ROBOTIQUE &
INGÉNIERIE SYSTÈMES

Alexandre
Le Bourch
26 mars 2023

Rapport

Projet mini-jeu :

Space Invaders

LE PROJET :

Principe du projet :

Dans le cadre du cours de micro-python nous avons eu l'occasion de créer notre propre mini-jeu. Le mini-jeu devait être réalisé sur une STM32, nous avions quelques contraintes à respecter pour le réaliser :

- utilisation de la communication UART
- utilisation de la communication SPI
- utilisation d'un Timer
- utilisation d'un bouton

Mon projet :

Pour ce jeu j'ai choisi de reproduire le principe de fonctionnement du jeux « space invaders ». Pour répondre aux contraintes imposées, je me suis donné des objectifs principaux à atteindre pour être sur de respecter ces contraintes, une fois ces objectifs atteints, j'ai continué le développement du jeu pour le rendre plus agréable.

Contrairement au jeu « space invaders » où les ennemis apparaissent en hauteur, dans mon projet les ennemis apparaitront sur la gauche de mon écran. Une barre de vie sera en haut de l'écran pour indiquer la vie des ennemis. Pour me faciliter la conception de ce projet, j'ai décidé d'uniquement faire apparaitre des ennemis, ils ne seront pas capables de tirer sur le joueur ni de se déplacer et ils apparaitront à des emplacements aléatoires sur l'écran. Le joueur sera capable de se déplacer et de tirer des rockets pour tuer les ennemis, à chaque mort d'un ennemi la barre de vie perdra de la santé.

Pour remplir toutes les contraintes données, j'utilise VT100 qui passe par la communication UART, j'utilise aussi l'accéléromètre de la carte qui communique en SPI. Lorsque tous les ennemis seront éliminés, les LEDs de la carte se mettront à clignoter pour signifier la fin de la partie. Le bouton me permet de valider la sélection d'un menu au début du jeu ou bien d'envoyer une rocket pendant la partie.

AFFICHAGE DU JEU :

Pour afficher mon jeu, j'ai décidé d'utiliser un terminal en utilisant le mode VT100, pour cela il faut que j'utilise la communication UART. On initialise la communication avec cette ligne de code :

```
uart = UART(2, 115200)
```

'115200' correspond à la vitesse de communication

Ensuite nous pouvons utiliser '***uart.write('mon_text')***' pour écrire ce que l'on veut dans le terminal. Dans notre cas, nous avons besoin de bouger le curseur qui définit le point à partir duquel nous écrivons pour faire bouger le joueur ou de manière générale pour dessiner le jeu. Pour être en mesure de pouvoir effacer l'écran ou de dessiner des logos ou les bordures du jeux, j'ai créé plusieurs fonctions qui me permettent de le faire :

```
def clear_screen():
    uart.write("\x1b[2J\x1b[?25l")

def move(x, y):
    uart.write("\x1b[{};{}H".format(y, x))

def draw_logo(logo:str, x, y):
    lines = logo.splitlines()
    for index, line in enumerate(lines) :
        move(x, y + index)
        uart.write(line)
```

```
def clear_logo(logo:str, x, y):
    lines = logo.splitlines()
    for index, line in enumerate(lines) :
        move(x, y + index)
        uart.write(len(line)*" ")

def boundaries (boundaries_x, boundaries_y):
    for i in range(*boundaries_x):
        move(i, boundaries_x[0])
        uart.write("-")
    for i in range(*boundaries_y):
        move(boundaries_y[0], i)
        uart.write("|")
```

- la fonction '***move***' permet de bouger le curseur en fonction de x et y

- la fonction '***clear_screen***' permet d'effacer tout les éléments affichés sur le terminal

- les fonctions '***draw_logo***' et '***clear_logo***' permettent de dessiner et effacer de longues chaines de caractères, dans ce cas ce sont des ASCII Art de ce style :

