

Yanis DEZZAZ

Guillaume GREDER

Xavier KNOEPFFLER-POUT

Théo MARCHAND

---

## Introduction

### Présentation du projet

Ce projet vise à intégrer un gant connecté dans un environnement virtuel en utilisant un ESP32 pour interagir avec le jeu Minecraft. L'objectif est d'activer des super pouvoirs lors d'un mini-jeu afin de rendre l'expérience de jeu immersive et interactive.

### Liste du matériel

Au cours de cette SAE, le matériel à notre disposition comprend :

- 2 Raspberry Pi
- Borne Linksys
- Ordinateurs portables
- 2 ESP32
- Câbles
- Platine de montage
- Boutons

### Définition du Projet

Des user stories nous ont été imposées pour définir le but de la SAE. Voici les user stories :

1. Deux joueurs vont jouer en réseau à Minecraft. Quand un joueur joue, il voudra récupérer le plus rapidement possible un diamant qui aura été caché de manière aléatoire afin de remporter la partie.
  - Besoin d'un réseau connectant les Raspberry et les ESP.
  - Création d'un serveur Minecraft pour permettre plusieurs connexions simultanées.
  - Mise en place d'un mini-jeu qui cache aléatoirement un diamant.
2. Chaque joueur dispose d'un gant connecté qui lui donnera accès à des super pouvoirs et des pièges (malus) contre l'adversaire dans le jeu Minecraft.
  - Création de super pouvoirs et de pièges dans Minecraft.
3. Chaque gant dispose d'un bouton sur chaque doigt (sauf le pouce), chaque appui sur un bouton activera une action dans le jeu. Les gants seront autonomes en énergie avec la

- connexion d'une batterie sur chaque gant.
- Création d'un gant connecté avec des boutons.
  - Mise en place d'un serveur MQTT.
  - Mise en place d'un script Python.
4. Lorsqu'un joueur se déplace, il devra savoir s'il se rapproche ou s'éloigne du diamant. Un texte devra s'afficher pour l'informer. S'il est très proche, le texte devra indiquer aux 2 joueurs le gagnant.
- Création d'un système affichant la distance entre le joueur et le bloc de diamant.
  - Mise en place d'une détection de victoire et affichage aux joueurs.
5. On pourra faire un réglage du gant afin d'assigner des super pouvoirs à chaque doigt, le choix du délai anti-rebond, éventuellement du nombre de joueurs.
- Rendre le gant stable à son utilisation.
  - Augmenter le nombre de joueurs.
  - Création de super pouvoirs uniques à chaque doigt.

## Règles du Jeu

Nous allons affiner les règles du jeu pour avoir une meilleure vue du projet.

### But du jeu

Quatre joueurs vont jouer en réseau à Minecraft. L'une des deux équipes devra récupérer le plus rapidement possible un diamant qui aura été caché de manière aléatoire afin de remporter la partie. L'autre équipe devra protéger ce bloc de diamant.

### Règles du Jeu

- Quatre joueurs, par équipes de deux.
- La partie se termine lorsque le bloc de diamant est détruit ou après 10 minutes.
- Une équipe de défenseurs et une équipe d'assaillants.
- Chaque équipe aura un combattant chargé de casser/défendre le bloc et un sorcier qui activera les pouvoirs tout en assistant l'autre joueur.

### Sommaire :

#### I. User Story 1

##### a. Mise en place d'un réseau

##### b. Serveur Minecraft

##### c. Bloc de diamant

#### II. User Story 2

##### a. super pouvoir

#### III. User Story 3

##### a. Serveur MQTT

b. Détection de bouton

c. Connexion WIFI

d. Publication MQTT

e. Script Python

f. Énergie

IV. User Story 4

a. Détection de victoire

b. Afficher la distance du bloc de diamant

V. Bonus

a. Texture Pack

b. Map

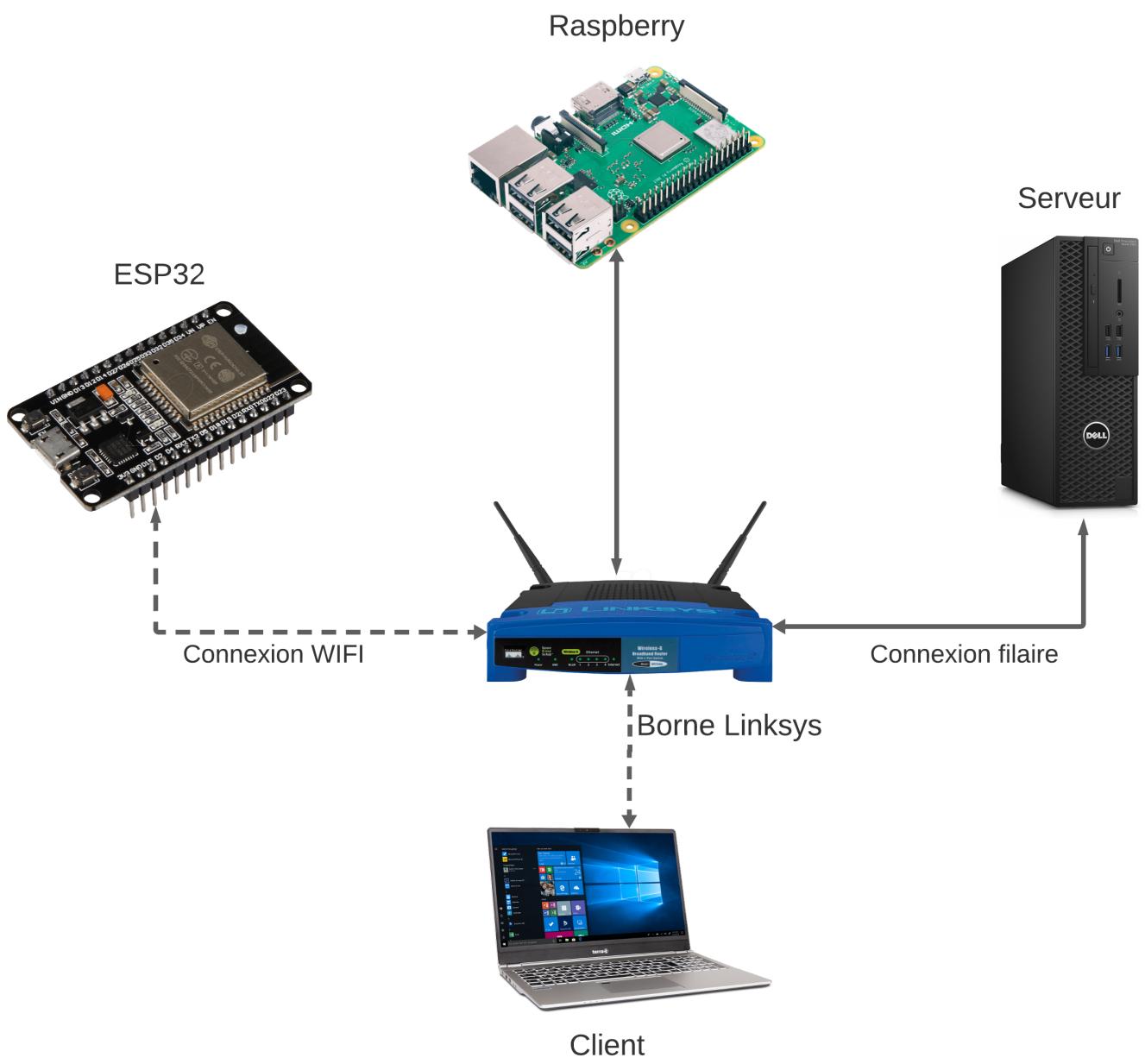
c. Outil

VI. Gestion de Projet

VII. Conclusion

## I. User Story 1

### a. Mise en place d'un réseau



*Schéma de notre réseau*

L'architecture de notre réseau est représentée ci-dessus. Le client peut être un ordinateur personnel ou la Raspberry, avec un client Minecraft ou MQTT pour des tests, et le serveur est effectivement une Raspberry. Cependant, nous avons utilisé nos ordinateurs personnels pour les tests afin de ne pas être limités par la puissance de la Raspberry. Pour la configuration de la borne, nous avons effectué une réinitialisation et changé le SSID et le mot de passe, ainsi que la sécurité de la connexion en WPA Personnel pour que l'ESP puisse se connecter.

