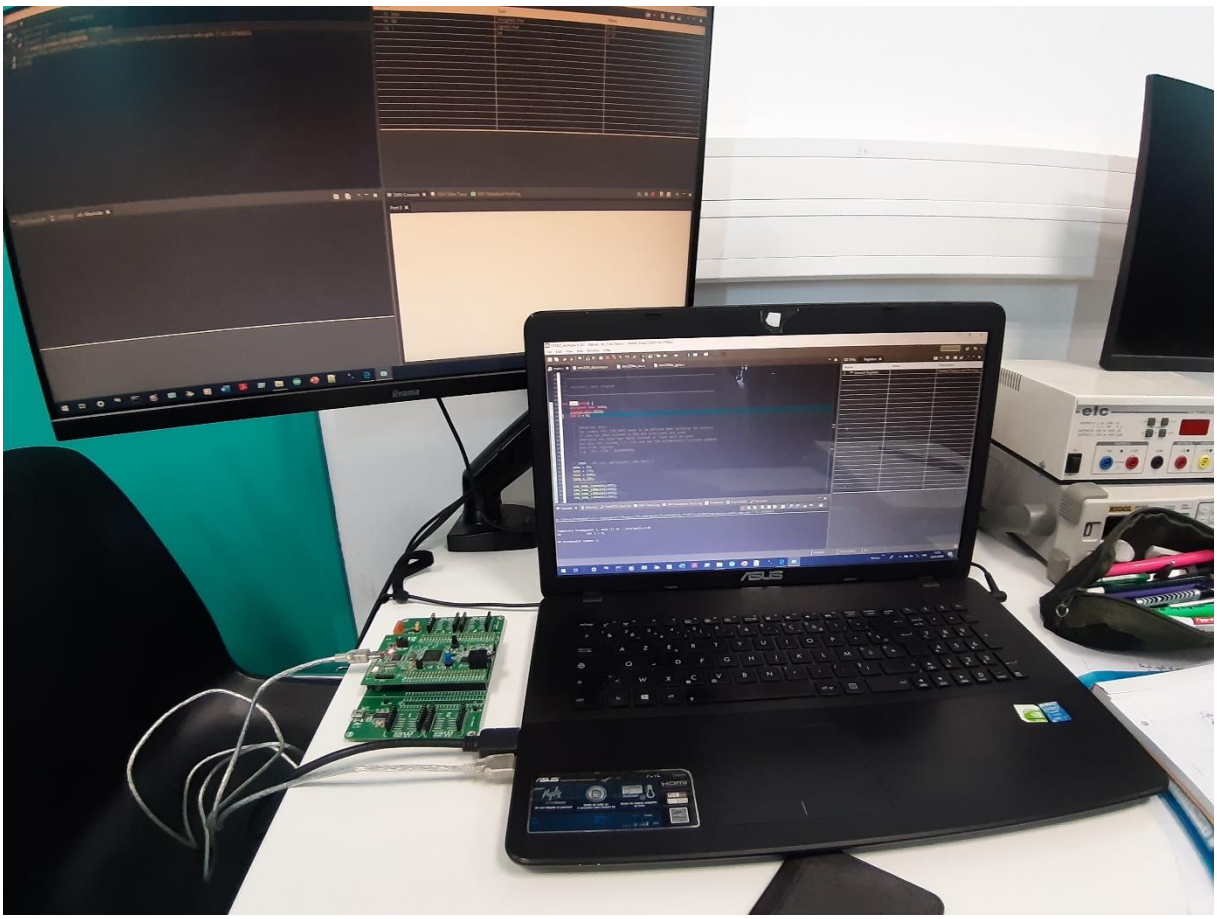
Compte rendu projet

Space Invaders

Objectif : Création du jeu Space Invaders pour carte STM32F4 Discovery

Repository git : https://github.com/will33400/projet-space-inviders/tree/master/space_invaders

Sommaire :



1/ Conception préliminaire :

- Cahier des charges :

Diagramme SADT :

