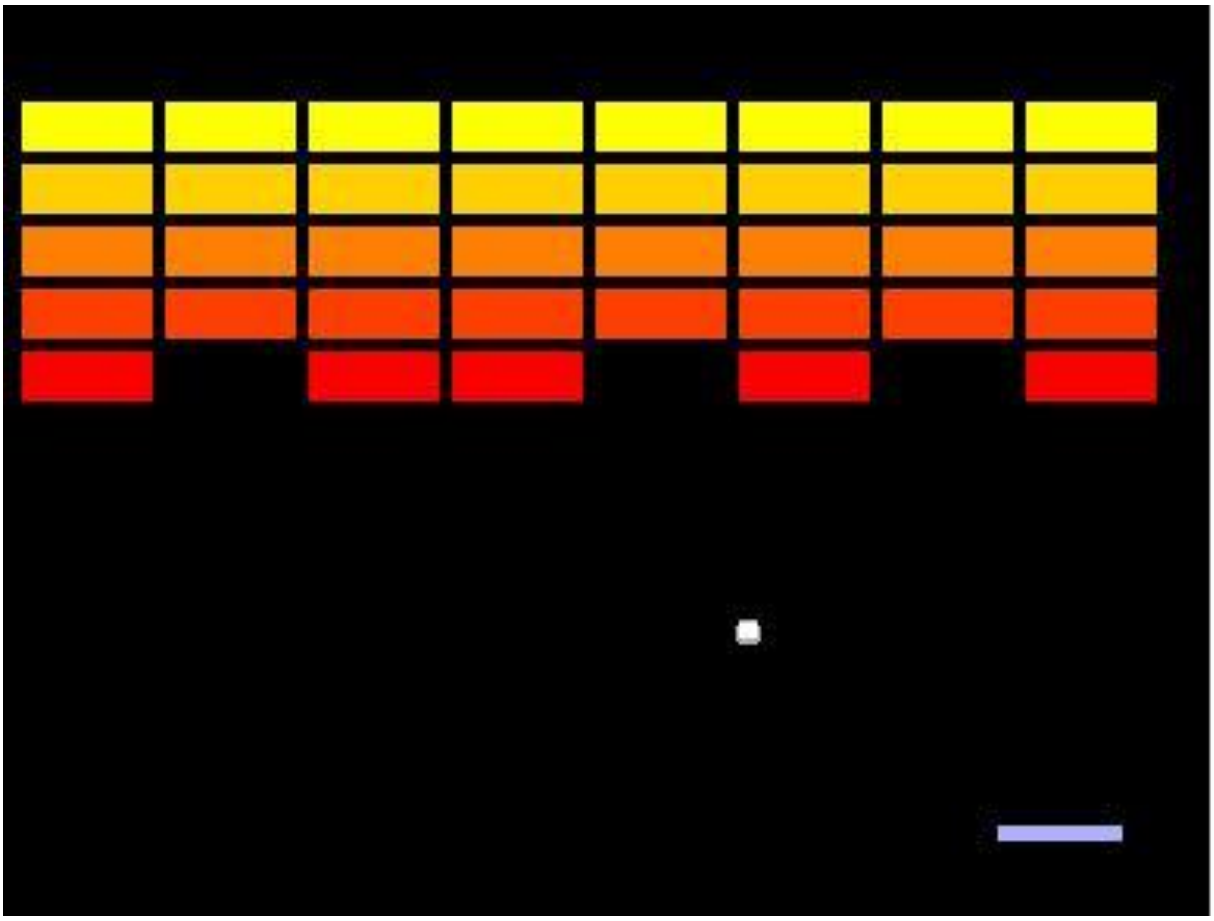
INSA TOULOUSE

# Compte rendu du BE C++

## Jeu Casse-Cou



BOUKEZZATA Aymen

BERTON Thomas

MENECEUR Leila

Année scolaire 2021

# Introduction :

Dans le cadre de l'enseignement de C++, nous avons été amenés à réfléchir à une conception orientée objet. L'objectif était de mettre au point une application attractive et innovante, développée avec le langage C++, et utilisant une bibliothèque modulaire, extensible pour s'interfacer avec des capteurs et des actionneurs. L'ensemble de cette invention devait faire partie du monde des objets connectés et de l'Internet des objets.