## b. Serveur Minecraft

L'architecture matérielle de la Raspberry limite les versions de Minecraft que nous pouvons utiliser. Les versions "Classic" ne sont pas adaptées à la Raspberry. Cependant, grâce au launcher (application qui permet de lancer Minecraft) Prism Launcher (<https://prismlauncher.org/>), nous pouvons avoir accès à toutes les versions de Minecraft. Nous l'avons installé depuis un app store dédié à la Raspberry, Pi-Apps (<https://pi-apps.io/>). Nous avons utilisé la version Fabric de Minecraft. Fabric est un "mod loader", une version modifiée

qui permet d'ajouter des mods (rajoute de contenu) qui peut être utile à la création de mini-jeu ou une aide de jeu. Par exemple, à la construction de map, à l'affichage de données utiles au débogage. Voici les commandes nécessaires pour faire fonctionner le serveur :

```
sudo apt install openjdk-17-jre
mkdir ~/Documents/ServeurMinecraft
cd ~/Documents/ServeurMinecraft
wget https://meta.fabricmc.net/v2/versions/loader/1.20.4/0.15.3/1.0.0/server/jar
java -Xmx2G -jar fabric-server-mc.1.20.4-loader.0.15.3-launcher.1.0.0.jar
nogui nano eula.txt
### Remplacer false par true
java -Xmx2G -jar fabric-server-mc.1.20.4-loader.0.15.3-launcher.1.0.0.jar
nogui
```

Et nous avons un serveur fonctionnel.

## c. Bloc de diamant

Pour placer aléatoirement le bloc de diamant, nous avons utilisé les datapacks. Les datapacks sont un langage natif de Minecraft qui permet de créer des scripts. Nous retrouvons des choses similaires à un langage de programmation ordinaire : le système de classes, de fonctions, de variables, etc. À l'exécution du datapack, nous définissons un tableau de valeurs qui stockera les coordonnées du bloc de diamant avec ces commandes.

```
scoreboard objectives remove coordinates
scoreboard objectives add coordinates dummy
```

La première ligne supprime l'ancien tableau existant, puis nous le recréons. Ensuite, nous allons stocker des coordonnées générées de manière aléatoire dans le tableau avec le joueur x, y et z.

```
execute store result score x coordinates run random roll 0..50
execute store result score y coordinates run random roll 0..50
execute store result score z coordinates run random roll 0..50
```

Il génère un nombre entre 0 et 50. Ensuite, vu que nous ne pouvons pas simplement faire apparaître ce bloc à ces coordonnées, nous allons faire autrement. Nous allons faire apparaître une armure invisible, indestructible et non soumise à la gravité avec un identifiant pour qu'elle soit unique aux coordonnées 0 en x, 0 en y et 0 en z.

```
summon armor_stand 0 0 0 {NoGravity:1b,Invulnerable:1b,Invisible:1b,Tags:[ "diamond" ]}
```

Ensuite, nous ne pouvons pas le faire téléporter d'un coup aux coordonnées, mais nous pouvons utiliser une autre méthode.

```
execute at @e[tag=diamond,limit=1] run tp @e[tag=diamond] ~1 ~ ~
```

Cette ligne s'exécute en tant que l'armure avec l'identifiant diamond et la téléporte de sa position initiale mais ajoute +1 en x. Donc, par exemple, si le nombre généré est 23 en x et que nous exécutons 23 fois, l'armure sera à la coordonnée x:23 y:0 et z:0.

```
execute if score x coordinates matches 1.. at @e[tag=diamond,limit=1] run tp @e[tag=diamond] ~1 ~ ~  
execute if score x coordinates matches 1.. run scoreboard players remove x coordinates 1
```

La première ligne correspond à : "Si x est différent de 0, tu téléportes l'armure à +1 de ses coordonnées x", et la deuxième : "Si x est différent de 0, faire -1 à x".

Donc, on fait ça pour tous les axes (x, y et z) et on rappelle la fonction tant que toutes les valeurs ne sont pas à 0. Quand x, y et z sont à 0, alors nous pouvons exécuter cette commande :

```
execute if score x coordinates matches 0 if score y coordinates matches 0 if score z coordinates matches 0 at @e[tag=diamond] run setblock ~ ~ ~ diamond_block
```

Et on place le bloc de diamant à la position de l'armure, puis nous pouvons faire disparaître cette armure.

## II. User Story 2

### a. super pouvoir

Donc on va créer 4 super pouvoirs. Voici la liste des pouvoirs qu'on souhaite :

- Super Saut
- Super Punch
- Cocon

- Lenteur

Le Super Saut se décompose en deux phase la monté et la descente. Pour la monté nous appliquons un effet de Minecraft qui s'appelle lévitation. Il fait s'envole le joueur.

```
effect give @a[gamemode=survival,team=red] levitation 5 10 true
```

la ligne donne un effet de lévitation au joueur en survie de l'équipe rouge (combattant) pendant 5s de puissance 10 et on cache les effets au joueur (true). Donc le joueur va avoir l'impression d'avoir fait un grand saut mais le problème si la chute est trop importante, il peut mourir de dégât de chute. Donc nous allons données en même temps un effet de chute ralenti pendant plus longtemps.

```
effect give @a[gamemode=survival,team=red] slow_falling 30 1 true
```

Pendant 30 secondes de puissance 1 donner l'effet chute ralenti.

Le Super Punch va être un gant qui inflige plus de dégâts et donne un effet de recul plus important qu'un gant normal.

```
give @a[gamemode=survival,team=red]
stone_sword{HideFlags:1,Damage:129,CustomModelData:1,Enchantments:
[{"id": "minecraft:knockback", "lvl": 10}]} 1
```

On donne une épée au joueur avec l'enchangement knockback de niveau 10 et il ne reste plus que 3 trois à l'épée avant de se casse. Comme ça le joueur ne garde pas indéfiniment cette épée. "Damage:129" signifie que l'épée a été endommager 129 fois et la durabilité de l'arme est de 131. Et pour que ce soit un gant nous utilisons un pack de texture et nous applique le model customisé numéro 1 ou 2 selon l'équipe.

Le cocon crée une cage autour du joueur soit de bloc de magma qui font des dégâts assez léger ou fait apparaître une bulle d'eau selon l'équipe.

```
execute at @a[team=red,gamemode=survival] run setblock ~ ~ ~ water keep
```

On fait apparaître sur l'équipe d'adverse un bloc seulement si l'endroit est vide (ou de l'air fait avec l'option keep) sinon on garde le bloc en place pour éviter de casser le bloc de diamant.

On répète l'opération pour le nombre de bloc à poser.

La lenteur est un pouvoir qui obstrue la vue et ralenti l'ennemi comme avec une malédiction. Pour faire ça nous utilisons deux effets de potions avec l'effet slowness et blindness.

```
effect give @a[team=red,gamemode=survival] slowness 15 3 true  
effect give @a[team=red,gamemode=survival] blindness 10 3 true
```

Donc l'effet de ralentissement est donné pendant 15 secondes et de puissance 3 et d'aveuglement pendant 10 secondes à l'équipe adverse.

## III. User Story 3

### a. Serveur MQTT

Pour installer MQTT sur la Raspberry, utilisez la commande `sudo apt install mosquitto`. Ensuite, créez un fichier `default.conf` dans le répertoire `/etc/mosquitto/conf.d/` avec la configuration suivante :

```
listener 1883  
allow_anonymous true
```

Enfin, lancez le serveur avec la commande `mosquitto -c /etc/mosquitto/conf.d/default.conf`. Notez que lors de l'installation du paquet mosquitto, un serveur est démarré automatiquement, assurez-vous de l'éteindre avec la commande `service mosquitto stop`.

Pour tester le serveur, installez `mosquitto-clients` avec la commande `apt install mosquitto-clients`. Effectuez un test de connexion sur la même machine et ensuite sur le même réseau.