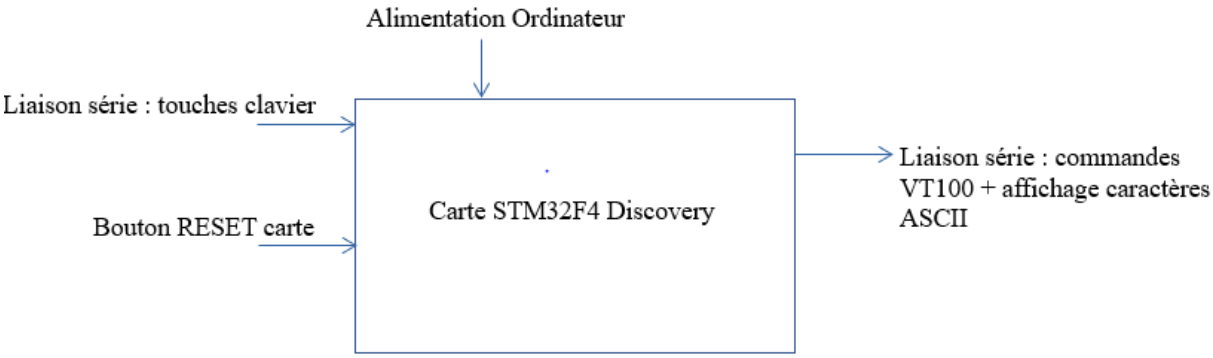


Diagramme bête à cornes :

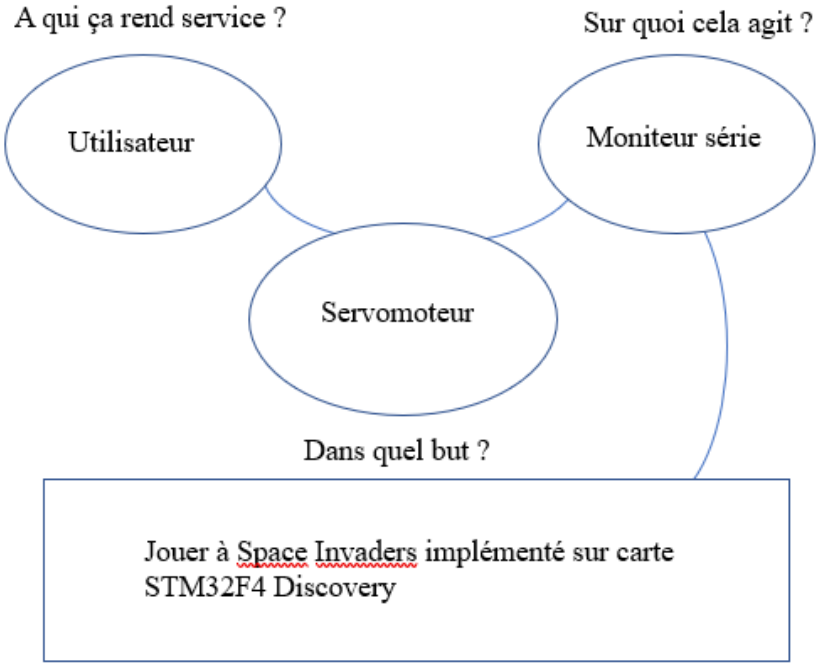
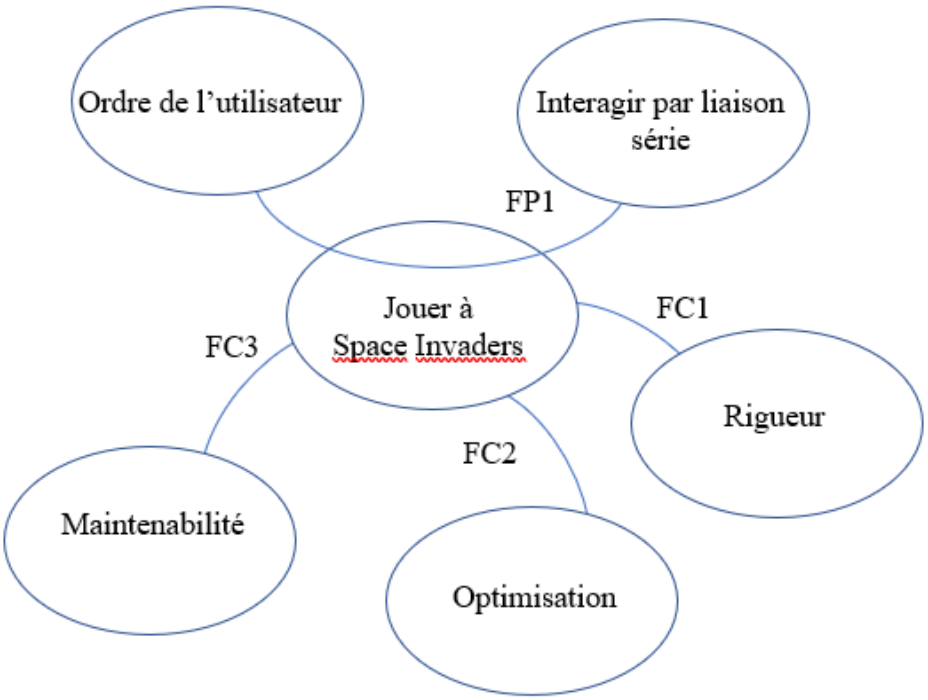


