Introduction

L'objectif de ce projet est de concevoir un jeu embarqué inspiré du Dino Run de Google, implémenté en C++ sur une carte ESP8266. Le projet vise à appliquer les notions de programmation orientée objet, de gestion du temps réel et d'interaction matériel-logiciel dans un environnement contraint. Nous avons choisi de faire un dino runner, qui reprend les codes du jeu très connu sur Chrome. Pour cela, il nous a fallu adapter l'affichage et les mécaniques du jeu avec les capteurs / actionneurs à notre disposition. Tout cela, en ajoutant certaines features qui nous semblaient pertinentes afin de mettre en œuvre des concepts de POO.

I. Environnement matériel du projet

Comme évoqué précédemment, notre logiciel est embarqué sur une carte Arduino ESP8266. Un certain nombre de capteurs et actionneurs étaient à notre disposition. L'actionneur principal de notre application est l'écran LCD qui permet l'affichage du jeu et de certaines explications lorsque nécessaires. Cet écran fonctionne par communication I2C avec la carte arduino, ces pins du GPIO associés étaient donc choisis par définition. Les autres actionneurs de notre environnement matériel sont des leds et un buzzer. En ce qui concerne les capteurs, le logiciel utilise un potentiomètre et des boutons. Ce sont des boutons led qui sont néanmoins séparés dans le code comme des boutons et des leds usuels, associés à un pin respectif du GPIO. Le mapping de la carte est le suivant :

- LCD : PIN D1 e tD2
- Led1 : Pin D5
- Led2 : Pin D0
- Buzzer : Pin D8
- Bouton rouge : D6
- Bouton orange : D3
- Potentiomètre : A0

II. Fonctionnement global du jeu

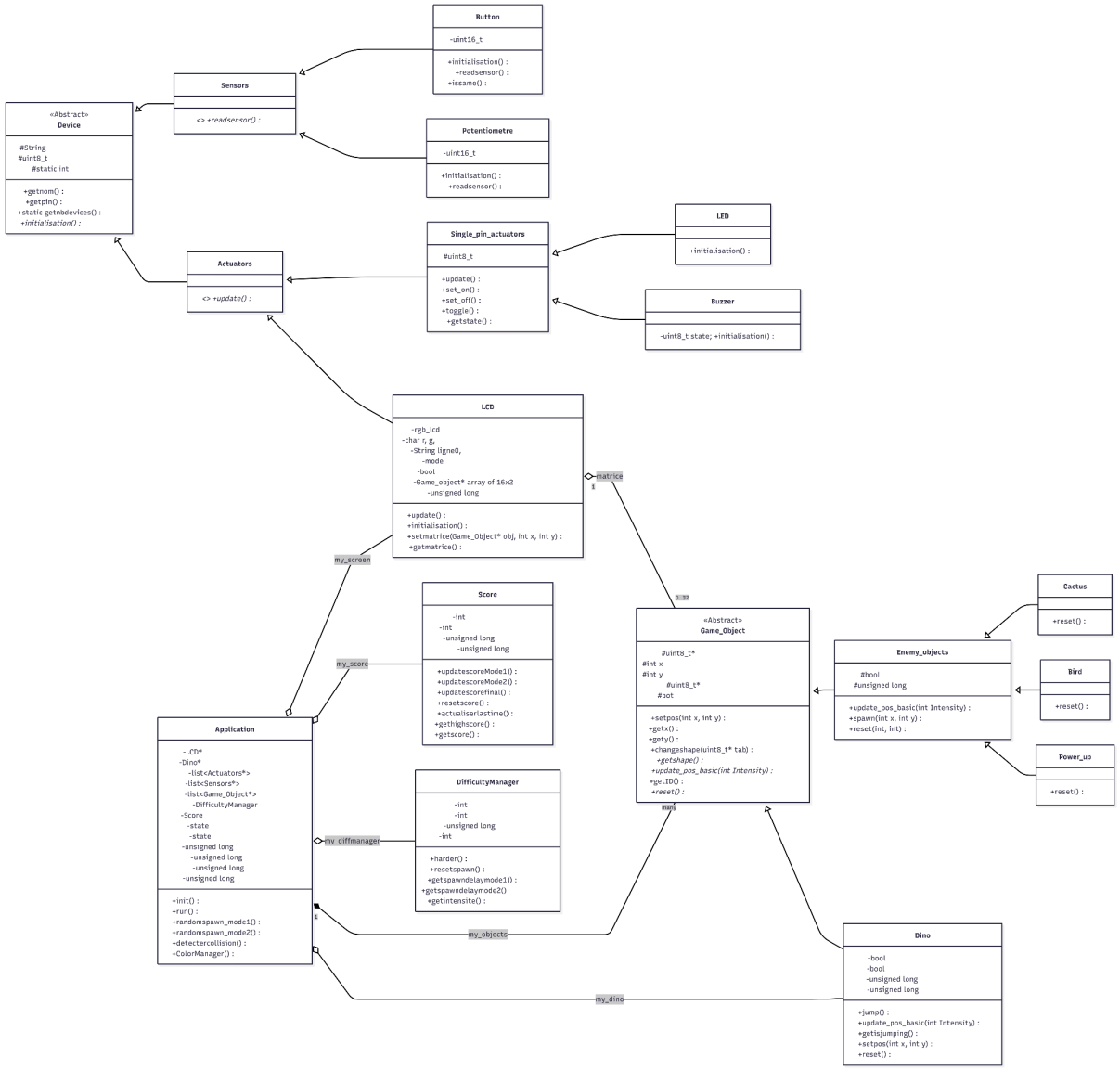
L'objectif du jeu est de faire survivre le dinosaure le plus longtemps possible en évitant les obstacles générés de manière continue. Le joueur contrôle le dinosaure à l'aide de boutons lui permettant de sauter ou de se baisser, afin d'esquiver des cactus et des oiseaux apparaissant à l'écran. Le score, correspondant au temps de survie, est affiché en continu en haut à gauche de l'écran.