Nous avons donc décidé de développer un jeu inspiré du casse-briques, mais que l'utilisateur pourra le contrôler avec le geste d'une main. Il s'agira dans ce rapport d'expliquer les différentes étapes de conception du produit et d'en faire la démonstration.

## 1. Principe de fonctionnement du jeu

Un casse-brique est un type de jeu où le joueur contrôle une raquette et doit détruire des briques à partir d'une balle pour passer au niveau suivant. Étant passionné par les jeux-vidéos, nous voulions combiner un jeu ancien avec les technologies actuelles. C'est pour cette raison que nous avons voulu faire un casse-brique dont la raquette soit contrôlable à partir d'un gant connecté.

Nous avions à disposition une carte ESP8266 qui contient un module WIFI. Le jeu est codé principalement en C++ sur le quelle on traite tous les calculs et applicatifs du jeu :

- L'initialisation du jeu (l'Arena, la barre, la balle et les briques)
- Le calcul de déplacement pour l'animation de la balle et la barre
- La gestion des collisions entre la balle, la barre et les murs
- Le calcul du score et l'interruption du GameOver

La gestion de la barre est contrôlée par le mouvement d'un accéléromètre (MMA7660) externe connecté au microcontrôleur ESP82. L'accéléromètre est de 3 axes x, y, z et renvoie l'accélération des 3 axes d'une valeur entre $\pm 1.5g$. Nous avons choisi $\vec{y}$ comme axe de référence pour le mouvement de la barre. Une bibliothèque déjà codé en C++ qui permet renvoyer ces valeurs avec la fonction

```
bool MMA7660 :: getAcceleration(float *ax,float *ay,float *az); (ESP86_accelerometer.h)
```

Concernant l'affichage du jeu nous avions eu deux choix possibles :
- Gérer l'affichage avec un écran à partir de sa bibliothèque
- Gérer l'affichage sur une page web à partir de requête HTML

Nous avons opté pour l'option de la page web car celle-ci nous permettait d'avoir un jeu qui puisse être affiché sur un plus grand écran qu'un écran LED arduino.

En effet, c'est à partir d'un navigateur WEB capable d'exécuter du HTML/JavaScript/CS qui servira d'interface graphique pour animer le jeu.

Ensuite,Il faut être capable de pouvoir faire communiquer les deux instances à savoir la carte et la page HMTL.

Pour cela nous avons profité du module WIFI pour créer un serveur WEB avec les bibliothèques <ESP8266WiFi.h> <ESP8266WebServer.h> (ESP86_server.h). La communication par défaut entre le client/serveur se fait par la méthode « Post/Request http » mais entraine un

temps de latence très important pour un jeu (de l'ordre de 1-2 secondes même en dynamique). Nous avons donc opté donc pour une communication en temps réel avec le $WebScocket$ qui offre, une fois que le client est connecté au serveur, une connexion TCP/IP pour le transfert des données.
On a utilisé la bibliothèque $< WebSocketsServer.h >$ pour faire ce mode de connexion. Dans un premier temps il faut ouvrir un nouveau port (on a choisi le port 81), ensuite on initialise le WebSocket et on lance un évènement pour récupérer les données.

```
WebSocketsServer webSocket =
WebSocketsServer(81); // ouvrir le
port 81

webSocket.begin(); //initialiser
le websocket
webSocket.onEvent(webSocketEvent);
// lancer l'événement

void webSocketEvent(uint8_t num,
WStype_t type, uint8_t * payload,
size_t length);//l'évènement

webSocket.loop(); // loop pour
l'évènement
```