Diagramme fonctionnel :

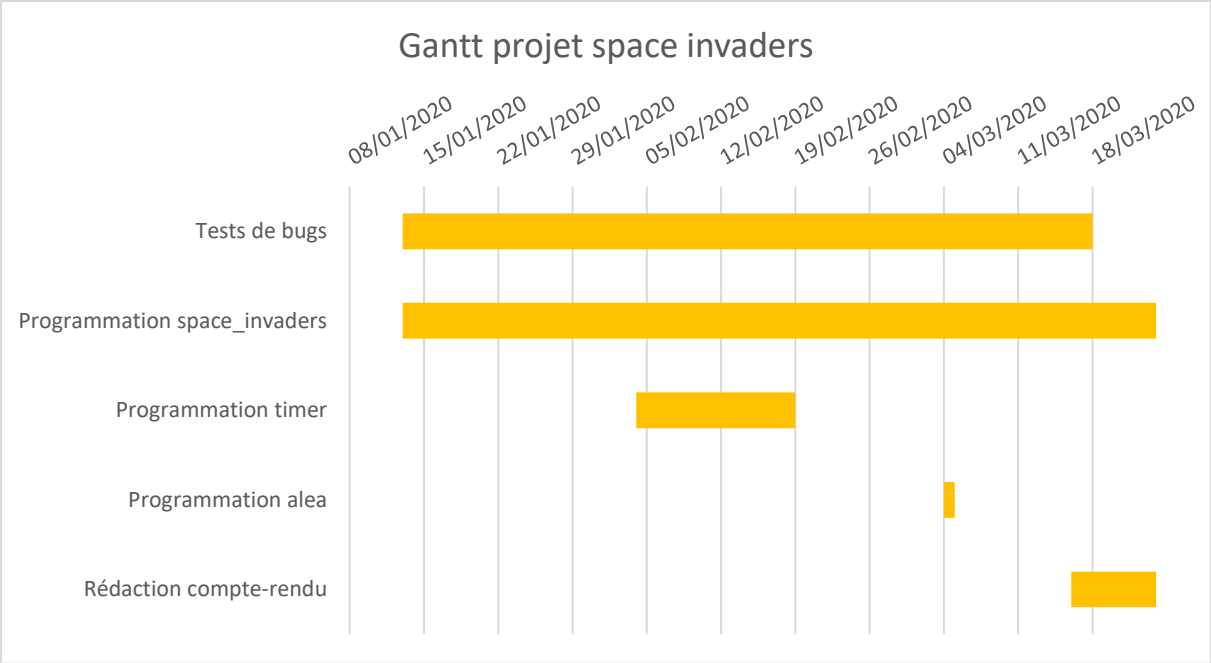


FP1 : Interagir avec la carte STM32F4 Discovery par liaison série avec un ordinateur
FC1 : Être rigoureux
FC2 : Être optimisé
FC3 : Être maintenable

Fonction	Critères	Niveau	Flexibilité
FP1 : Interagir avec la carte STM32F4 Discovery par liaison série avec un ordinateur	Gestion de l'alimentation	- 5V de l'ordinateur passant par câble USB (qui gère la programmation et/ou la liaison série)	F5
	Gestion programmation	- Utilisation du logiciel Attolic TRUEStudio	
	Gestion liaison série	- Utilisation d'un émulateur de liaison série (PuTTY)	
FC1 : Être rigoureux	Gestion du code en C	- Rendre le code lisible	F0
FC2 : Être optimisé	Gestion du code en C	- Rendre le code organisé	F0
FC3 : Être maintenable	Gestion du code en C	- Rendre le code facilement modifiable	F0

Diagramme gantt :

