Moteurs de jeu

“VAMPIRE SURVIVORS”-LIKE 2D

GUILLON Valentin 20002588
KSAL Halima 19000939
FAKIH Cheïmâa 21001308

Sommaire

Introduction

Incarnant un personnage central à l'écran, le joueur devra faire face à un flot continu d'ennemis générés en bordures d'écran, qui le poursuivra sans répit.
Le joueur doit simplement éviter le contact avec les ennemis, tandis qu'il tire automatiquement vers eux.
Éliminer un ennemi peut faire apparaître un bonus, qui, une fois ramassé, donne un boost temporaire au joueur (tir plus rapide, projectile qui se divise à l'impact, invincibilité).

Le jeu possède un mode Démo, dans lequel le déplacement du personnage est automatisé.

1. Ressources

1.1. Sons

Bande-son créée sur BeepBox.co.

Musique :

- Menu
- Ambiance du jeu

Effets sonores :

- Bruits d'explosion
- Bruits de collision

1.2. Sprites et Tileset :

Majoritairement récupérés sur spriters-resource.com.

Tileset, tour, personnage, ennemis, projectiles, explosion, bonus

Boutons, barre des Bonus faits sur Krita.

1.3. Javascript

Librairies :

- dat.GUI
- Math (de JavaScript)

Classes de JavaScript :

- Image()
- Audio()

2. Caractéristiques du projet

Le déroulement d'une partie de jeu repose sur des instances de `My_Object`.

```
// PROPRIÉTÉS
//-- d'instance
position, vélocité, vitesse
image, hitBox
group

//-- de classe
instances //liste

// MÉTHODES
action() //déplacement, vérification de collision, auto_actions()
animate()
draw()

//-- réservé aux sous-classes
auto_actions() //suite de action()

// EFFET DE COLLISION
die()
recul(objet) // éloigne l'instance de l'objet
```

Une instance de `My_Object` est composée d'une image (`My_Img` ou `My_Animated_Img`) et

d'une hitbox (`HitBox_Mask`).
Il est possible qu'elle n'ait qu'un seul, ou aucun de ces deux éléments.

`My_Object` possède une propriété importante nommée "group".
Cette propriété est observée lors d'une collision, afin de décider comment affecter les deux instances concernées.

`My_Object` décrit le comportement commun (déplacement et collision) à toutes les sous-classes.

Ses méthodes principales sont `action()`, `animate()` et `draw()` :

- `action()` est fragmentée en 3 parties:
    - `update_status()`, met à jour l'état de vie de l'instance
    - `check_collisions(...)`, compare la hitBox de l'instance avec celle de tous les autres.
    - `auto_actions()`, réservée aux sous-classes, décrit le comportement de celles-ci.
- `animate()` permet aux images animées de passer à la frame suivante.
- `draw()` dessine l'image sur le canvas

2.1. Collisions

Lors de la création d'une instance de `My_Object`, une instance de `hitBox` y est associée.

Lors d'une collision établit entre deux instances de `My_Object`, la propriété `group` de chacune est vérifiée, et un effet leur est octroyé en conséquence.

Les effets sont nombreux (mort, gain de bonus, "recul").
Certains sont définis dans la classe My_Object, d'autres, dans des sous-classes.
Le "recul" affecte certaines sous-classes de `My_Object`, lors d'une collision avec une instance de `My_Object` de "group" "obstacle".
Il consiste à éloigner l'objet de l'obstacle d'un pixel, tant que ceux-ci sont toujours en collision.

Nous utilisons le système “pixel perfect” (ou “mask”), défini dans la classe `HitBox_Mask`.

La zone de collision est définie par une liste de booléen, correspondant à chaque pixel d’un mask.

Un mask est une image noire et transparente, où chaque pixel visible correspond à une partie de la hitBox.

Figure 1 - Mask tour et sprite de base

Lors de l’initialisation, l’image mask est dessinée sur le Canvas, puis on parcourt la zone dessinée. Pour chaque pixel noir ou transparent, on rajoute un booléen dans la liste liéée à la hitBox.
Cette liste sera analysée lorsqu’il faudra vérifier les collisions entre deux instances de My_Object.