***Coté Serveur (C++)***
***(ESP86_webSocket.h)***

```
Socket = new WebSocket("ws://" +
window.location.hostname + ":81/");
//on se connecte au serveur

Socket.onopen() // L'évènement
lorsque le client est connecté

Socket.onopen().onmessage() //
L'évènement pour récupérer le
message du serveur
```

***Coté Client (JavaScript)***
***(game_html.h)***

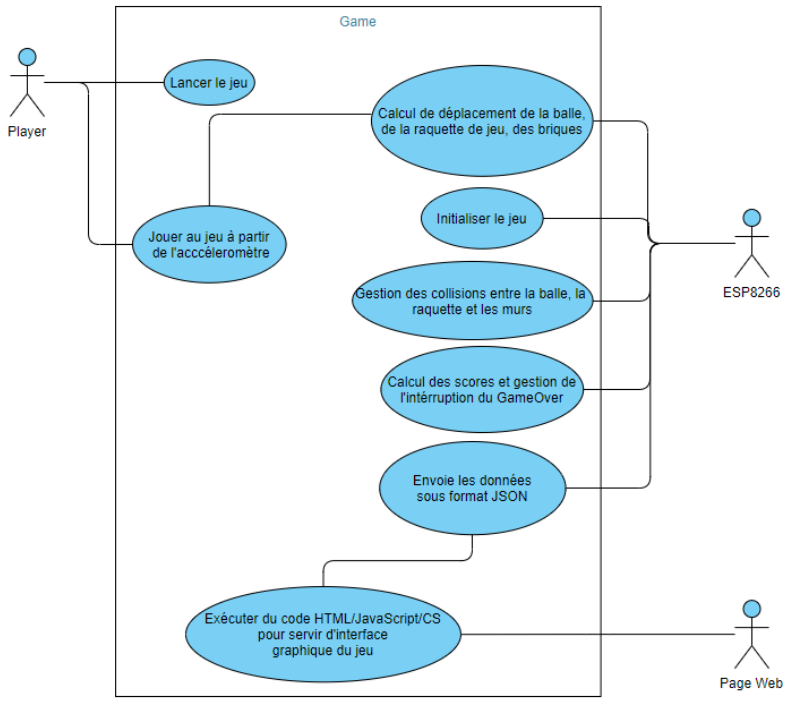
Vu que les deux codes sont différent (C++ et Javascript), on a décidé que le format du flux de données le plus adéquate serait sous format JSON (JavaScript Object Notation). Cela permet de générer un objet de données, qui va le mettre en $String$ en C++, puis l'envoyer au client. Ensuite le JavaScript reçoit le JSON pour l'analyser et récupérer les variables envoyées sous forme de clés $(String)$ et de valeurs $(int, float, boolean, object, array)$. Les bibliothèques utilisées en C++ pour générer du JSON sont $'$rapidjson/writer.h' et 'rapidjson/stringbuffer.h$'dans\ (JSON.h)$.

Pour chaque élément du jeu nous avons créé une propre classe qui contient les variables et méthodes spécifier $'ball.h''barre.h''Area.h''briques.h''component.h'$.

Enfin dans $'init.h'$ on trouve la classe qui initialise tous les composants du jeu, leurs méthodes de calcul et gère l'envoie des data au moniteur (PC)

