game Rules

## 3.1 Objective

Be the last surviving meeple on the board.

## 3.2 Players

Each team consists of two people:

- **Mover (Pawn Controller)**: Controls the movement of the meeple on the board.
- **Operator (Strategist)**: Makes decisions on whether to shoot or pass the bullet without seeing the board.

## 3.3 Game Components

- **Magnetic board(s)**: Includes hidden magnets under each position.
- **Meeples (pawns)**: Each player team has one meeple. It has two LEDs: one for knowing who has the bullet and another for magnet detection feedback.
- **Operation Base**: Includes LCD display, push button, and other components for decisions made by operators.
- **Broker**: Handles the game logic and the state.

## 3.4 Setup

- **Initial Meeple Placement**: Each pawn begins in a random position on the board. This is decided before the game starts.
- **Bullet Assignment**: One player is randomly assigned the Bullet at the start of the game.

## 3.5 Gameplay Overview

The game proceeds in rounds, with the following steps repeated until only one meeple remains:

1. **Movement Phase:**
    - Pawns move one space in one of four directions: up, down, left, or right.
    - Movement order is decided randomly at the start of each round.
2. **Decision Phase:**
    - The operator of the team with the Bullet decides if they want to shoot.
    - The operator cannot see the board, but may communicate with other pawns (within 60 seconds).

- Shooting affects all meeples in the same row, column, or diagonal as the operator's meeple (Figure 5).
  - **Hit**: If the shot hits any meeples, they are eliminated.
  - **Miss**: If no other meeple is hit, the shooting operator is eliminated.
- If the operator chooses not to shoot, there is a small probability of dying.

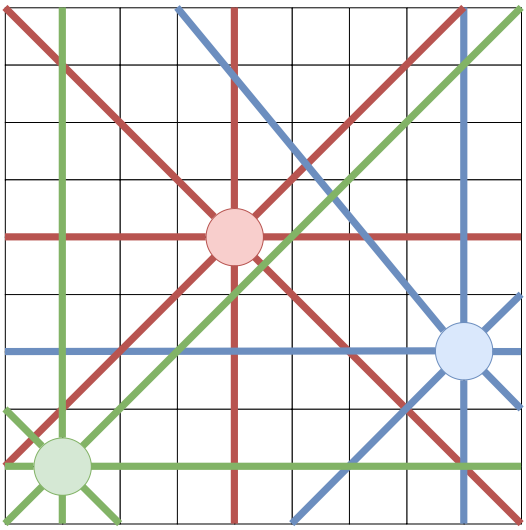


Figure 5: Shooting directions examples

### 3.6 Additional Rules

- **Time Constraints**: Operators must decide to shoot or not within 60 seconds. If time runs out, the decision is to not shoot.
- **Lives**: Each meeple has 1 life. If they are hit, they are eliminated.
- **Random Player Selection**: When the bullet is reassigned, the game selects a random player who hasnt been eliminated.

### 3.7 End Condition

The game continues until only one meeple remains. This team is declared the winner.

# 4 Conclusions

The game we developed is functional, but it currently does not account for certain unpredictable player behaviors, such as publishing to topics at inappropriate times.

Developing with microcontrollers is undoubtedly challenging, yet we successfully created an engaging and enjoyable game that integrates all the essential components of an embedded system. Additionally, we achieved interconnectivity among multiple devices through networking, unlocking numerous possibilities for future projects. For instance, the network capabilities could be leveraged to input user data via a captive portal rather than hardcoding network credentials into the source code. Another potential enhancement could be implementing Over-The-Air (OTA) updates, allowing remote updates to the devices' firmware.

A significant challenge we faced was establishing a robust communication standard, including a well-defined contract and MQTT topic structure, to ensure seamless interaction between devices. This standardization enabled different teams to work independently while ensuring all components integrated cohesively. Achieving this required a clear understanding of the project requirements and the components involved.

One obstacle that slowed our progress was the time required for compiling and uploading code to the microcontrollers, particularly with the ESP-01, which requires flashing via a USB-to-Serial adapter. Using a development-friendly board with a micro-USB interface could have expedited our iteration process. Once the code reached a more stable state, we could then transition to the ESP-01 for deployment.

Overall, this project provided valuable insights into the complexities of embedded systems and offered a foundation for exciting future developments.