2.2. Entités animées

2.2.1. Joueur

Les classes `Player` et `Player_Auto` sont des sous-classes de `My_Object`. Il est possible de déplacer le personnage du joueur à l’aide des touches Z,Q,S,D dans toutes les directions (y compris en diagonale en maintenant deux touches à la fois). Ce dernier peut ramasser des bonus, entrer en collision avec les obstacles, et disparaître au contact avec l’ennemi, soit un Game Over, sauf en cas de bonus d’invincibilité.

Dans le mode démo, le personnage s’éloigne automatiquement de l’entité “dangereuse” la plus proche (les ennemis, les obstacles), tout en se rapprochant de l’entité “bénéfique”

(bonus) la plus proche.
Lorsque le personnage est sous l'effet du bonus d"invincibilité", les ennemis ne sont plus considérés comme "dangereux", mais "bénéfiques".

2.2.2. Ennemis

Enemy Chasing :

Définit le comportement des ennemis qui poursuivent activement le joueur. L'objectif de la classe `Enemy_Chasing` est d'ajuster la direction de l'ennemi pour qu'il se déplace vers le joueur.

Les ennemis sont générés en continu sur les bordures de l'écran, via la classe `Enemy_Generator`, sous-classe de `My_Object`.

2.2.3. Projectiles

Les projectiles dans le jeu sont conçus pour recevoir une direction à leur création, qui ne change pas jusqu'à une collision. Les projectiles ne sortent pas de l'écran. Il y a les projectiles lancés par les tours ennemies (générées à la mort d'ennemis) et ceux qui sont lancés par les alliés.

2.3. Entités statiques

Ces entités (obstacles, bonus collectibles…) sont des sous-classes de `My_Object`.
Elles n'ont aucun comportement, mais influence les autres "group" d'instances qui entrent en collision avec (mort, gain de bonus, recul) grâce à la propriété "type".

3. Autres

3.1. Sous-classes additionnelles

Il existe d'autres sous-classes de `My_object` qui n'ont pas de hitBox (et parfois pas d'image). Elles sont enfants de `My_Object` afin de faciliter leur suppressions lors d'un changement de menu.

Les `Timer` exécutent une certaine action à la fin d'une durée donnée.
Par exemple, lancer le mode Démo, ou quitter la page de Game-Over.

`Moving_Background` déplace lentement une image de droite à gauche.

3.2. Audio

La classe `Jukebox` permet de jouer musiques et effets sonores.

3.3. Caméra

La classe `Camera` donne l'illusion d'une caméra fixée sur le joueur.

À chaque rafraîchissement du jeu, une instance de `My_Object` est donnée à l'instance unique de `Camera`.
Celle-ci va déplacer toutes les instances de `My_Object` selon un même vecteur, de sorte que l'objet donné voit ses coordonnées correspondre avec le centre du Canvas.

3.4. Génération de la carte

La carte est découpée en morceaux provenant d'un tileset.

Lors du lancement de la partie, la carte est généré selon une carte pré-écrite, définie par une liste de chiffre (où par exemple, 0 est un sol, 1 est un obstacle…).

Si le chiffre correspond à une partie pouvant être représentée par des images différentes (les morceaux d'un obstacle; un coin, un côté, un virage…), les chiffres adjacents sont analysés afin de choisir la "tile" adéquate.

3.5. dat.gui

Nous l'utilisons à fins de tests et de debug.
Il nous permet par exemple de visualiser les collisions, mettre le jeu en pause, agir sur les propriétés du joueur (vitesse, cadence de tir, bonus, kill) ou faire apparaître des entités là où on clique.

Géré uniquement dans script.js, nous avons créé un dictionnaire qui contient une entrée pour chaque interaction.
Certaines de ces entrées sont vérifiées lors de la mise à jour du jeu ("game freezed", "show image", "collision enabled", "show hitBox").
Une interaction peut parfois agir directement sur une instance de MyObject (pour arrêter le tir, ou donner un bonus).