Rapport du projet n°3 de NSI
Space Invader

Romain Seznec

Mai 2021

Introduction

Le projet *Space Invader* est un programme Python prenant la forme d'un jeu-vidéo de tir rétro et utilisant le moteur graphique Processing. L'élaboration du projet est dirigée par les consignes décrites dans le fichier `README.md`. Une vidéo de démonstration du jeu en action est disponible à l'adresse suivante : `https://cloud.cyborger.net/s/RrGWt9KHTBZnSqp`. Ce rapport vous permettra de mieux comprendre le fonctionnement du programme ainsi que mes choix quant à l'organisation du code.

1 L'utilisation d'un dictionnaire unique

Afin de palier au fait de devoir éviter l'utilisation de classes et, par conséquent, de la programmation orientée objet, je pris rapidement le choix de n'utiliser qu'un unique dictionnaire intitulé `jeu` qui concentre toutes les propriétés du jeu.

1.1 Architecture principale

Architecture du dictionnaire `jeu` à l'initialisation de celui-ci. (Cliquez sur les éléments pour accéder à leurs détails).

- `jeu`
  - `statut`
  - `images`
    - `...`
  - `polices`
  - `couleurs`
    - `rouge`
    - `vert`
    - `bleu`
    - `blanc`
  - `sauvegarde`
    - `record`
    - `couleur`
    - `vies`
    - `max_ennemis`
    - `vitesse`
    - `debug`

Détail de l'architecture

statut Chiffre entier défini de 0 à 3 et permet d'indiquer l'état du jeu allant de *Menu Principal*, *Jeu en cours*, *Game Over* et *Options* respectivement.

images Dictionnaire contenant 4 autres dictionnaires avec pour clé les couleurs données dans la clé `couleurs` de la variable jeu qui servent à contenir les trois images du jeu (associés à leur couleur) : `joueur.png` ; `ennemi.png` ; et `projectile.png`.

polices Dictionnaire contenant les polices d'écriture utilisées dans le jeu.

couleurs Dictionnaire contenant 4 tupple pour stocker les codes couleurs rouge, vert, bleu et blanc qui sont utilisés pour le remplissage des éléments de l'interface graphique.

sauvegarde Dictionnaire contenant les propriétés sauvegardées lors du redémarrage du jeu. On y note notamment le score record, la couleur de l'interface sélectionnée, le nombre de vies du joueur choisi, le nombre maximal d'ennemis affichés à l'écran, le facteur de vitesse choisi, mais aussi l'utilisation du fichier *debug* (voir 4.2). De plus amples détails sur la sauvegarde sont disponibles dans la partie 4.5.

1.2 Affichage des menu

À l'affichage des menu (*Menu Principal*, *Game Over* et *Options*), deux nouvelles valeurs sont utilisées et intégrées au dictionnaire `jeu`. Celles-ci sont retirées au changement du statut de jeu :

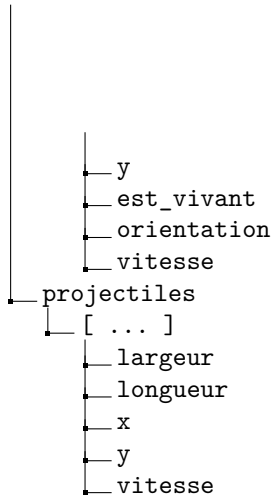
boutons Liste des boutons à afficher sur le menu.

curseur Nombre entier qui permet d'indiquer le bouton sélectionné parmi ceux de la liste précédemment mentionnée.

1.3 « Jeu en cours ! »

Pour l'affichage du joueur, des ennemis, des projectiles et du score, de nouvelles propriétés s'ajoutent aux valeurs de l'architecture principale de la variable `jeu` dont celle-ci est décrite ci-dessous. Ces propriétés sont retirées à la sortie du jeu (sauf pour le *score* qui permet d'indiquer un nouveau record dans l'écran *Game Over*) :

```
jeu
├── ...
├── score
├── joueur
│   ├── largeur
│   ├── longueur
│   ├── x
│   ├── y
│   ├── vies
│   └── est_vivant
└── ennemis
    └── [ ... ]
        ├── largeur
        ├── longueur
        └── x
```



Détail de l'architecture

score Nombre entier permettant au joueur de comptabiliser ses points.

joueur Dictionnaire comportant les propriétés du joueur : *largeur* et *longueur* désignant, en pixel, la taille du joueur et de sa boîte de collision (voir 3) ; x et y sa position dans la fenêtre bien que la propriété y ne varie pas mais nécessaire pour l'affichage de son « sprite » (de son image) ; *vies* indiquant le nombre de vies restantes au joueur ; et *est_vivant* permettant de définir l'état vivant du joueur sous forme de booléen, utilisé pour l'animation de mort de celui-ci.

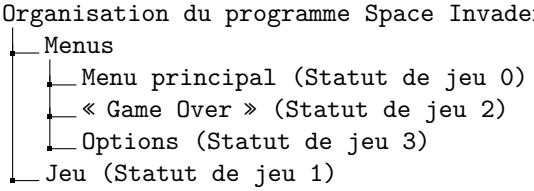
ennemis Liste comportant les dictionnaires liés aux ennemis avec les propriétés *largeur*, *longueur*, x, y, et *est_vivant* identiques au joueur ; *orientation* pour indiquer sa direction, 1 vers la droite et -1 vers la gauche ; *vitesse* définissant la vitesse de déplacement en nombre de pixel par frame. Les ennemis sont supprimés trois secondes après avoir été touché par un projectile pour laisser le temps à leur animation de s'afficher.

projectiles Liste identique à celle des ennemis à l'exception des propriétés *est_vivant* et *orientation* indisponibles parmi les dictionnaires de propriétés des projectiles. Les projectiles sont supprimés dès l'instant où ils ne sont plus visibles dans la fenêtre et où ils touchent un ennemi.

