Juan SANCHEZ MEA4

ICOBS Game Project

Pour aller plus loin dans la découverte de l'architecture ICOBS, nous devons implémenter un jeux vidéo contenant au moins 2 sprites, et dans le quel nous serons capables de déplacer librement ces sprites dans l'écran tout en gérant les collisions entre les mêmes.

D'un point de vue hardware, nous devons réutiliser l'architecture ICOBS utilisé jusqu'ici pour l'adapter à nos besoins. Du cote software, nous allons construire l'intégralité du jeu, c'est-à-dire, les interactions entres les sprites, le niveau de difficulté, la gestion des points, l'arrêt du jeu, etc.

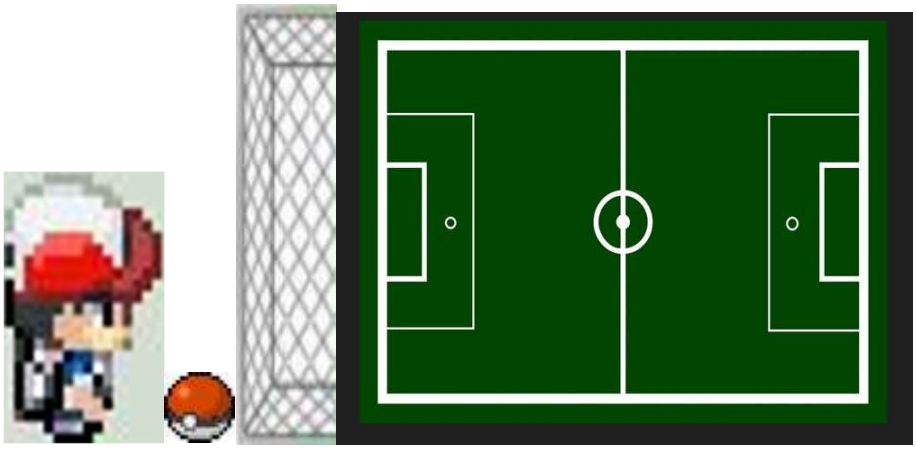
Dans ce cadre-là, j'ai eu l'idée de faire un jeu à 2 joueurs que j'ai décidé d'appeler pokefoot, basé sur l'univers Pokémon, et où pour gagner le joueur doit défendre son but tout en essayant de marquer dans le but de l'adversaire.

Pour y arriver, dans un premier temps j'ai ajouté un total de 4 sprites différents, qui me permettent de représenter les joueurs, la balle la cage, et finalement le terrain de football. Puis, j'ai réalisé les modifications nécessaires au niveau hardware qui m'ont permis d'intégrer et gérer chaque sprite de manière différente avec ses propres particularités. J'ai ensuite créé des registres indépendants en X et en Y pour chacun des sprites pour pouvoir les déplacer librement sur la map.

Et pour finir, j'ai implémenté le code nécessaire pour assigner un sprite à 2 boutons pour chaque joueur. Puis je me suis concentré sur l'implémentation de la gestion de collision entre les sprites. Ensuite, j'ai codé en c une partie me permettant de savoir si la balle est rentrée dans l'une des cages, et en fonction du joueur qui marque je donne un point à l'un ou à l'autre et j'arrête momentanément le jeu jusqu'à ce que le joueur de gauche appuie sur le bouton up (reset). Et finalement, j'ai mis en place l'affichage des points de chaque jouer sur le 7 segments.

Ainsi, j'ai combiné des fonctionnalités matérielles et logicielles pour créer un environnement de jeu dynamique où deux joueurs peuvent interagir et défier l'adversaire. Ce jeu est conçu pour offrir une expérience agréable tout en exploitant l'architecture ICOBS pour une gestion fluide des sprites et des interactions.

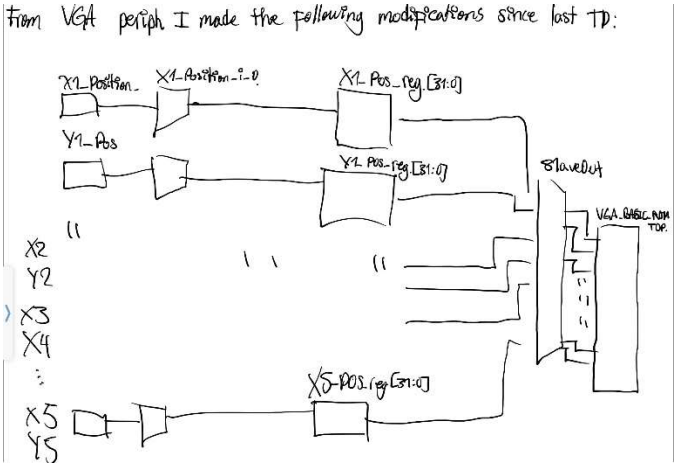
Voici les quatre images utilisées pour représenter les éléments de mon jeu :