Si vous souhaitez renforcer la sécurité en ajoutant des mots de passe, utilisez la commande `mosquitto_passwd -b passwordfile Utilisateur MotdePasse`. Modifiez le fichier de configuration `/etc/mosquitto/conf.d/default.conf` comme suit :

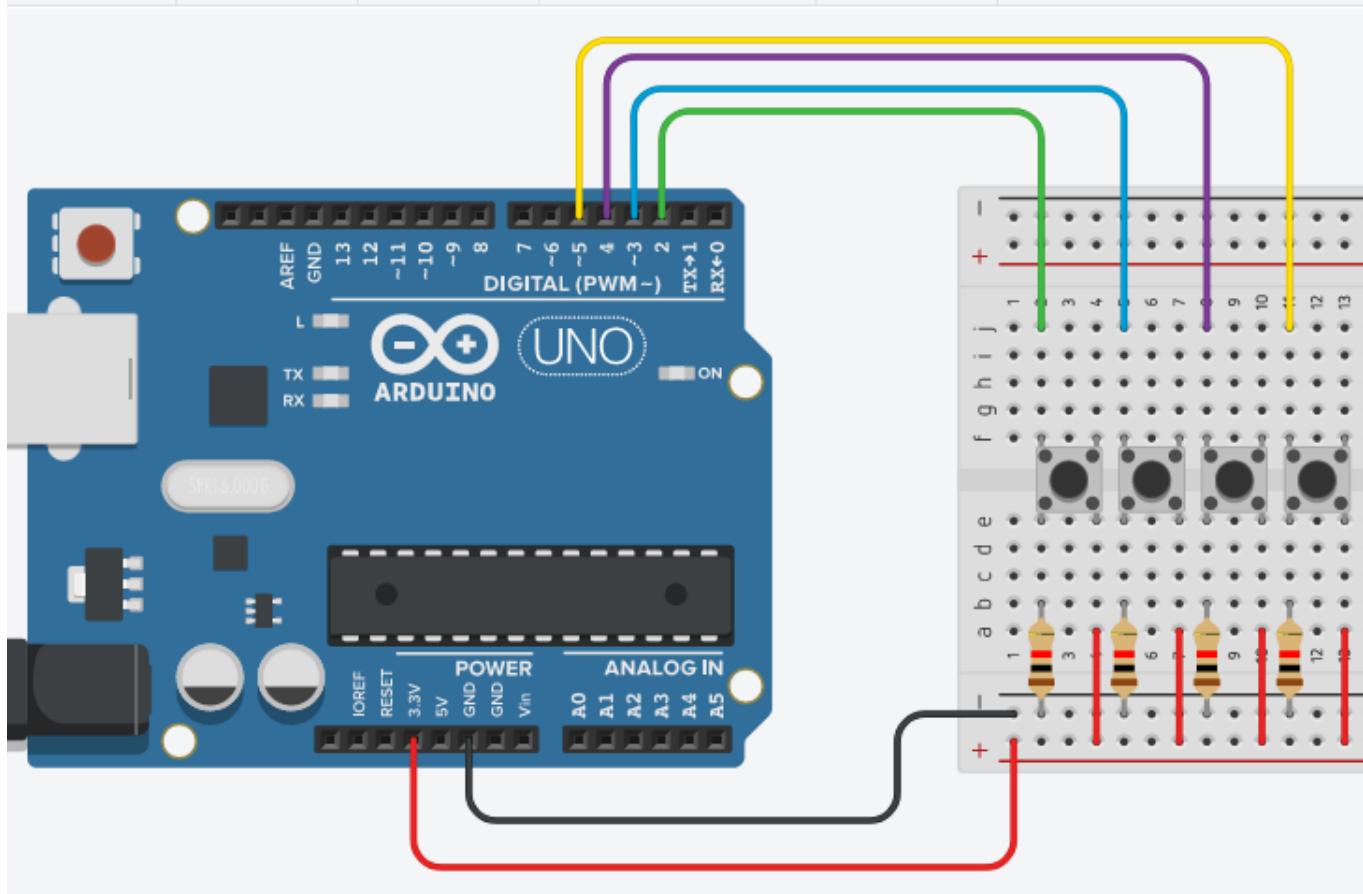
```
per_listener_settings true  
allow_anonymous false  
password_file /etc/mosquitto/passwordfile  
listener 8883
```

Après avoir redémarré le serveur Mosquitto, vous pouvez effectuer des tests avec les commandes suivantes :

```
mosquitto_sub -h localhost -t \'# -v -u Utilisateur -P MotdePasse  
mosquitto_pub -h localhost -m "test" -t topic/test -u Utilisateur -P  
MotdePasse
```

## b. Détection de bouton

La réalisation du gant se base sur le schéma ci-dessous, initialement conçu avec une carte Arduino mais implémenté sur l'ESP32. Le programme détecte la pression des boutons et réagit en conséquence.



```
const int but1 = 34;  
const int but2 = 35;  
const int but3 = 32;  
const int but4 = 33;  
  
void setup()  
{  
    pinMode(but1, INPUT);  
    pinMode(but2, INPUT);  
    pinMode(but3, INPUT);  
    pinMode(but4, INPUT);  
    Serial.begin(921600);  
}
```

```

void loop()
{
    delay(100);
    if (digitalRead(but1) == 1) {
        Serial.println("Bouton 1 pressé.");
        while (digitalRead(but1) == 1) {
            delay(50);
            Serial.print(".");
        }
        Serial.println("Fin de la pression");
    } else if (digitalRead(but2) == 1) {
        Serial.println("Bouton 2 pressé.");
        while (digitalRead(but2) == 1) {
            delay(50);
            Serial.print(".");
        }
        Serial.println("Fin de la pression");
    } else if (digitalRead(but3) == 1) {
        Serial.println("Bouton 3 pressé.");
        while (digitalRead(but3) == 1) {
            delay(50);
            Serial.print(".");
        }
        Serial.println("Fin de la pression");
    } else if (digitalRead(but4) == 1) {
        Serial.println("Bouton 4 pressé.");
        while (digitalRead(but4) == 1) {
            delay(50);
            Serial.print(".");
        }
        Serial.println("Fin de la pression");
    }
}

```

```

Bouton 1 pressé.
.....fin de la pression
Bouton 1 pressé.
.....fin de la pression
Bouton 1 pressé.
..fin de la pression
Bouton 1 pressé.
.....fin de la pression
Bouton 1 pressé.
.fin de la pression

```

Avec ce montage utilisant un bouton poussoir de type pulldown, aucune perturbation n'est constatée, et l'appui sur le bouton génère une réponse unique. La réactivité de la réponse est qualitative.

## c. Connexion WIFI

Le programme utilise le `WifiClientBasic` trouvé dans les exemples de codes Arduino pour tester la connexion au réseau. Assurez-vous de remplacer les valeurs `WIFI_SSID` et `WIFI_PASSWORD` par celles de votre réseau.

```
#include <WiFi.h>
#define WIFI_SSID          "Votre_SSID"
#define WIFI_PASSWORD      "Votre_Mot_de_Passe"

WiFiClient client;

void setup()
{
    byte mac[6];
    WiFi.macAddress(mac);
    device.setUniqueId(mac, sizeof(mac));
    Serial.begin(921600);
    delay(10);

    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.println();
    Serial.println();
    Serial.print("Attente du WiFi... ");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println();
    Serial.println("Connecté au réseau");
    Serial.println("");
    Serial.println("WiFi connecté");
    Serial.println("Adresse IP : ");
    Serial.println(WiFi.localIP());
    delay(500);
}

void loop(){}
```

Voici le résultat après exécution :

```
WiFi connected
IP address:
192.168.1.125
```

## d. Publication MQTT

Ce programme permet de publier des messages MQTT après avoir configuré la connexion au broker.

```
#include <WiFi.h>
#include <ArduinoHA.h>
#define BROKER_ADDR      IPAddress(192, 168, 1, 115)
#define WIFI_SSID         "Votre_SSID"
#define WIFI_PASSWORD     "Votre_Mot_de_Passe"

WiFiClient client;
HADevice device;
HAMqtt mqtt(client, device);

void setup()
{
    byte mac[6];
    WiFi.macAddress(mac);
    device.setUniqueId(mac, sizeof(mac));
    Serial.begin(921600);
    delay(10);

    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.println();
    Serial.println();
    Serial.print("Attente du WiFi... ");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println();
    Serial.println("Connecté au réseau");
    Serial.println("");
    Serial.println("WiFi connecté");
    Serial.println("Adresse IP : ");
    Serial.println(WiFi.localIP());
    delay(500);

    mqtt.begin(BROKER_ADDR, 8883, "dracaufeu", "carapuce");
}

void loop()
{
    mqtt.loop();
    mqtt.publish("ESP/test/", "1");
    delay(100);
}
```

Après exécution nous pouvons voir les messages (ESP/test/ 1)envoyer via un souscription sur un client.