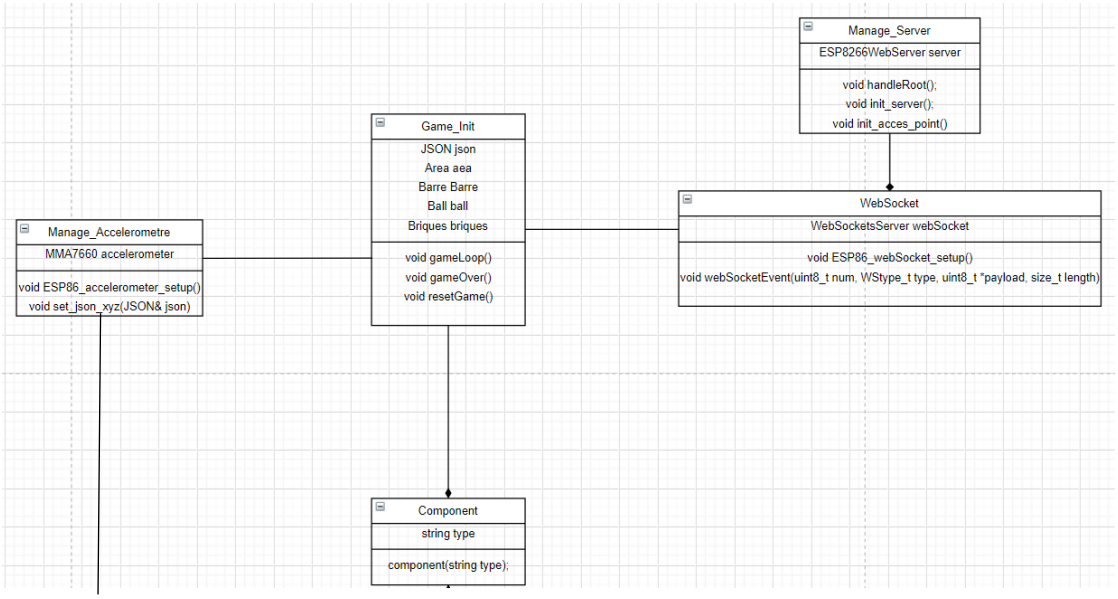
## 2. Use case diagram:

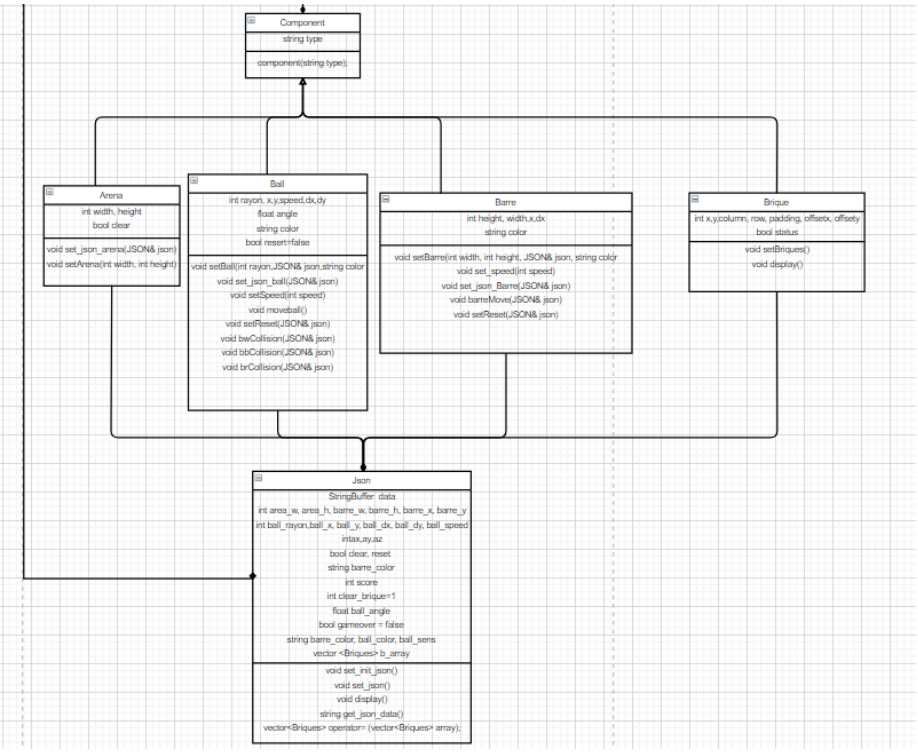
Notre diagramme des cas d'utilisation est divisé en trois acteurs. Il y a le joueur, qui doit contrôler l'accéléromètre à partir de sa main pour jouer au jeu. Ensuite il y a la carte ESP82 qui permet de faire tourner le jeu en faisant tourner tout le calcul et l'applicatif. Et enfin la page WEB qui permet d'afficher le jeu en exécutant du code Javascript/HTML. Chaque acteur a ses propres fonctions.