Tâches	Début	Nombre de jours	Fin	Ecart
Tests de bugs	lundi 13 janvier 2020	13	mercredi 18 mars 2020	65
Programmation space_invaders	lundi 13 janvier 2020	14	mardi 24 mars 2020	71
Programmation timer	mardi 4 février 2020	3	mercredi 19 février 2020	15
Programmation alea	mercredi 4 mars 2020	1	jeudi 5 mars 2020	1
Rédaction compte-rendu	lundi 16 mars 2020	4	mardi 24 mars 2020	8



- Règles du jeu :

(Ce sont des règles que j'ai personnellement arrangées)

La règle du jeu consiste à finir les 3 niveaux en éliminant tous les vaisseaux ennemis se trouvant à l'écran.

Particularités :

Les ennemis se déplacent en serpentant du haut vers le bas de l'écran. Ils tirent « au hasard » à l'aide d'une variable qui change de valeur au démarrage de la carte STM32F4 Discovery. Pour éviter que certains vaisseaux ne puissent pas tirer, une condition permet de savoir si le vaisseau qui devait tirer existe encore. Si ce n'est pas le cas, il va alors passer à l'ennemi suivant et vérifier s'il existe etc. La cadence de tir est rapide d'où la réelle nécessité de se protéger sous un des 3 boucliers.

Toucher avec un tir son propre bouclier peut l'entamer et les ennemis font de même.

Les tirs du vaisseau du joueur peuvent repousser des tirs ennemis (et inversement) et ainsi obtenir 10 points de score. Un vaisseau ennemi éliminé rapporte 200 points de score.

Lorsque l'on passe à un niveau suivant, le vaisseau du joueur récupère toute sa vie. Le vaisseau du joueur possède 5 points de vie.

Il n'y a aucune différence de difficulté entre chaque niveau, c'est juste l'apparence des ennemis qui change.

Si au moins 1 ennemi atteint la même ligne que celle du vaisseau du joueur, cela génère automatiquement un game over, le vaisseau joueur étant incapable de tirer horizontalement.

Le vaisseau contrôlé par le joueur ne peut se déplacer qu'horizontalement.

Fidèlement au jeu original, le jeu s'accélère au fur et à mesure que le nombre d'ennemis sur l'écran se réduit pour des raisons de performances.

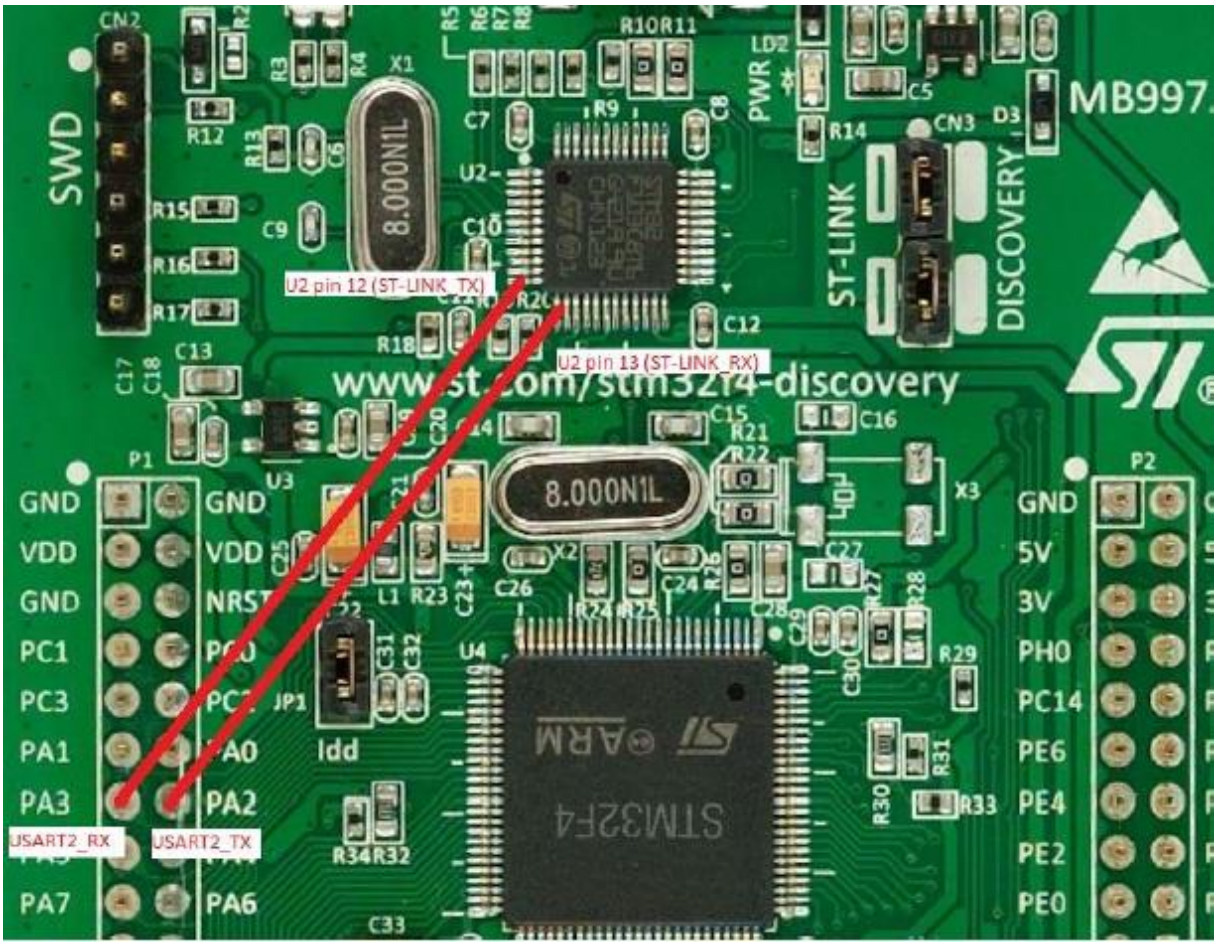
Le meilleur score (highscore) est toujours conservé même après un game over. Lors d'un game over, attendre quelques secondes pour revenir automatiquement à l'écran de démarrage du jeu. Appuyer sur le bouton RESET de la carte va effacer le meilleur score…

