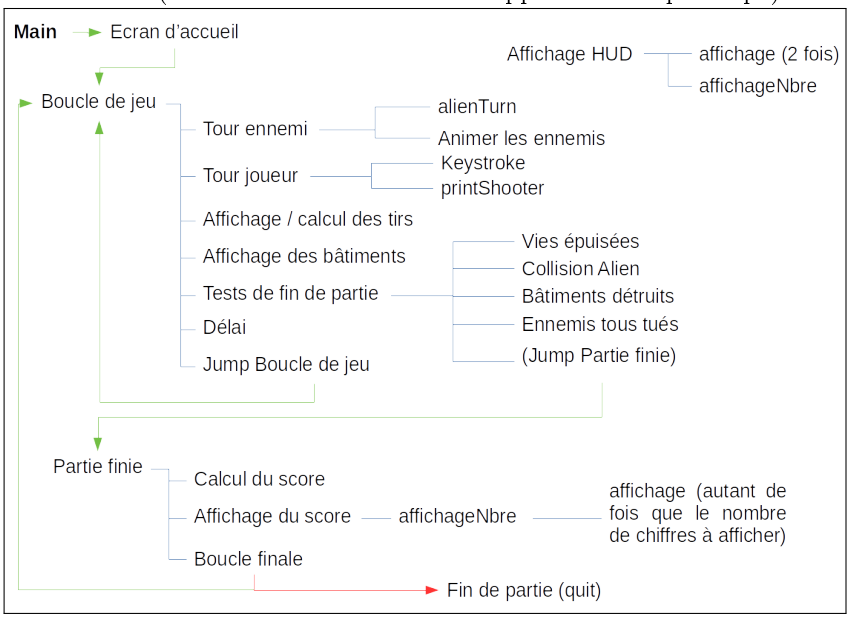
Projet d'Architecture : Mars Invaders

Louis Politanski, Louis Wermelinger

1 Choix d'implémentations

1.1 Organisation générale du programme

Afin de donner une idée générale de l'organisation choisie pour notre programme, on peut représenter l'ordre d'exécution des étapes du programme par le diagramme suivant (non exhaustif des fonctions appelées à chaque étape):



La boucle d'accueil et la boucle finale sont des boucles simples qui permettent toutes deux au joueur soit de (re)jouer soit de quitter le programme, et se distinguent par des différences d'affichage. Elles utilisent pour cela la fonction *keyStroke* qui était fournie à la base dans le sujet.

La boucle de jeu est la boucle principale où s'effectue la partie en elle-même.

1.2 Gestion de la vitesse d'exécution des différentes étapes de la boucle

Pour ralentir les différentes étapes de notre programme les unes par rapport aux autres, on a défini un pas de boucle stocké dans $s3. On exécute ensuite chaque seulement si le pas de boucle est multiple d'un nombre que l'on définit.

1.3 Test des vies restantes des ennemis et bâtiments

Pour vérifier si les ennemis ont tous été vaincus ou si les bâtiments sont tous détruits on a créé une fonction *vieRestante* qui renvoie la somme des éléments d'un tableau qu'elle parcourt. Il aurait été possible de créer une fonction qui quitte le programme si elle ne rencontre aucune valeur égale à 1 lors du parcours, mais l'avantage de notre fonction est qu'elle peut être réutilisée pour le calcul du score.

1.4 Méthode de détection de la collision

Les hitbox des ennemis et des bâtiments étant toutes des rectangles, nous avons choisi d'utiliser un algorithme de détection de collision simple et connu, celui des boîtes englobantes alignées sur les axes (axis-aligned bounding box), qui consiste en 4 tests de positions des 4 axes des côtés des rectangles (les aliens ne pouvant pas passer en dessous des bâtiments -sans collision- dans notre cas particulier on peut omettre l'un des 4 tests).

1.5 Méthodes d'affichage dans le bitmap et calcul du score

On a calculé le score en sommant le nombre d'ennemis vaincus et les bâtiments restants, avec des multiplicateurs.
Pour l'affichage on a défini une fonction *affichage* affichant des tableaux de données correspondant avec chaque mot du tableau codant une couleur (0xffffffff pour blanc par exemple). On aurait cependant pu choisir d'avoir une fonction travaillant avec des tableaux de booléens qui prend en paramètre une couleur et affiche celle-ci dans la case correspondante.
On a réutilisé cette fonction pour créer la fonction *affichageNbre* et celles d'affichages de mots. Celles-ci se basent sur des tableaux que nous avons créés correspondants aux chiffres et aux lettres.

2 Répartition des tâches

Les tâches étant relativement indépendantes, nous avons pu séparer le travail. L. Wermelinger s'est chargé du test des vies du joueur, des ennemis et des bâtiments. L. Politanski s'est chargé de la collision et des fonctions d'affichage. Nous avons fait ensembles les tableaux de chiffres et de lettres nécessaires à l'affichage, ainsi que la modification des formes du joueur et des ennemis.

3 Difficultés rencontrées

Déterminer comment calculer l'indice de tableau où afficher dans le bitmap *frameBuffer* n'était pas évident. Le manque de clarté sur le fonctionnement des fonctions précodées fournies a rendues certaines tâches plus difficiles, notamment l'animation des ennemis qui nécessitait de re-effacer une partie de l'écran.

4 Conclusion

Le sujet était intéressant car il demandait de résoudre des problèmes variés,avec des voies d'améliorations intéressantes, mais on aurait aimé avoir des fonctions précodées dont on pouvait mieux connaître le fonctionnement.