Une fois que chaque programme fonction nous avons tous assemblés et ajouter une fonctionnalité qui permet d'allumé une LED leur de l'envoyer d'un messages MQTT spécifique pour vérifier que l'ESP est bien connecté.

```
#include <WiFi.h>
#include <ArduinoHA.h>

// Définition des constantes
#define BROKER_ADDR      IPAddress(192, 168, 1, 115) // Adresse IP du broker
MQTT
#define WIFI_SSID         "plouf"   // SSID du réseau WiFi
#define WIFI_PASSWORD     "dracaufeu" // Mot de passe du réseau WiFi
#define LED_PIN           2 // Broche de la LED

// Initialisation des objets et des variables
WiFiClient client; // Objet client WiFi
HADevice device; // Objet représentant le périphérique Home Assistant
HAMqtt mqtt(client, device); // Objet MQTT avec le client WiFi et le
périphérique Home Assistant

// Déclaration d'un interrupteur Home Assistant
HASwitch ledSwitch("led");

// Broches pour les boutons
const int but1 = 34;
const int but2 = 35;
const int but3 = 32;
const int but4 = 33;
int firstload = 0; // Variable pour gérer le premier chargement

// Fonction appelée lorsqu'une commande d'interrupteur est reçue
void onSwitchCommand(bool state, HASwitch* sender)
{
    Serial.print("Received switch command: ");
    Serial.println(state ? "ON" : "OFF");
    digitalWrite(LED_PIN, state ? HIGH : LOW);
    sender->setState(state); // rapporte l'état à Home Assistant
}

// Configuration initiale du programme
void setup()
{
    // Configuration des broches des boutons en tant qu'entrées
    pinMode(but1, INPUT);
    pinMode(but2, INPUT);
    pinMode(but3, INPUT);
```

```
pinMode(but4, INPUT);

// Obtention de l'adresse MAC pour identifier de manière unique le
périphérique
byte mac[6];
WiFi.macAddress(mac);
device.setUniqueId(mac, sizeof(mac));

Serial.begin(921600);
delay(10);

// Connexion au réseau WiFi
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.println();
Serial.println();
Serial.print("Waiting for WiFi... ");
while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
}
Serial.println();
Serial.println("Connected to the network");
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
delay(500);

// Configuration du périphérique Home Assistant
device.setName("ESP-AQUA");
device.setSoftwareVersion("1.0.0");

// Configuration de la LED
pinMode(LED_PIN, OUTPUT);
digitalWrite(LED_PIN, LOW);

// Configuration de l'interrupteur Home Assistant
ledSwitch.onCommand(onSwitchCommand);
ledSwitch.setName("LED Switch"); // optionnel

// Configuration de la connexion MQTT
mqtt.begin(BROKER_ADDR, 8883, "dracaufeu", "carapuce");
}

// Boucle principale du programme
void loop()
{
    // Gestion des événements MQTT
    mqtt.loop();
```

```

// Publication d'un message lors du premier chargement
if (firstload == 0) {
    mqtt.publish("ESP/AQUA/", "1");
    firstload = 1;
}

// Détection des pressions des boutons
delay(100);
if (digitalRead(but1) == 1) {
    Serial.println("Bouton 1 pressé.");
    mqtt.publish("ESP/AQUA/buttonPress", "1");
    while (digitalRead(but1) == 1) {
        delay(50);
        Serial.print(".");
    }
    Serial.println("Fin de la pression");
} else if (digitalRead(but2) == 1) {
    Serial.println("Bouton 2 pressé.");
    mqtt.publish("ESP/AQUA/buttonPress", "2");
    while (digitalRead(but2) == 1) {
        delay(50);
        Serial.print(".");
    }
    Serial.println("Fin de la pression");
} else if (digitalRead(but3) == 1) {
    Serial.println("Bouton 3 pressé.");
    mqtt.publish("ESP/AQUA/buttonPress", "3");
    while (digitalRead(but3) == 1) {
        delay(50);
        Serial.print(".");
    }
    Serial.println("Fin de la pression");
} else if (digitalRead(but4) == 1) {
    Serial.println("Bouton 4 pressé.");
    mqtt.publish("ESP/AQUA/buttonPress", "4");
    while (digitalRead(but4) == 1) {
        delay(50);
        Serial.print(".");
    }
    Serial.println("Fin de la pression");
}
}

```

Le programme fonctionne bien le broker MQTT reçoit toutes les informations et on peut allumer la LED avec une publication.

## e. Script Python

Ce script Python surveille les messages MQTT du gant connecté et envoie des commandes au serveur Minecraft via RCON. Assurez-vous de modifier le fichier `server.properties` du serveur Minecraft avec les lignes suivantes :

```
rcon.port=25575
enable-rcon=true
rcon.password=MotdePasse
```

```
import paho.mqtt.client as mqtt
from mcrcon import MCRcon

# Connexion au serveur Minecraft RCON
with MCRcon("192.168.1.117", "dracaufeu") as mcr:
    resp = mcr.command("/say mqtt allume")
    print(resp)

# Initialisation de la connexion au serveur Minecraft RCON
mcr = MCRcon("192.168.1.117", "dracaufeu")
mcr.connect()

# Fonction appelée lors de la connexion au serveur MQTT
def on_connect(client, userdata, flags, rc):
    print("Connecté avec le code de résultat " + str(rc))
    client.subscribe("ESP/#")

# Fonction appelée lorsqu'un message est reçu sur un topic MQTT
def on_message(client, userdata, msg):
    if str(msg.topic) == "ESP/AQUA/buttonPress":
        handle_button_press(msg.payload, "aqua")
    elif str(msg.topic) == "ESP/MAGMA/buttonPress":
        handle_button_press(msg.payload, "magma")
    elif str(msg.topic) == "ESP/AQUA/":
        handle_esp_connection("aqua")
    elif str(msg.topic) == "ESP/MAGMA/":
        handle_esp_connection("magma")
    else:
        print("Topic inconnu")
        return "Erreur"

# Fonction pour traiter la pression d'un bouton
def handle_button_press(payload, team):
    if str(payload) == "b'1'":
        print(f"Bouton 1 pressé pour l'équipe {team}.")
        resp = mcr.command(f"/function {team}:supersaut")
        print(resp)
    elif str(payload) == "b'2'":
        print(f"Bouton 2 pressé pour l'équipe {team}.")
```

```

        resp = mcr.command(f"/function {team}:superpunch")
        print(resp)
    elif str(payload) == "b'3'":
        print(f"Bouton 3 pressé pour l'équipe {team}.")
        resp = mcr.command(f"/function {team}:cocon")
        print(resp)
    elif str(payload) == "b'4'":
        print(f"Bouton 4 pressé pour l'équipe {team}.")
        resp = mcr.command(f"/function {team}:lenteur")
        print(resp)

# Fonction pour traiter la connexion d'un ESP
def handle_esp_connection(team):
    print(f"ESP de la team {team} est connecté.")
    client.publish(f"aha/08d1f9e8e990/led/cmd_t", "ON")

# Initialisation du client MQTT
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# Configuration des informations d'identification pour la connexion au
serveur MQTT
client.username_pw_set(username="dracaufeu", password="carapuce")

# Connexion au serveur MQTT
client.connect("192.168.1.115", 8883, 60)

# Boucle principale pour maintenir la connexion au serveur MQTT
client.loop_forever()

```

## Explications du Code Python

### Importation des bibliothèques

Le code utilise les bibliothèques `paho.mqtt.client` pour la communication MQTT, `mcrcon` pour la communication avec le serveur Minecraft RCON.