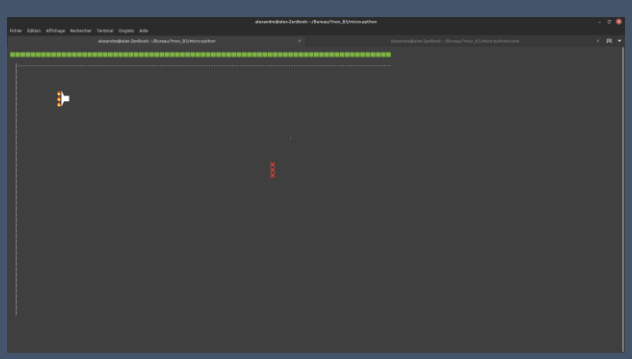
```
logo_play = """\
 _____ _
|  __ \| |
| |__) | | __ _ _   _
|  ___/| |/ _` | | | |
| |    | | (_| | |_| |
|_|    |_|\__,_|\__, |
                 __/ |
                |___/
"""
```

```
logo_score = """\
  _____
 / ____|
| (___   ___ ___  _ __ ___
 \___ \ / __/ _ \| '__/ _ \
 ____) | (_| (_) | | |  __/
|_____/ \___\___/|_|  \___|


"""
```

Pouvoir afficher ceci me permet de rendre le jeu plus visuel

- la fonction '***boundaries***' me permet de dessiner et d'initialiser les limites de l'espace de déplacement du joueur

Pour dessiner le joueur et les ennemis j'ai donc utilisé la fonction '***draw_logo***' : voici l'aperçu pour le menu du jeu et pendant une partie :



FONCTIONNEMENT DU JEU :

Pour faire fonctionner le jeu, j'ai dû créer plusieurs fonctions (hors celle expliquées précédemment), les voici :

- La fonction '***blink***' permet de faire clignoter toutes les LEDs à une fréquence de 8Hz, la fonction fait appel à '***leds***' qui permet de contrôler les LEDs en fonction d'un Timer.

```
# LEDS init #
def leds(timer): ···

def blink(): ···
#/////////////#
```

- Pour faire bouger le joueur nous devons utiliser l'accéléromètre de la carte. Pour ce faire nous utilisons la fonction '***read-acceleration***' qui nous renvoie l'accélération de l'accéléromètre en fonction X, Y et Z. Cette fonction fait appel à la fonction '***read-reg***' et '***convert_value***' ce qui nous permet d'avoir une valeur compréhensible. '***write-reg***' sera utilisé plus tard pour initialiser la communication avec l'accéléromètre.

```
# accelerometer functions #
def read_reg(addr): ···

def write_reg(addr, value): ···

def convert_value(high, low): ···

def read_acceleration(base_addr): ···
#/////////////#
```

- Les fonctions ci-contre sont toutes celles directement utilisées pour le jeu, les fonctions vues précédemment nous servaient à gérer l'affichage ou à initialiser des LEDs ou la communication. Ces fonctions sont utilisées en cascade, c'est-à-dire que la fonction '***shooting***' utilise la fonction '***rocket_hit_test***' qui utilise la fonction '***life-bar***' qui utilise la fonction '***restarting***'. Elles sont dépendante les unes des autres.

```
# Game functions #
def life_bar (life_lenght, hit = False): ···

def shooting (curseur_x, curseur_y, entry, ennemy_pos): ···

def rocket_hit_test (ennemy_pos, rocket): ···

def spawn_ennemy (): ···

def menu(btn): ···

def score(btn): ···

def restarting (): ···
#////////////#
```

La fonction '***shooting***' est appelée lorsque le bouton est pressé ce qui envoie une rocket. Pour savoir si un ennemi a été touché, on utilise la fonction '***rocket_hit_test***' qui va vérifier si l'ennemi est touché. Si c'est le cas, elle l'efface et en fait apparaitre un nouveau avec la fonction '***spawn_ennemy***' qui génère un ennemi sur une position aléatoire. Elle appelle aussi la fonction '***life_bar***' qui va faire baisser la vie des ennemis et si tous les ennemis ont été tués, alors elle fait appelle à la fonction '***blink***'. Ensuite, la fonction '***restarting***' relance une nouvelle partie. La fonction '***menu***' appelle la fonction '***score***' pour afficher une page de score.