Le jeu fonctionne selon une boucle principale qui gère successivement la lecture des entrées utilisateur, la mise à jour des éléments du jeu et l'affichage. À chaque itération, la position du dinosaure et des obstacles est recalculée. Une détection de collision est effectuée : si le dinosaure entre en contact avec un cactus ou un oiseau, la partie se termine et un état Game Over est déclenché.

Un mode spécial est introduit via un power-up, inspiré des mécaniques de jeux comme Mario. Lorsqu'il est activé, le gameplay change temporairement pour se rapprocher de celui de Geometry Dash. Dans ce mode, le dinosaure peut rester dans les airs en appuyant une fois sur la commande « haut », et redescendre en utilisant la commande « bas ». Ce mode est limité par un compte à rebours de 10 secondes affiché, au terme duquel le jeu revient automatiquement au mode de fonctionnement normal. Aussi, un potentiomètre permet à l'utilisateur de régler la couleur d'affichage du LCD à sa préférence.

Afin de renforcer la variété visuelle, un mode nuit est activé de manière périodique, modifiant l'apparence du jeu sans en affecter les mécaniques. Par ailleurs, la difficulté augmente progressivement avec le temps : la vitesse de déplacement des obstacles ainsi que leur fréquence d'apparition sont accrues, rendant la survie de plus en plus complexe au fil de la partie.

III. Architecture logiciel



Conclusion

Ce projet a permis de développer avec succès un jeu de type Dino Run en C++ sur une plateforme embarquée ESP8266. Le jeu est pleinement fonctionnel et intègre l'ensemble des mécaniques prévues : gestion des obstacles, détection des collisions, affichage continu du score, modes de jeu alternatifs et augmentation progressive de la difficulté. Le comportement du système est stable et offre une expérience de jeu cohérente malgré les contraintes du matériel embarqué.

Néanmoins, certaines limitations matérielles ont été observées. Le temps de rafraîchissement et la résolution de l'écran LCD peuvent rendre le jeu difficile à suivre lorsque la vitesse des éléments augmente, provoquant parfois des artefacts visuels, comme l'impression d'apparition temporaire de plusieurs obstacles à l'écran. De plus, la mémoire limitée de l'ESP8266 restreint l'ajout de nouveaux éléments ou de modes de jeu plus complexes, pouvant entraîner une saturation des ressources et empêcher une exécution correcte du programme.

Malgré ces contraintes, ce projet illustre efficacement les enjeux et compromis du développement logiciel en environnement embarqué.