### Connexion au serveur Minecraft RCON

Le code se connecte au serveur Minecraft RCON à l'adresse IP "192.168.1.117" avec le mot de passe "dracaufeu". Il envoie ensuite une commande "/say mqtt allume" au serveur Minecraft.

### Initialisation de la connexion au serveur Minecraft RCON

Le code initialise une connexion au serveur Minecraft RCON sans envoyer de commande immédiatement.

## Fonctions MQTT

Deux fonctions (`on_connect` et `on_message`) sont définies pour gérer les événements de connexion et les messages MQTT reçus.

## Traitement des Messages MQTT

La fonction `on_message` analyse les messages reçus sur les topics MQTT "ESP/AQUA/buttonPress", "ESP/MAGMA/buttonPress", "ESP/AQUA/", "ESP/MAGMA/" et appelle des fonctions spécifiques en conséquence.

## Traitement des Pressions de Bouton et Connexions d'ESP

Les fonctions `handle_button_press` et `handle_esp_connection` effectuent des actions spécifiques en fonction des messages reçus, comme l'exécution de commandes Minecraft RCON et la publication de messages MQTT.

## Configuration du Client MQTT

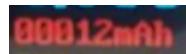
Le client MQTT est configuré avec des informations d'identification.

## Connexion au Serveur MQTT et Boucle Principale

Le client MQTT se connecte au serveur MQTT à l'adresse IP "192.168.1.115" sur le port 8883. La boucle principale (`loop_forever()`) maintient la connexion active.

## f. Énergie

Pour évaluer la consommation d'énergie de l'ESP pendant une partie intense, simulant une utilisation maximale, nous avons constaté que sur 10 minutes, l'ESP a consommé 12mAh. Avec une batterie de 2000mAh, cela offre une autonomie d'environ 27 heures et 46 minutes. Cela garantit une journée entière d'utilisation même en cas de batterie légèrement dégradée.



Après le jeu, en vérifiant les performances du système, la température du processeur est restée à un niveau acceptable (38.4°C). Bien que le serveur Minecraft ait montré des pics d'utilisation CPU, le système dans son ensemble reste stable et réactif.

En conclusion, le système fonctionne bien dans des conditions de jeu normales, offrant une expérience satisfaisante malgré quelques limitations liées à la charge du serveur Minecraft.

## IV. User Story 4

### a. Détection de victoire

Nous avons deux conditions à vérifier. La première consiste à briser seul le bloc de diamant. Nous pouvons créer un Scoreboard avec les datapacks de Minecraft pour compter le nombre

de blocs de diamant cassés. Par exemple, si le joueur Dracaufeu casse un bloc de diamant, son score augmente de 1.

```
scoreboard objectives remove diamondMined
scoreboard objectives add diamondMined
minecraft:mined:minecraft:diamond_block
```

Ainsi, si le score augmente de un, nous savons quelle équipe a gagné, et un message est affiché en conséquence.

```
execute if score @a[team=red,limit=1,gamemode=survival] diamondMined matches
1.. run scoreboard players set red win 1
execute if score @a[team=red,limit=1,gamemode=survival] diamondMined matches
1.. run title @a title {"text":"L'ÉQUIPE MAGMA A GAGNÉ
!!!","color":"gold","bold":true}
```

Pour déterminer si dix minutes se sont écoulées, nous utilisons un scoreboard "time" avec le joueur "t" qui augmente de 1 toutes les 10 secondes. Si le score atteint 60, l'autre équipe gagne.

```
execute if score t time matches 60 run scoreboard players set red win 1
execute if score t time matches ..59 run scoreboard players add t time 1
execute if score red win matches 1 run gamemode spectator
execute if score red win matches 0 run schedule function partie:windetect
10s
execute if score red win matches 1 run function partie:end

execute if score t time matches 60.. run title @a title {"text":"L'ÉQUIPE
AQUA A GAGNÉE !!!","color":"aqua","bold":true}
```

La commande "schedule" permet l'exécution d'une fonction avec un délai, et la commande "title" permet d'afficher du texte en gros plan sur l'écran des joueurs.

## b. Afficher la distance du bloc de diamant

Lors de l'apparition du bloc de diamant, nous récupérons ses coordonnées via le porte-armure et les stockons dans le scoreboard "coordinatesDiamond". Pendant la partie, nous récupérons les coordonnées du joueur que nous stockons dans un scoreboard "coordinatesBlue" (ou

"coordinatesRed") et effectuons l'opération "coordonnées du joueur - coordonnées du bloc de diamant". Ainsi, nous obtenons une valeur qui indique si nous nous rapprochons du bloc.

```
execute store result score x coordinatesBlue at
@a[gamemode=survival,team=blue] run data get entity
@a[gamemode=survival,team=blue,limit=1] Pos[0]
execute store result score y coordinatesBlue at
@a[gamemode=survival,team=blue] run data get entity
@a[gamemode=survival,team=blue,limit=1] Pos[1]
execute store result score z coordinatesBlue at
@a[gamemode=survival,team=blue] run data get entity
@a[gamemode=survival,team=blue,limit=1] Pos[2]
scoreboard players operation x compassAqua = x coordinatesDiamond
scoreboard players operation y compassAqua = y coordinatesDiamond
scoreboard players operation z compassAqua = z coordinatesDiamond
scoreboard players operation x compassAqua -= x coordinatesBlue
scoreboard players operation y compassAqua -= y coordinatesBlue
scoreboard players operation z compassAqua -= z coordinatesBlue
execute if score red win matches 0 run schedule function
partie:redstoneradarblue 1s
```

Nous affichons les scoreboards avec ces commandes :

```
scoreboard objectives setdisplay sidebar.team.blue compassAqua
scoreboard objectives setdisplay sidebar.team.red compassMagma
```

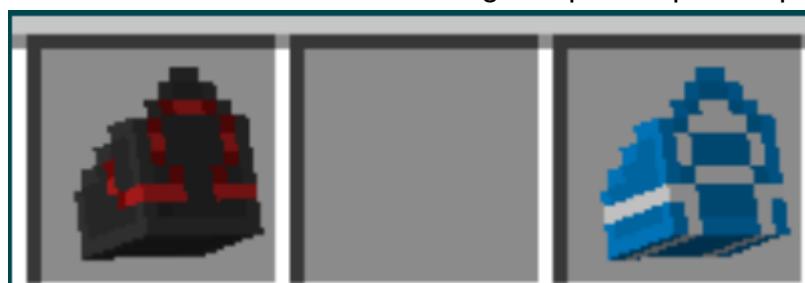
Pour les afficher uniquement pour chaque équipe.

## V. Bonus

### a. Texture Pack

Nous avons créé un pack de textures qui est un fichier à ajouter au client Minecraft. Pour l'installer, il suffit de le télécharger, puis dans les options du jeu, ouvrir le dossier des packs de textures et glisser le fichier dans ce dossier. Celui-ci modifie les sons et l'apparence du jeu.

Nous avons créé des modèles modifiés en forme de gants pour l'épée en pierre.



Les deux gants ont les mêmes propriétés qu'une épée en pierre, seule leur apparence change.