Certaines fonction sont appelées au tout début du jeux pour initialiser la partie :

- '***menu***' lance le menu de départ avec la possibilité d'accéder à un menu de scores
- '***life_bar***' permet de dessiner la barre de vie
- '***spawn_ennemy***' fait apparaitre le premier ennemi

FONCTIONNEMENT DU JEU :

Pour que le jeu fonctionne correctement il faut associer toutes ces fonctions, c'est ce qu'il se passe dans la boucle while :

```
# THE GAME #
while True:
    # GAME SETUP#
    ennemy_pos = []
    number_hit = 0
    curseur_x = 20
    curseur_y = 10
    previous_pos_x = curseur_x
    previous_pos_y = curseur_y
    boundaries_x = [4,150]
    boundaries_y = [4,50]
    start = True
    first_display = True
    #############/#

    # game start #
    clear_screen()
    menu(push_button)
    boundaries(boundaries_x, boundaries_y)
    spawn_ennemy()
    life_bar(150)
    #############/#
```

Lorsque la boucle '***while***' commence le jeux initialise toutes les valeurs et listes nécessaires, ensuite le jeu commence par nettoyer l'écran et affiche le menu. Une fois le menu quitté la partie commence, les limites apparaissent ainsi que la barre de vie et le premier ennemi.

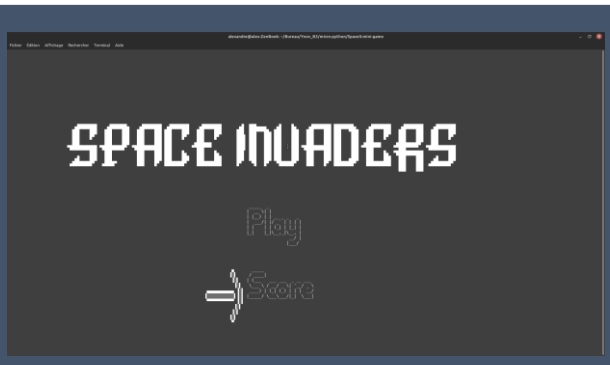
Ensuite le programme rentre une nouvelle fois dans une boucle '***while***' c'est dans celle-ci que toute la partie va se dérouler. La condition de cette boucle '***while***' correspond au nombre d'ennemis abattus. Dans cette boucle, le programme va acquérir les données de l'accéléromètre pour ensuite déterminer dans quelle direction va bouger le curseur. Le cinquième '***if***' dit que si la position précédente est la même que la position actuelle, on ne rafraichit pas la position sauf si '***first_display***' est vrai. Cette condition nous permet d'afficher le joueur la première fois mais aussi d'éviter comme un clignotement car le joueur s'efface et se réécrit sur la même position.

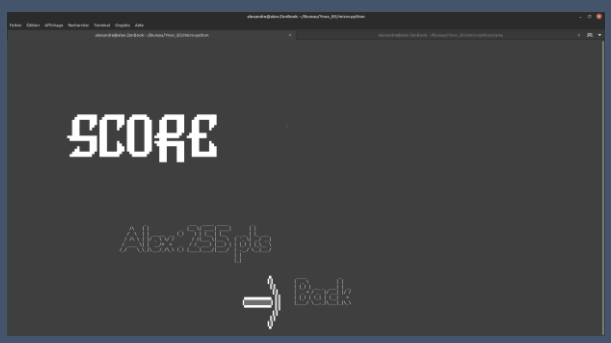
Ensuite on appelle la fonction '***shooting***' qui s'occupe d'actualiser le jeu si un ennemi est touché. D'une certaine manière, c'est la fonction '***shooting***' qui fait tout le travail du jeu, hormis pour le déplacement du joueur.

```
    while(number_hit != 5):

        x_accel = read_acceleration(0x28)
        y_accel = read_acceleration(0x2A)
        z_accel = read_acceleration(0x2C)

        delay(50)

        if x_accel < 300:···

        if x_accel > -300:···

        if y_accel > 300:···

        if y_accel < -300:···

        if curseur_x != previous_pos_x or curseur_y != previous_pos_y or first_display == True:
            clear_logo(player, previous_pos_x, previous_pos_y)
            previous_pos_x = curseur_x
            previous_pos_y = curseur_y
            draw_logo(player, curseur_x, curseur_y)
            first_display = False

        shooting(curseur_x, curseur_y, push_button.value(), ennemy_pos)
#############/#
```