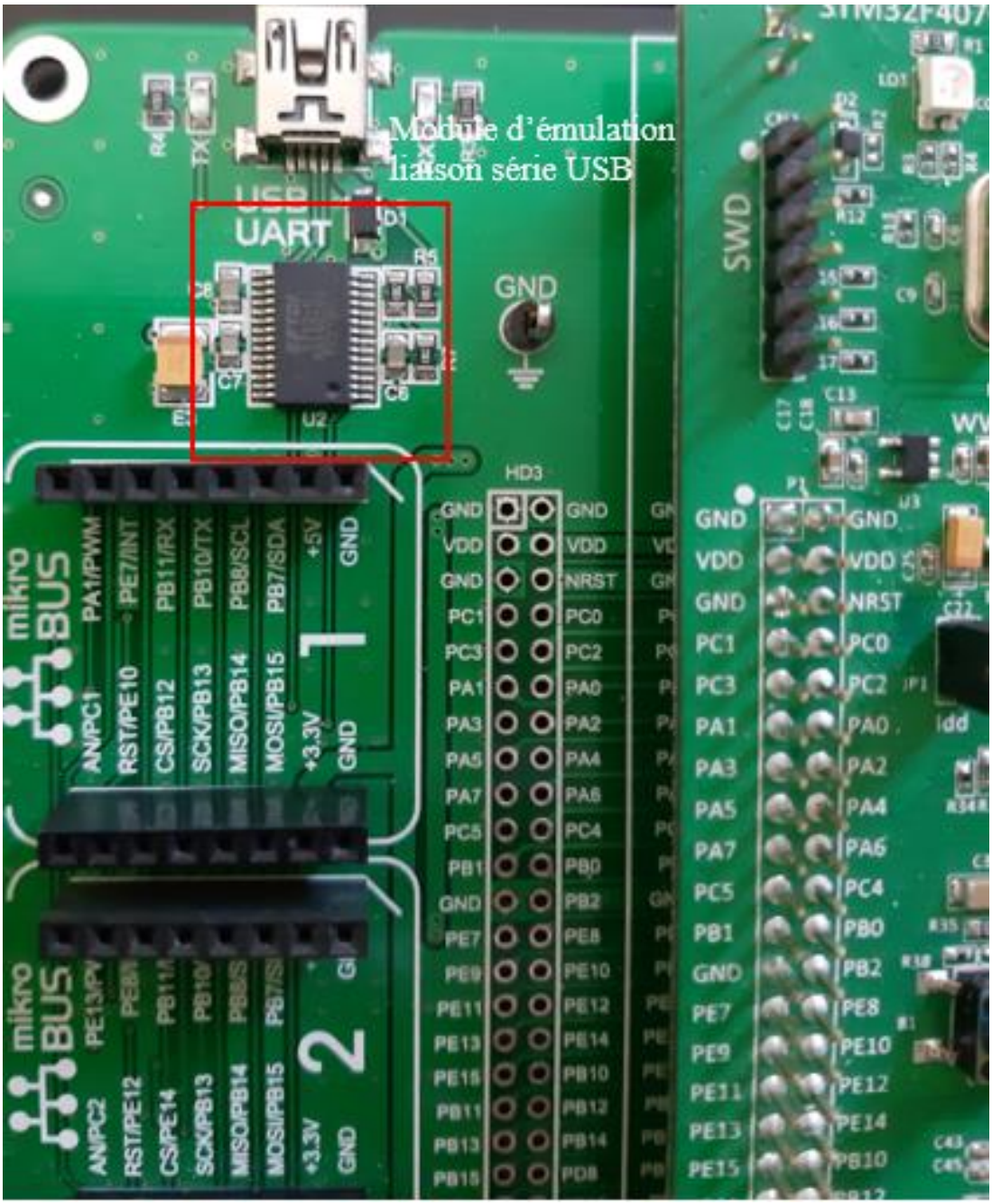
2/ Description bibliothèques :