Nous avons également modifié les musiques de Minecraft en nous inspirant de l'histoire d'un jeu Pokémon. Ainsi, nous avons remplacé certaines musiques de Minecraft par celles de Pokémon, comme illustré par les commandes suivantes :

```
playsound music.nether.nether_wastes master @a  
playsound music.overworld.badlands master @a
```

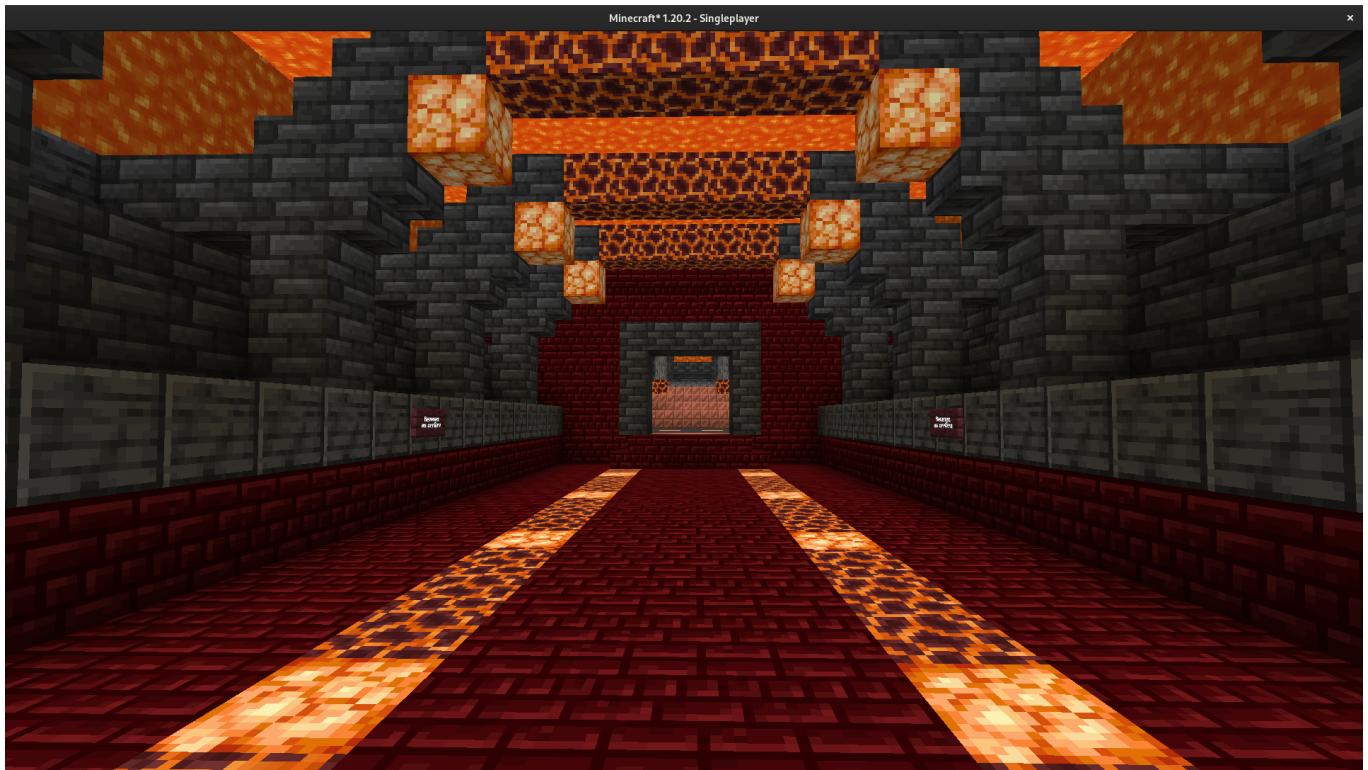
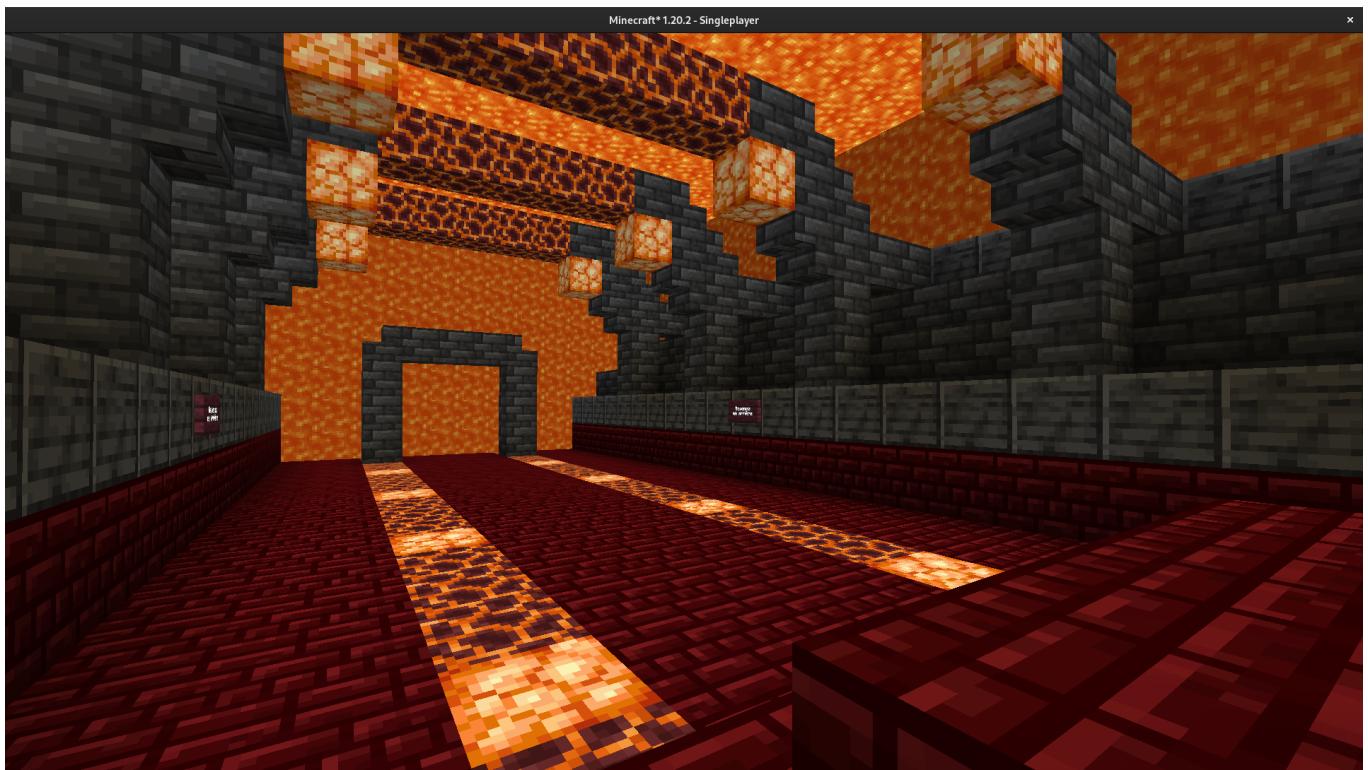
Une pour l'introduction et l'autre pendant la partie.