2 Optimisation de l'arborescence du programme

Dans le but de limiter le nombre de modules, je choisi d'organiser toutes les interfaces concernant les menus dans le même module en les différentiant par rapport au statut de jeu.

Cette organisation me permet d'utiliser le même type de menu dans tout le programme et me fait gagner un certain espace au détriment de la lisibilité. C'est pour cela que je déplace les parties de chaque menu dans leur nouveau fichier respectif tout en gardant la même organisation de code pour en faciliter cette même lisibilité.



3 Des boîtes rectangulaires pour les collisions

Pour vérifier si les ennemis entrent en collision avec une autre entité (peut être le joueur, un projectile voire un autre ennemi), je vérifie simplement si l'entité à comparer se situe à l'intérieur de l'un des ennemis présent dans leur liste de la variable `jeu`.

Ainsi, si la position d'un ennemi additionnée à la moitié de sa longueur se trouve également occupée par la différence de l'entité par la moitié de sa longueur, la collision a lieu. Réitérons cette vérification dans les trois autres côtés pour chacun des ennemis et une boîte de collision rectangulaire de taille *longueur* et *largeur* est en quelques sortes définie pour ces ennemis.

La traduction en code Python dans mon programme est la suivante où est retourné l'ennemi pour lequel l'entité est entré en collision si le cas en est avéré (les flèches rouges indiquent un retour à la ligne automatique) :

```
def collisions(entite, jeu):
  for ennemi in jeu["ennemis"]:
    if (ennemi["x"] + ennemi["longueur"] // 2 > entite["x"] - entite["
        ↪ longueur"] // 2 and
      ennemi["x"] - ennemi["longueur"] // 2 < entite["x"] + entite["
          ↪ longueur"] // 2 and
      ennemi["y"] + ennemi["largeur"] // 2 > entite["y"] - entite["
          ↪ largeur"] // 2 and
      ennemi["y"] - ennemi["largeur"] // 2 < entite["y"] + entite["
          ↪ largeur"] // 2 and
      entite != ennemi and ennemi["est_vivant"]):
    return ennemi
  return {}
```

4 Contenu additionnel

4.1 Menu des options

Le menu des options permet au joueur de personnaliser son expérience en modifiant directement les différents paramètres du jeu.

Celui-ci permet notamment de modifier la couleur de l'interface en rouge, vert, bleu ou blanc, de modifier le nombre de vies du joueur, le nombre maximal d'ennemis affichés en même temps à l'écran, le facteur de vitesse des ennemis et projectiles et du `debug` (voir 4.2).

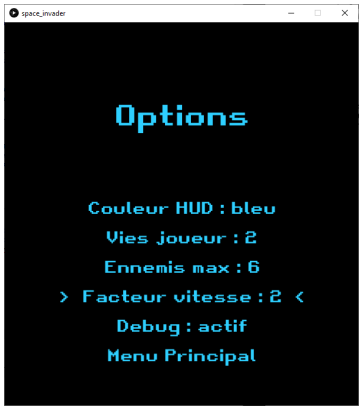


FIGURE 1 – Menu des options

4.2 Fonctionnalité de `debug`

La fonctionnalité de `debug` permet d'inscrire la variable `jeu` au format `json` dans le fichier `data/debug.json` en écrasant son contenu et à chaque frame ce qui permet de prendre conscience des propriétés du jeu à un instant donné.

4.3 Comptabilisation du score et du record

À chaque fois qu'un projectile entre en collision avec l'un des ennemis (voir 3), le joueur remporte 10 points qui lui sont ajoutés au score. À la fin de la partie, ce score est affiché dans le menu *Game Over*. Si celui-ci constitue un record, le score est alors sauvegardé dans le fichier de sauvegarde (voir 4.5) et le joueur est félicité dans ce même menu.

4.4 Vies supplémentaires

Selon les paramètres choisis, le joueur peut posséder entre 1 et 2 vies supplémentaires pour un maximum de 3 vies. Cela lui permettra, s'il se fait toucher par un ennemi, de continuer à jouer sans passer par le menu de *Game Over* et de conserver son score. Le plateau de jeu est néanmoins réinitialisé après l'animation de mort du joueur.

4.5 Sauvegarde du score record et des options

Pour qu'au redémarrage du jeu, le joueur n'ait pas à redéfinir ses options et pour que le score record en soit conservé, je choisi de mettre en place un système de sauvegarde. Rien de bien complexe néanmoins, au démarrage du jeu, si des valeurs se trouvent dans le fichier `data/sauvegarde.json`, celle-ci seront chargées dans la clé `sauvegarde` de la variable `jeu`, des valeurs par défaut le seront le cas échéant. Ces valeurs, une fois modifiées sont enregistrées dans ce même fichier au format `json` pour en faciliter la manipulation.

5 Crédits

Je me suis intégralement chargé du développement du jeu, de la conception, à la programmation, jusqu'au design des « sprites » constituant ses images en m'inspirant de jeux rétro déjà existants.