## 3. Diagramme de Classe :

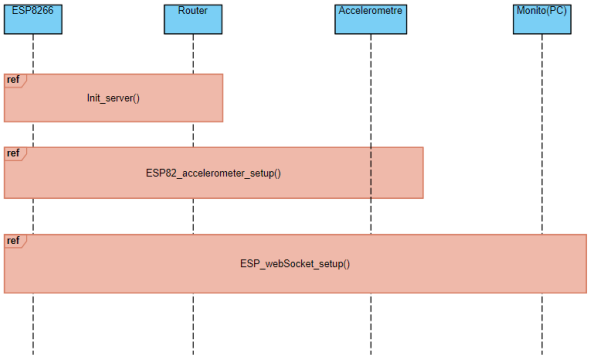
La classe Game_init est en quelque sorte notre classe « principale », c'est cette classe qui va faire appel à toutes les classes que nous allons vous présenter ici.
Il y a tout d'abord il y a les classes Manage_Server et le WebSocket qui permettent d'assurer la communication entre le serveur (notre carte ESP8266) et le client (le navigateur WEB) .Ensuite il y a la classe Manage_Accelerometre qui permet de récupérer les valeurs de l'accéléromètre pour faire le calcul du mouvement de la barre.
Enfin cette classe a une relation d'agrégation avec la classe Component, qui est la classe mère assurant toutes les fonctionnalités du jeu et qui sera décrite juste en dessous.