Juan SANCHEZ MEA4

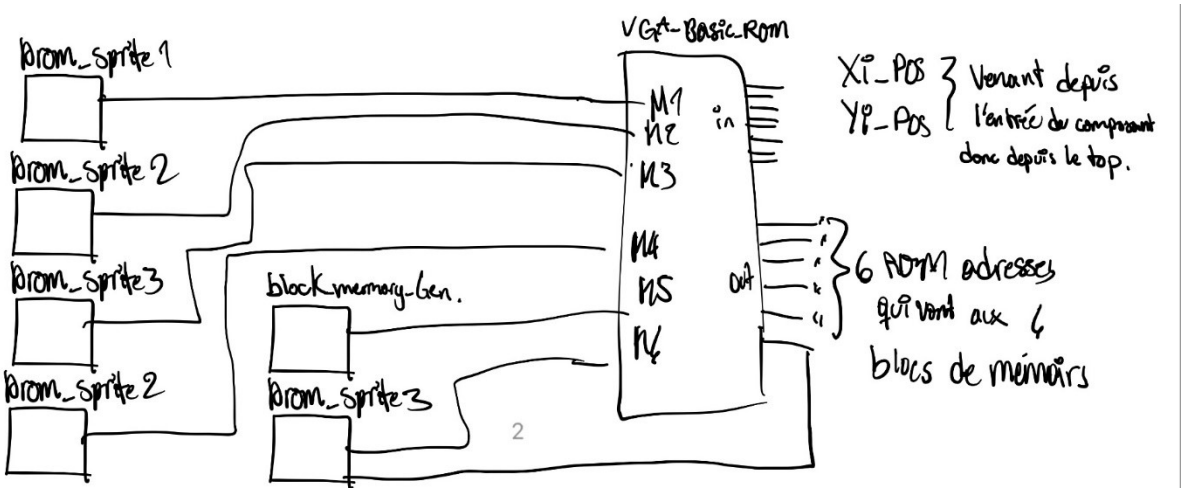
Hardware architecture

Depuis le dernier TD, j'ai rajouté 2 registres pour chaque image rajoutée dans le jeu sauf pour le terrain de foot qui n'a pas besoin d'être bougé dans le jeu, et j'ai donc pas la nécessité de X6pos Y6pos pour cette image.



Puis dans basic top, nous rajoutons un sprite par image. Les blocs de mémoire marchent de manière conventionnelle, il prend un rom adresse en entrée et il renvoie une data en sortie qui est donc Mi.

Ensuite dans VGA_Basic_ROM nous affichons pixel par pixel grâce aux Xi_pos Yi_Pos.