Bibliothèques fournies :

Serial :

C'est la librairie qui permet de gérer la liaison en UART vers un module de la carte STM32F4 Discovery Shield qui sert d'émulateur relié en USB à l'ordinateur :





La bibliothèque dans le .h contient :

```
void serial_init(const uint32_t baudrate);
```

Initialisation de la liaison série en précisant le nombre de bauds (bits par secondes). La valeur utilisée pour le jeu est 115200bps.

```
void serial_putchar(const volatile char c);
```

Envoi d'un caractère ASCII.

```
void serial_puts(const volatile char *s);
```

Envoi d'une chaîne de caractères ASCII en passant par un pointeur pointant sur le premier élément de la chaîne.

signed char serial_get_last_char(void);

Réception d'un caractère ASCII.

La bibliothèque dans le .c contient :

Définition de la fonction **serial_init() :**

Utilisation de structures permettant le paramétrage des registres gérant :

- Entrées / Sorties (GPIO)
- Liaison série (UART)
- Interruptions (NVIC)

La définition de ces structures se trouvent dans les librairies définies dans **space_invaders>Libraries>STM32F4xx_StdPeriph_Driver>inc** accessibles dans le projet.

Le plus important à savoir pour la liaison série :

- Taille de l'information : 8bits
- Mode : TX / RX actifs
- Parité : aucune
- Contrôle de flot : aucun
- Bit de stop (longueur) : 1bit
- N° USART du composant : USART2 (PA2 : TX, PA3 : RX)

Gestion des interruptions :

serial_input_not_empty()

Cette fonction renvoie un état différent de 0 lorsque le registre de réception des informations de la liaison série n'est pas vide.

serial_input_character()

Cette fonction donne le contenu du registre DR qui contient la donnée reçue.

void serial_it_handler()

Routine d'interruption qui mène à l'adresse de la mémoire de programme vers lequel le stack pointer va aller lorsqu'il y a une interruption générée par USART2_IRQHandler.

Vt100 :

Cette bibliothèque gère la partie orientée graphique de la liaison série :

La bibliothèque dans le .h contient :

Des #defines de constantes qui gèrent le dimensionnement de la fenêtre du moniteur et un qui remplace la touche escape par sa forme ASCII.

void vt100_move(uint8_t x**, uint8_t** y**);**

Cette fonction permet de changer la position du curseur.

`void vt100_clear_screen(void);`

Cette fonction efface l'écran du moniteur série.

Plus d'informations sur https://www.vt100.net/.

Bibliothèques personnelles :

Timer :

J'ai créé cette bibliothèque car il m'a semblé nécessaire de faire une gestion du parallélisme des actions (scroll vaisseaux ennemis, scroll vaisseau du joueur, scroll des tirs) pour gagner grandement en performance et ainsi avoir un affichage du jeu fluide. Je voulais faire en sorte que lorsque le registre qui stocke la valeur de temps correspond à une valeur définie (time base), cela génère une interruption qui active une des fonctionnalités citées ci-dessus.