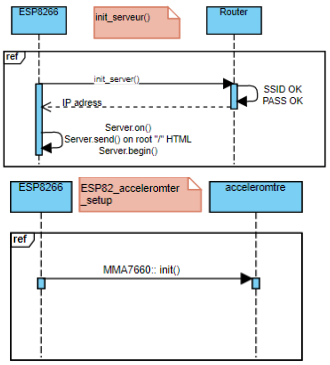


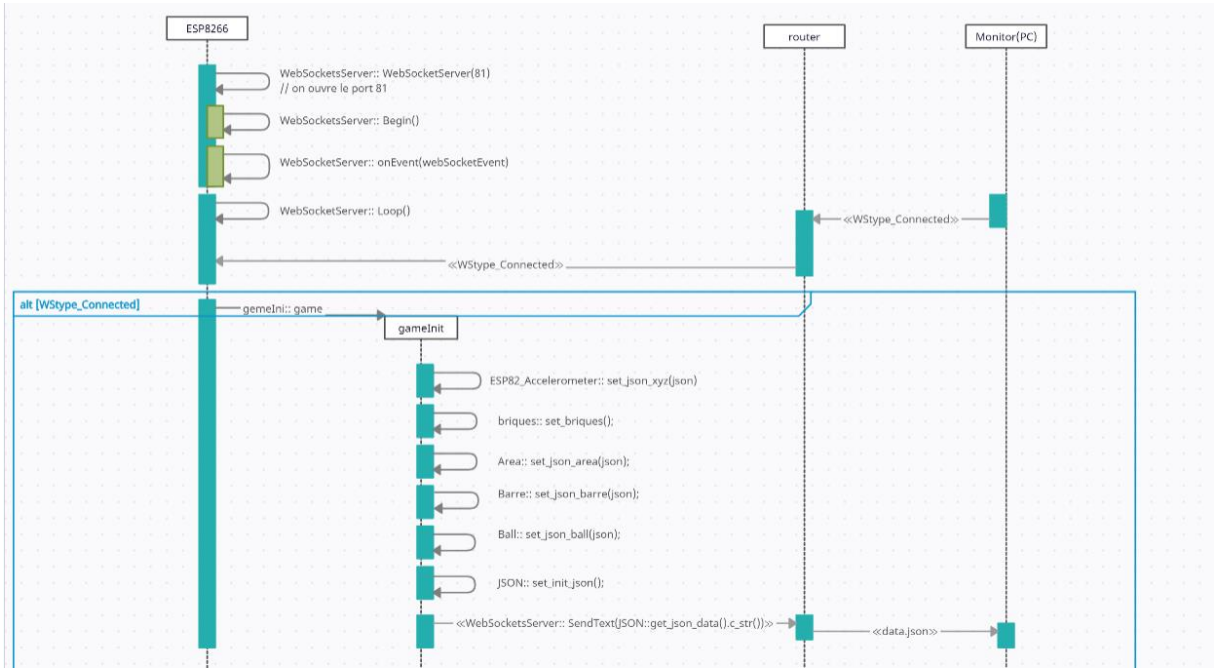


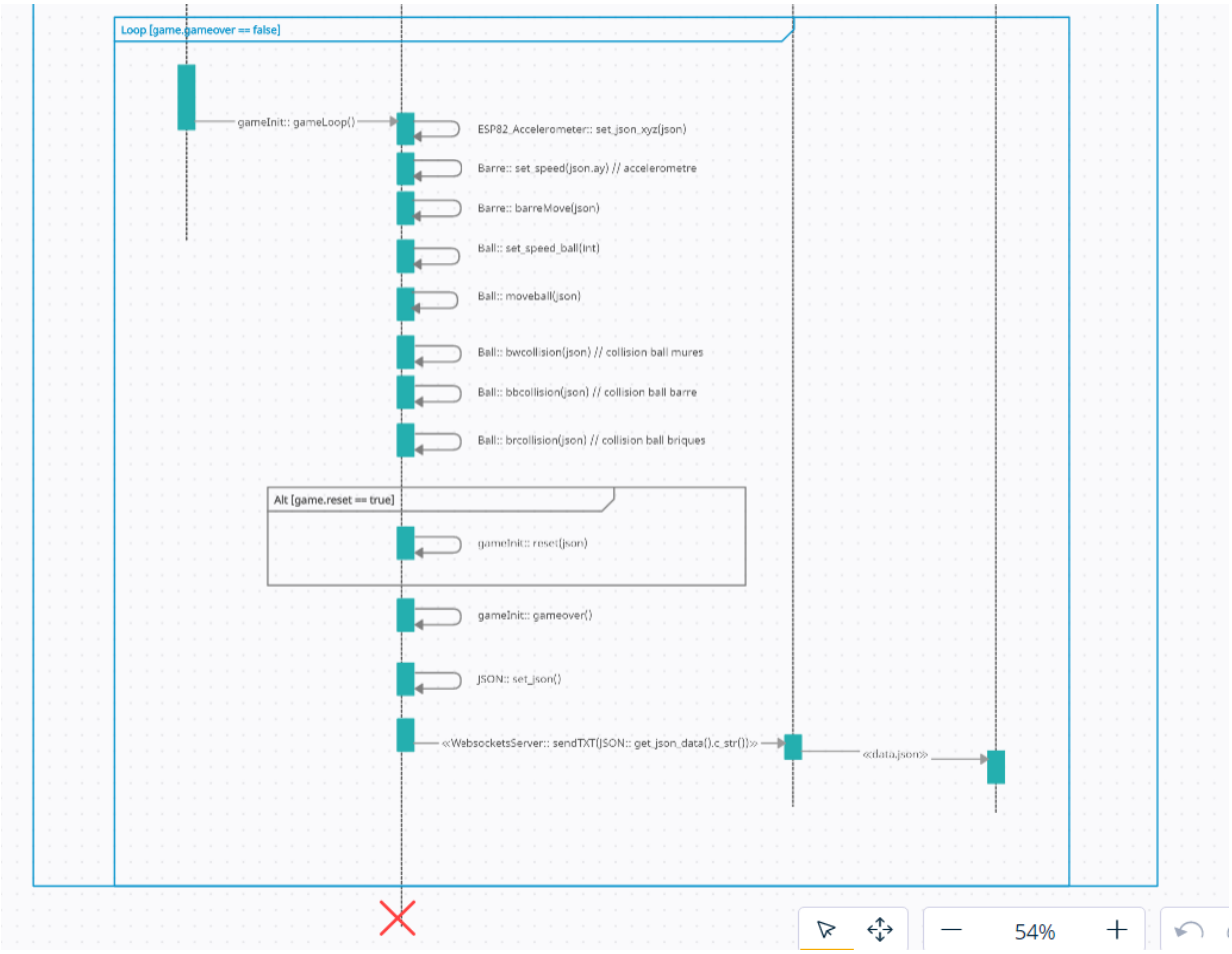

La classe Component est la classe mère des classes Arena, Ball, Barre et Brique. Chacune de ces classes permet des gérer les différents paramètres qui est la base d'un jeu casse brique à savoir la balle, les différentes briques et la raquette contrôlé par le joueur. L'Arena fait référence à la taille de l'espace jouable du jeu.
Enfin chacune de ces classes a une relation d'agrégation forte avec la classe JSON. En effet, chacune de ces classes doit créer son propre JSON afin d'envoyer les données qu'elle vient de calculer au serveur. Il est donc obligatoire que chaque classe des fonctionnalités du jeu soit reliée à la classe JSON pour communiquer avec le client.

## 4. Diagramme de séquence




ESP8266
router
Monitor(PC)
WebSocketsServer:: WebSocketServer(81)
// on ouvre le port 81
WebSocketsServer:: Begin()
WebSocketServer:: onEvent(webSocketEvent)
WebSocketServer:: Loop()
«WStype_Connected»
«WStype_Connected»
alt [WStype_Connected]
gameIni:: game
gameInit
ESP82_Accelerometer:: set_json_xyz(json)
briques:: set_briques();
Area:: set_json_area(json);
Barre:: set_json_barre(json);
Ball:: set_json_ball(json);
JSON:: set_init_json();
«WebSocketsServer:: SendText(JSON::get_json_data().c_str())»
«data.json»


Loop [game.gameover == false]
gameInit:: gameLoop()