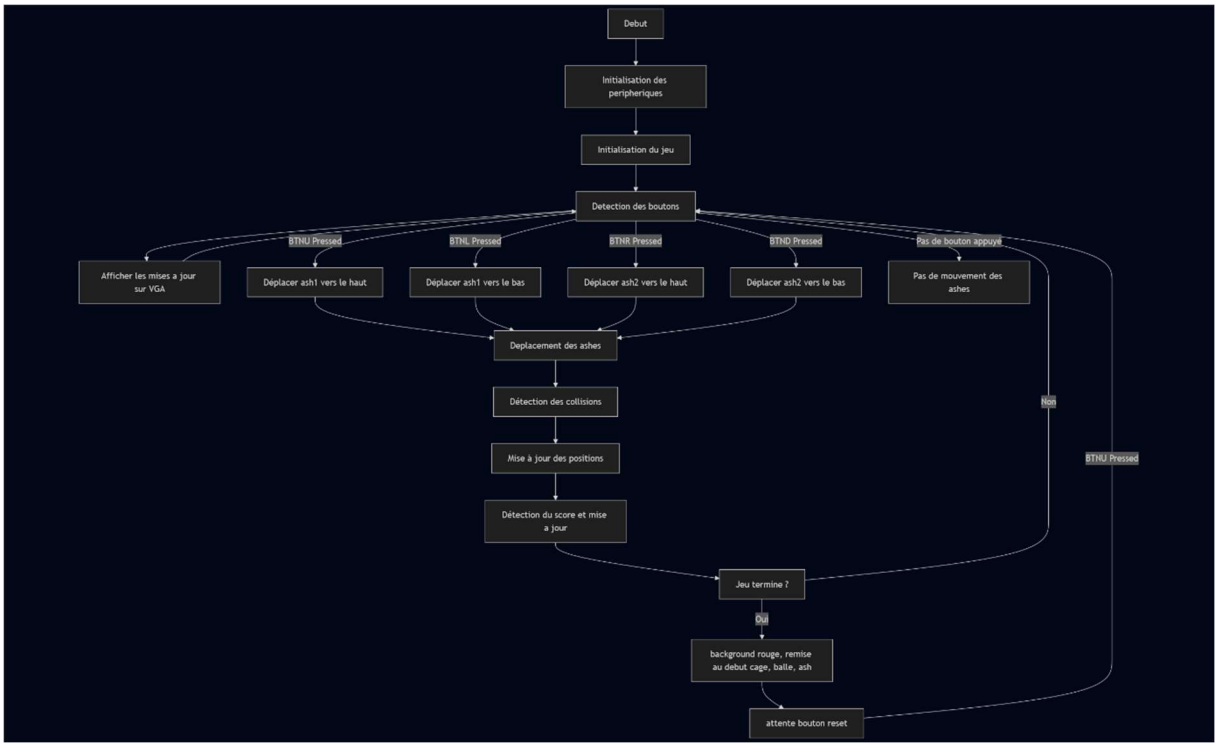


Si jamais deux images se superposent pour une partie de l'écran, dans mon cas c'est la première image qui restera affichée :

```
129     if vidon = '1' then
130         if spriteon1 = '1' then --affichage ash gauche
131             red <= M1(11 downto 8);
132             green <= M1(7 downto 4);
133             blue <= M1(3 downto 0);
134         elsif spriteon4 = '1' then --affichage ash droit
135             red <= M4(11 downto 8);
136             green <= M4(7 downto 4);
137             blue <= M4(3 downto 0);
138         elsif spriteon2 = '1' then --affichage pokeball
139             red <= M2(11 downto 8);
140             green <= M2(7 downto 4);
141             blue <= M2(3 downto 0);
142         elsif spriteon3 = '1' then --affichage cage gauche
143             red <= M3(11 downto 8);
144             green <= M3(7 downto 4);
145             blue <= M3(3 downto 0);
146         elsif spriteon5 = '1' then --affichage cage droit
147             red <= M5(11 downto 8);
```

Flowchart

Puis, pour la partie en C nous avons le flowchart suivant :



Le but de mon programme c'est de contrôler les ashs a l'aide des 4 boutons de la carte. Le bouton up et left me servent à bouger le ash de gauche, alors que le bouton right et down servent à bouger le ash de droite. Le bouton up sert également à reprendre le jeu lorsque l'un des deux joueurs marque un but.

L'idée dans ma tête c'était de faire un jeu le moins bloquant possible, tout en étant très intuitif et facile a jouer.

Plusieurs fonctionnalités auraient pu être rajoutées en ayant un peu plus de temps pour le finir. Comme un écran de debut, un sprite de game over à la fin, plusieurs niveaux de difficultés avec les switches que je n'utilise pas dans mon code, et plein d'autres.

Lien entre le soft et le hard

Un cas facile pour expliquer le lien entre la partie soft et la partie hard peut etre l'affichage de la couleur background dans mon jeu. En effet, pour la partie hard je donne le registre background a mon vga basic rom, qui vient donc mettre à la couleur désiré (donné par la valeur du registre background) les 3 trois composantes rgb de chacun des pixels qui ne soient pas déjà allumés par un sprite.

```
112             sw => Background (11 downto 0),

142         elsif spriteon3 = '1' then --affichage cage gauche
143             red <= M3(11 downto 8);
144             green <= M3(7 downto 4);
145             blue <= M3(3 downto 0);
146         elsif spriteon5 = '1' then --affichage cage droit
147             red <= M5(11 downto 8);
148             green <= M5(7 downto 4);
149             blue <= M5(3 downto 0);
150         elsif M6/='0' then
151             red <= "1111";
152             green <= "1111";
153             blue <= "1111";
154         elsif vidon ='1' then --affichage du fond en bleu
155         red   <= sw(11 downto 8);
156         green <= sw(7 downto 4);
157         blue  <= sw(3 downto 0);
158         end if;
```

Ce registre est ensuite changé dans mon programme main, pour que quand un joueur gagne le background devienne rouge, et quand un joueur click sur le bouton de rejouer la couleur revienne à 112 ce qui fait le vert foncé du background :

```
397         VGA.background = background_color;
398         // Si le jeu est terminé, remettre le jeu à zéro (réinitialiser les variables de jeu)
399         while (game_over== true)
400         {
401
402
403             if (BTNU)
404             {
405                 game_over = false; // Réinitialiser l'état du jeu
406                 background_color = 112; // Initialement vert
407             }
```

Et voici les résultats visibles des deux cas :