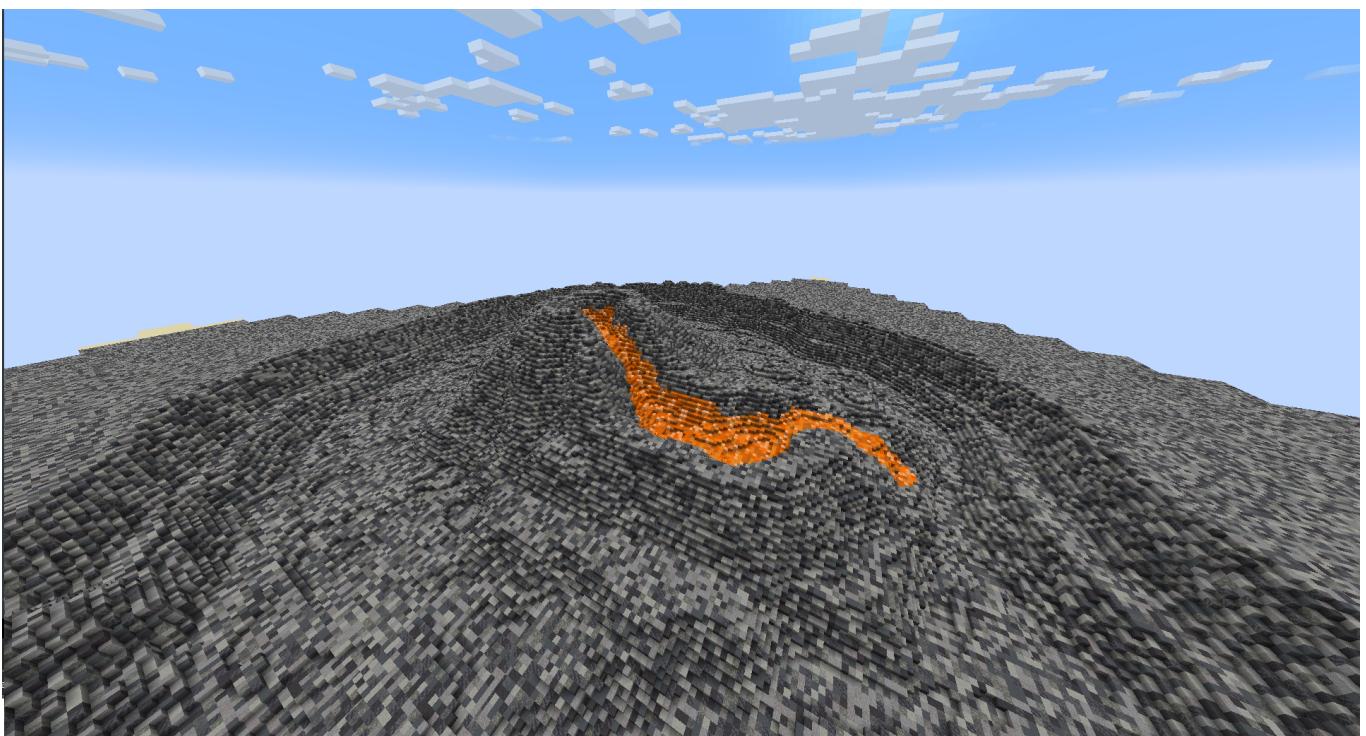
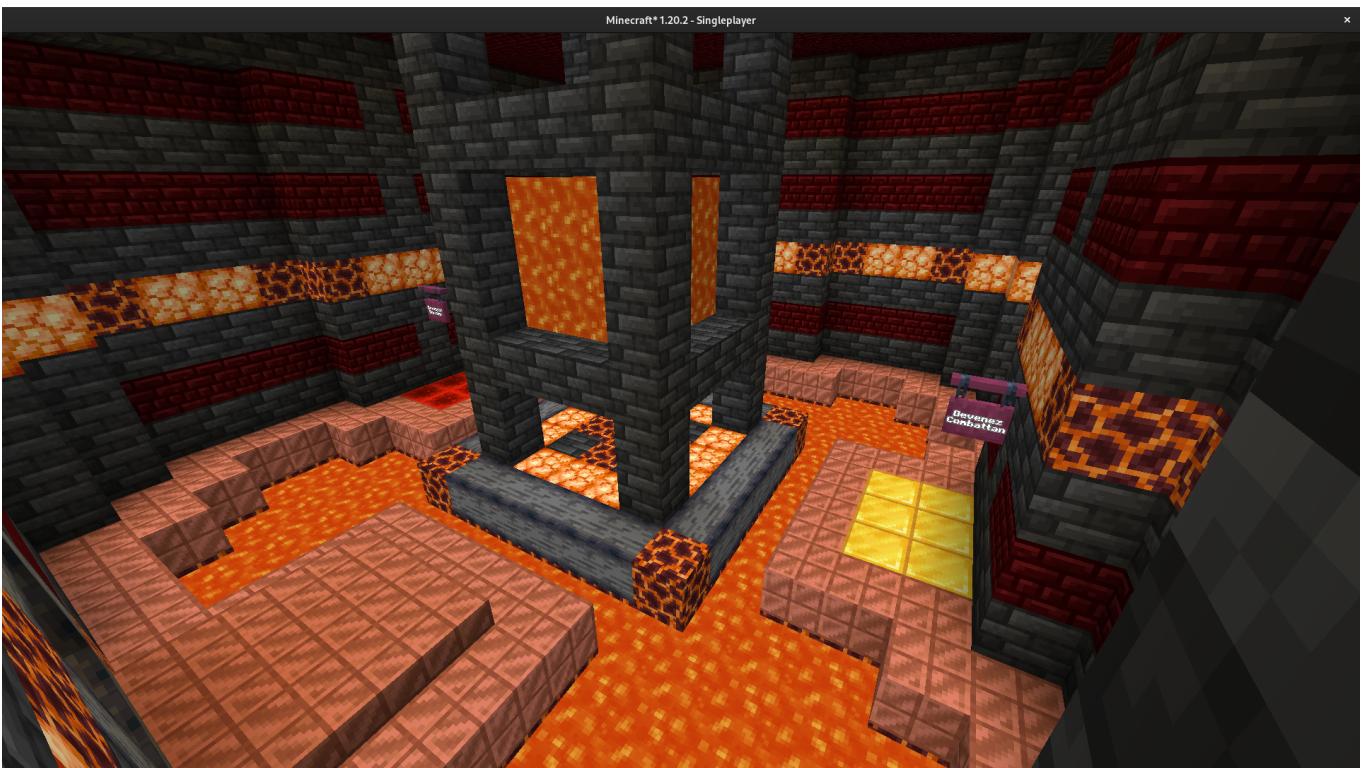
## b. Map

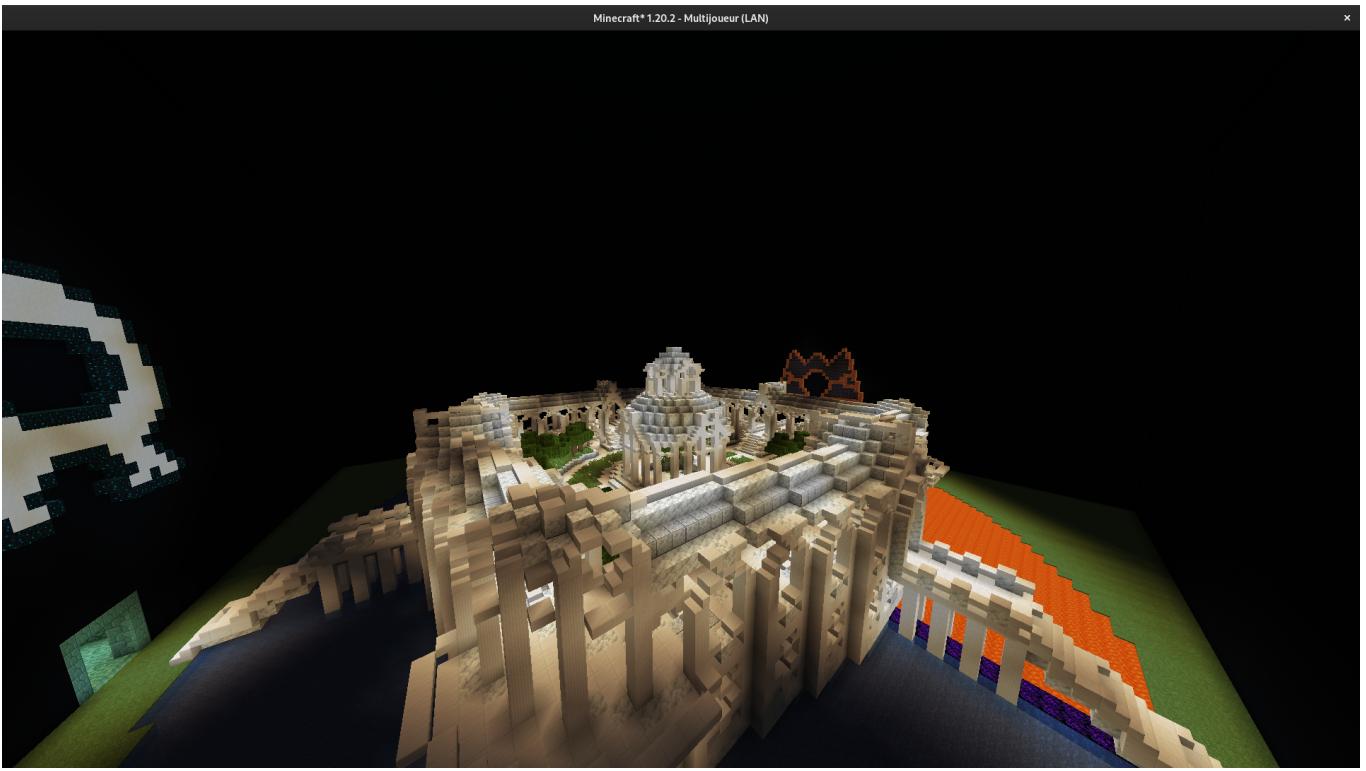
Le monde que nous avons utilisé pour la SAE a été entièrement créé par nous.