ESP82_Accelerometer:: set_json_xyz(json)
Barre:: set_speed(json.ay) // accelerometre
Barre:: barreMove(json)
Ball:: set_speed_ball(int)
Ball:: moveball(json)
Ball:: bwcollision(json) // collision ball mures
Ball:: bbcollision(json) // collision ball barre
Ball:: brcollision(json) // collision ball briques
Alt [game.reset == true]
gameInit:: reset(json)
gameInit:: gameover()
JSON:: set_json()
«WebsocketsServer:: sendTXT(JSON:: get_json_data().c_str())»
«data.json»
54%

## Notre conclusion (difficultés rencontrés, améliorations…)

Malgré la situation sanitaire et des séances qui se sont enchainés assez rapidement, nous sommes très fières de ce que nous avons pu accomplir. De manière générale, le jeu marche très bien.
Le codage du jeu en C++ et le codage de la page WEB en html/javascript n'étaient pas si compliqués que ça finalement. Mais c'est surtout la communication entre le serveur et le client qui était la tâche la plus complexe. Lorsque nous avons décidé de choisir une page WEB pour l'affichage du jeu, nous n'étions pas au courant à quel point cela pouvait être compliqué de gérer une communication client/serveur entre deux instances qui communiquent avec un langage différent. Mais cela nous a permis de découvrir et d'apprendre à utiliser la bibliothèque JSON. Enfin, le jeu pourrait un peu plus propre et synthétique au niveau du code, il est très chargé et assez compliqué à prendre en main.