FONCTIONNEMENT DU JEU :

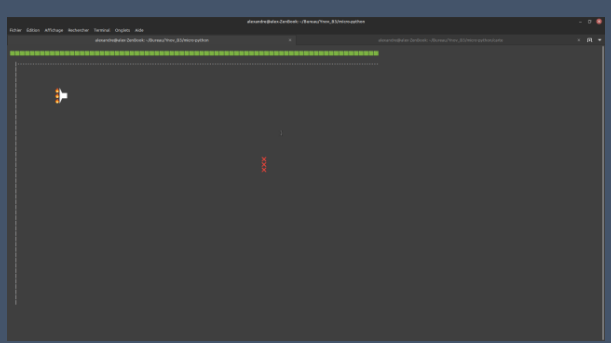
Quand le jeux démarre, on a le choix entre deux menus, le tableau des scores ou jouer :





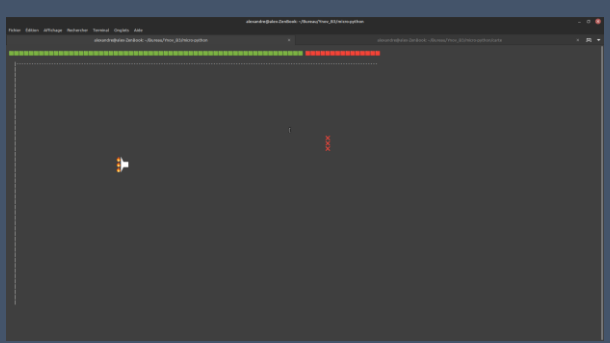
Si on choisit le tableau des scores voici ce qui s'affiche

on peut ensuite revenir en arrière pour choisir le jeu



La partie commence !

Un ennemi tué !



Une fois tous les ennemis tués, cet écran s'affiche avec une animation disant « Restarting . . . », une fois l'animation terminée le jeux revient sur l'écran de départ.

CONCLUSION :

Ce projet était très intéressant à réaliser, il m'a permit de revoir chaque aspect du cours tout en approfondissant mes connaissances sur certaines fonctionnalités. Lors de la réalisation de ce projet j'ai rencontré plusieurs difficultés, en voici 3 :

- Lorsque le joueur se déplace il fallait effacer sa position précédente pour éviter des bugs sur l'écran, ce problème fut réglé en mettant le même nombre de caractère sur chaque ligne du logo.
- Lorsque le joueur tire une rocket en direction d'un ennemi, il fallait que le jeu soit capable de détecter si l'ennemi a été touché, or l'ennemi existe sur trois caractères et je n'en enregistre qu'un seul. J'ai donc créé la fonction '***rocket_hit_test***' qui teste si la rocket a touché l'ennemi en fonction d'un offset.
- Lorsqu'un ennemi vient d'être abattu, je voulais que la barre de vie diminue de plus en plus. Pendant le développement de cette fonctionnalité il y a eu plusieurs problèmes. Le signe Unicode que j'ai choisi pour la dessiner est composé de 2 caractères, ce qui causait plusieurs bugs. Suite à cela, lorsque la barre de vie diminuait, la barre rouge apparaissaient derrière les ennemis sans raison, le problème était dû à la formule qui déterminait le nombre de caractère à écrire ainsi que leurs positions.

Ce projet m'a permit d'approfondir mes connaissances en micro-pyhton avec les communications UART et SPI, mais aussi sur l'utilisation des Timers.

Les améliorations que j'apporterais à mon code seraient les suivantes :

- Avoir une vraie page de scores avec des scores enregistrés dans un fichier (exemple .json)
- Avoir les ennemis qui bougent et qui tirent sur le joueur et rajouter une barre de vie pour le joueur
- Pouvoir changer le nombre d'ennemis et rajouter un mode vague d'ennemis avec un boss final

Ce que je trouve d'original dans mon jeu est l'aspect visuel que je lui ai donné avec les ascii art, la barre de vie ou encore le design du joueur et des ennemis. J'ai utilisé au maximum les caractères Unicode pour améliorer l'aspect visuel de mon jeu.