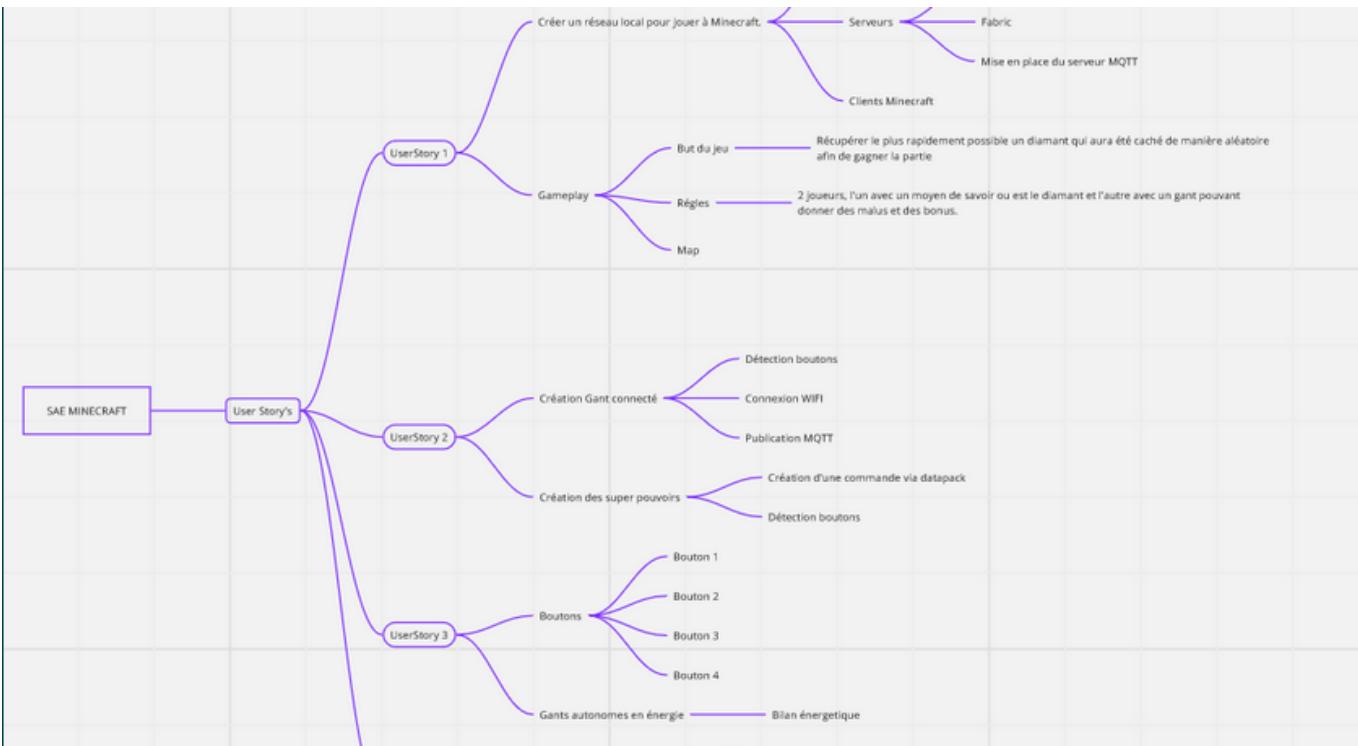
Nous avons également ajouté une détection automatique via des zones pour choisir les rôles et l'équipe. À chaque début de partie, une cinématique présente les règles et les objectifs de chaque équipe.

### c. Outil

Pour cette SAE, nous avons utilisé Visual Studio Code pour développer les datapacks avec le site [MCStacker](#) qui peut aider pour toutes les commandes, sauf les scoreboards et la partie "execute". Pour construire la map, nous avons utilisé WorldEdit, disponible à [cette adresse](#), qui est très utile pour les constructions de grandes envergures. Pour les modèles 3D des gants, nous avons utilisé BlockBench, accessible à [ce lien](#).

## VI. Gestion de Projet

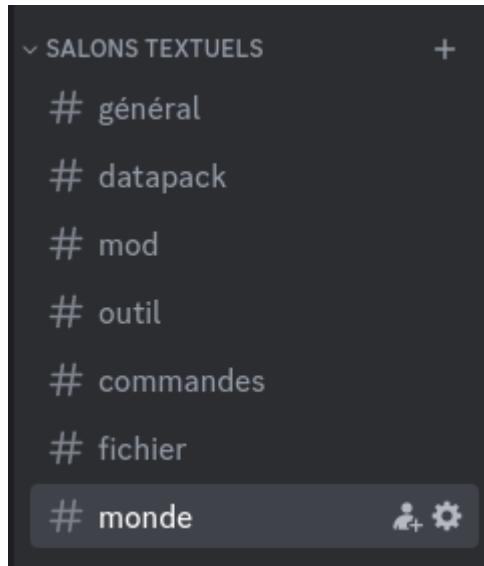
Nous avons initialement utilisé une carte mentale pour définir les besoins de la SAE.



Par la suite, nous avons organisé les tâches sur un tableau Trello.

Colonne	Tâches
Fait	Map Magma, Publication MQTT, Mise en place du serveur MQTT, Détection boutons, Créer un réseau local pour jouer à Minecraft, Fabric, Installation Java, Détection boutons dans le jeu
Sprint En Cours	Data Pack
En Cours	Bouton 1 Avec pouvoir, Bouton 2 Avec pouvoir, Bouton 3 Avec pouvoir, Bouton 4 Avec pouvoir, Construction du Gant, Cinématique de début, Rapport
En Attente	Map Aqua, Bilan énergétique, Cinématique de fin, Boussole, Sécurité de MQTT, Oral, Diaporama, Vérification du rapport
Boîte à idée	Musique pe, + Ajouter une carte

Nous avons mis en place un serveur Discord pour faciliter la communication, fixer des rendez-vous et organiser des réunions, ainsi que pour partager des fichiers. Il est réparti en plusieurs salons :



Nous avons veillé à faire travailler l'ensemble de l'équipe sur toutes les tâches afin d'assurer la maîtrise de tous les outils. Nous avons évité de nous spécialiser dans une seule tâche, favorisant le travail en groupe ou en binôme.

## VII. Conclusion

La réalisation de cette SAE a été une expérience enrichissante et stimulante. Nous avons pu mettre en œuvre nos compétences en programmation, en conception de jeux et en gestion de projet pour créer un mini-jeu Minecraft innovant et divertissant. L'utilisation de plusieurs technologies, telles que les datapacks, les scripts Python, MQTT, et la configuration d'un serveur Minecraft, a permis de créer une expérience de jeu unique.

La collaboration étroite entre les membres de l'équipe, la communication efficace via Discord, et l'utilisation d'outils tels que Trello ont facilité la coordination des tâches et la gestion du projet. La conception de la map, la création du datapack, l'implémentation des fonctionnalités du gant connecté, la gestion de l'énergie de l'ESP, et la détection de la victoire ont été des aspects essentiels du projet.

L'ajout de bonus tels que le pack de textures personnalisé et la création d'une map originale ont contribué à rendre le jeu plus immersif et divertissant. La gestion de l'énergie de l'ESP a également été abordée de manière réaliste, avec des mesures prises pour optimiser la consommation d'énergie et assurer une autonomie suffisante.

En fin de compte, la SAE a abouti à la création d'un jeu original et à une expérience complète qui a mis en avant notre créativité, nos compétences techniques et notre capacité à prévenir les problèmes. Nous avons également acquis une expérience précieuse dans la gestion de projet collaboratif et la réalisation de projets complexes.