`init_tim_base() ;`

Cette fonction initialise le timer souhaité en mode time base avec la valeur de temps souhaitée. La façon dont a été paramétré le timer est sous la même forme que pour l'USART plus haut.

Paramétrage :

- Prescaler : 0
- Période : valeur choisie
- Division de l'horloge : 1
- Mode de comptage : décrémentation
- Mode de répétition : répétitif
- Timer utilisé : TMR2 ou TMR3

La fonction ne marche pas d'où le fait que je n'ai pas pu m'en servir pour gérer les cadences du jeu.

Alea :

J'ai séparé cette fonction de la librairie space_invaders pour le différencier.

La variable LFSR est une variable accessible par les fonctions set et get (comme en orienté objet).
`void LFSR_update(void);` comme son nom l'indique, met à jour la valeur de LFSR en lui attribuant une valeur aléatoire comprise entre 0 et 255.

Space_invaders :

Defines :

- Gestion du baud rate
- Nombre de tirs max gérées par les événements des tirs
- Vie max du vaisseau du joueur
- Apparence du tir du vaisseau du joueur
- Vie des vaisseaux min
- Vie des vaisseaux max
- Apparence des tirs des vaisseaux ennemis
- Les différents types de tirs
- Nombre de structures gérant les ennemis max
- 3 dimensions max de de vaisseaux
- Dimensions des boucliers
- Nombre max de boucliers
- Nombre de niveaux
- Positions des chaînes de caractères qui affichent la vie et le score
- Définitions des points de score lors de la collision avec un type de hitbox

Enums :

- Booléen
- Types d'objets (volant ou non)
- Trois tailles arbitraires
- Directions
- Types d'écrans

Remarque :

Tout objet défini dans le jeu, qu'il vole ou non, possède une unité, la hitbox. Tout objet possède au moins une hitbox ou un groupe de hitbox sous forme de tableau qui est constitué :

- Genre d'objet
- Apparence
- Position x
- Position y
- Existence (booléen)

La variable qui stocke les tableaux des hitbox est sous forme d'union donc on peut facilement implémenter une nouvelle forme d'objet dans le code.

La structure qui gère les objets volants (flying_object_s) est pourvu :

- Vie
- Taille (enum)
- Tableau hitbox (union)
- Existence

Level_s est constitué de :

- Direction de déplacement du tableau d'ennemis
- Variable qui copie LFSR pour dire quel vaisseau ennemi tire
- Nombre d'ennemis restant dans le niveau
- Tableau qui stocke les structures des ennemis
- Variable qui stocke la structure du vaisseau du joueur
- Tableau qui stocke tous les tirs du niveau (s'il y a surplus, un tir encore existant peut-être remplacé par un autre tir !!)
- Index qui pointe vers une case du tableau qui stocke les structures des tirs
- Score
- Valeur du score courant
- Vie actuelle du vaisseau du joueur
- Taille des ennemis
- Valeur à implémenter à la fonction qui gère des délais d'attente

`destroy_hitbox``();`

Remet à 0 et efface sur l'écran une hitbox.

`init_flying_object``();`

Initialise les paramètres de base d'un objet volant.

`destroy_flying_object``();`

Remet à 0 et efface sur l'écran un objet volant.

`update_coordinates``();`

Attribue des coordonnées à un objet volant.

`display_flying_object``();`

Affiche un objet volant.

`erase_flying_object``();`

Efface de l'écran un objet volant.

`scroll_flying_object``();`

Déplace un objet volant déjà affiché.

`flying_object_move_limit``();`

Renvoie un booléen pour dire s'il y a collision ou non par rapport à un bord de l'écran lorsque qu'un objet volant se déplace dans une direction quelconque avec un offset quelconque.

`object_hitbox_collision``();`

Renvoie un booléen pour dire s'il y a collision entre deux hitbox quelconque.

`object_list_append``();`

Ajoute un objet volant dans un tableau de structure d'objets volants.

object_list_destroy();

Détruit tous les objets volants dans un tableau de structures d'objets volants.

object_list_display();

Affiche tous les objets volants du tableau de structures d'objets volants.

object_list_erase();

Efface sur l'écran tous les objets volants du tableau de structures d'objets volants.

object_list_scroll();

Déplace tout un tableau de structure d'objets volants dans une direction particulière avec un offset particulier.

init_shoot();

Créé une hitbox volante de type tir.

erase_shoot();

Efface sur l'écran le tir.

init_shield();

Initialise un bouclier (3 max)

A l'aide de toutes les fonctions définies ci-dessus, il m'est possible de gérer la génération, la destruction, l'affichage, l'effacement et la collision de tout objet ou hitbox.

Evénements :

Les événements sont fondamentaux pour le séquençage de toutes les fonctionnalités du jeu :

3/ Description du main.c

Pour mieux comprendre le fonctionnement des événements, il faut voir du côté pratique :

Avant la boucle dans **main**(), j'initialise LFSR (variable aléatoire) à une valeur quelconque (qui servira pour la gestion de tirs aléatoires pour les vaisseaux ennemis) puis j'initialise les paramètres de base de l'écran. J'affiche ensuite le premier écran appelé MENU puis on passe dans la boucle qui contient **charge_levels_events**() qui contient à son tour :

- Mise à jour de la variable aléatoire
- Gestion des événements du clavier avec **compute_keyboard**()
- Selon l'état de la variable qui possède le numéro d'écran actuel lance les événements d'un niveau (excepté MENU et GAME_OVER).

levels_events() possède par conséquent les caractéristiques suivantes :

- Vérification que le jeu ne se trouve pas en PAUSE, MENU ou GAME_OVER
- Récupération + masque de la valeur aléatoire
- En fonction de la taille des ennemis : récupération de la structure des vaisseaux ennemis se trouvant le plus à gauche, à droite, en haut et en bas pour la gestion des collisions.

- Déplacement de ce groupe d'ennemi en serpentant
- Faire tirer un ennemi au hasard tant qu'il existe sinon tester pour l'ennemi suivant
- Délai d'attente
- Evénement des tirs x2 (les tirs se déplacent 2x+ vite que les autres objets volants)
- En fonction de la taille du vaisseau ennemi se trouvant le plus en bas, vérification s'il se trouve sur la même ligne que le vaisseau du joueur entraînant un GAME_OVER
- Vérification si le nombre d'ennemi est nul, ce qui renvoie le jeu vers le prochain niveau

Description des événements des tirs **`shoot_events()`** :

- Boucle qui déplace tous les tirs :
- Si c'est un tir ennemi, le tir descend puis vérification s'il y a collision avec le vaisseau du joueur ou avec un tir du vaisseau du joueur
- Sinon si c'est un tir du joueur, le tir monde puis vérification s'il y a collision avec un vaisseau ennemi ou avec un tir d'un vaisseau ennemi
- Vérification si un tir quelconque a touché un bouclier

4/ Bilan

En travaillant activement durant tous les cours de « suivi de projet » et également à la maison (environ 2 semaines), j'ai réussi à mettre au point un jeu vidéo adapté à une carte de type STM32.

J'ai rencontré certains problèmes qui étaient causés par des constantes non définies et arbitraires dans la définition de fonctionnalités simples, qui ont généré des bugs et m'ont pris plus de temps afin de les résoudre (ex : fonction d'initialisation de structures).

Néanmoins, j'ai fini par être capable d'implémenter des fonctionnalités complexes pour des applications software.

Du reste, je passe le plus clair de mon temps libre à faire fonctionner des microcontrôleurs Atmel et PIC et je prends beaucoup de temps pour résoudre des problèmes hardware générant très souvent des problèmes software d'un point de vue de la stabilité ou autres.

5/ Conclusion

En réalisant ce projet, je me suis prouvé que j'étais capable d'implémenter des solutions softwares plus évoluées par rapport à ce que j'avais l'habitude de faire chez moi.

Lors de ce projet, les cours m'ont appris un certain nombre de choses dont je n'étais pas conscient.

Cela m'a permis notamment d'évoluer dans mon approche de l'utilisation d'instructions en C, comme les **union** ou les **struct**.

J'ai aussi appris en cours à implémenter facilement une fonction qui génère une valeur aléatoire.

J'ai acquis en autodidacte comment gérer des mécaniques spécifiques à un jeu vidéo de type « shoot them up », ce que j'ai trouvé très intéressant.

Pour terminer, j'aurais aimé être capable d'utiliser des interruptions avec un time base, pour la gestion des cadences de tirs et le déplacement des objets volants.