Compte rendu - Space Invader

Charlotte Menou (groupe info1) - Ange Leyrit (groupe info2)

Pour ce projet, nous avons réalisé un jeu avec un defender qui tire des bullets sur une flotte d'aliens. Le but pour le defender étant de détruire toute la flotte d'aliens avant qu'elle ne l'atteigne. Pour ce faire, on a utilisé la programmation objet, en définissant différentes classes.

Structure et fonctionnement des classes

Classe Alien

Dans la classe Alien, on gère l'affichage d'un alien grâce à une méthode de classe et une fonction *install_in* permettant de récupérer l'image d'un alien et de l'afficher dans le canvas. On y définit une fonction *move_in* qui gère le déplacement d'un alien.
La fonction *touched_by* gère la "mort" d'un alien. Quand cette fonction est appelée, l'alien est supprimé du canvas et la variable de classe self.alive est mise à "false".

Classe Fleet

La classe Fleet est une sous-classe de la classe Alien qui permet de gérer une flotte d'aliens.
La fonction *install_in* permet d'afficher une flotte d'aliens sur le canvas. Elle utilise la fonction install_in de la classe Alien pour afficher un par un tous les aliens de la flotte.
La fonction *move_in* permet de déplacer la flotte d'aliens sur le canvas. Elle utilise la fonction move_in de la classe Alien pour déplacer un par un tous les aliens de la flotte. Lorsque la flotte atteint un des bords du canvas, la flotte descend et repart dans l'autre sens.
La fonction *managed_touched_by_aliens* permet de gérer les collisions bullet-alien. Grâce à la fonction find_overlapping, on récupère toutes les coordonnées des points qui forment un alien. Si un des ces points a des coordonnées communes avec un autre objet, on regarde si c'est un bullet, et si s'en est un, on supprime le bullet, et on fait appel à la fonction *touched-by* de la classe alien pour
supprimer l'alien.

Classe Defender

La classe Defender est une sous-classe de la classe Bullet et permet de gérer le defender mais également le tir des bullets. La fonction *install_in* permet de créer le defender dans le canvas. La fonction *move_in* gère le déplacement du defender. La fonction *fire* permet de créer une instance de bullet. Quand elle est appellée, elle ajoute cette instance de bullet dans la variable de classe *self.fired_bullets* qui est une liste contenant tous les bullets présents dans le canvas. Puis elle fait appel à la fonction *install_in* de la classe Bullet pour créer le bullet dans le canvas. La variable de classe *self.max_fired_bullets* contient le nombre maximum du bullets présents en même temps dans le canvas.

Classe Bullet
La classe Bullet permet de gérer les bullets.
La fonction *install_in* permet de créer une instance de Bullet juste au dessus de l'emplacement du defender passé en paramètre de la classe Bullet.
La fonction *move_in* permet de déplacer le bullet vers le haut de l'écran.

Classe Game
La classe Game permet de mettre en place les différents éléments du jeu dans un canvas. C'est une sous-classe de la classe Fleet et de la classe Defender.
Dans la fonction *__init__* on y définit donc un canvas. De plus, on applique la fonction *install_in* de la classe Fleet sur une instance de Fleet et la fonction *install_in* de la classe Defender sur une instance de Defender pour installer la flotte d'alien et le defender dans le canvas.
La fonction *keypress* permet de gérer les touches clavier pour le defender. Si la touche flèche gauche est activée le defender va se décaler vers la gauche, si la touche flèche droite est activée le defender va se décaler vers la droite et si la touche espace est activée une instance de la classe Bullet va être lancée juste au dessus du defender.
La fonction *move_bullets* permet de gérer le déplacement des bullets. Si l'ordonnée d'un bullet dépasse le haut du canvas, le bullet sera supprimé de la liste des bullets et on rechargera d'un coup le nombre de bullets que peut lancer le defender.
La fonction *move_aliens_fleet* permet le déplacement de la flotte d'aliens.
Les fonctions *game_over* et *winner* sont des fonctions d'affichage à la fin de la partie en cas de victoire ou de défaite du joueur. Elle permettent d'afficher "You win" ou "Game Over" ainsi que le nom du joueur passé en variable de la classe Game et le score du joueur , score qui sera calculé en fonction du nombre d'alien détruit par le defender, à hauteur de un point par alien détruit. L'incrément du score est fait dans la fonction *manage_touched_by* définie dans la classe Fleet. La fonction *animation* est une fonction qui se répète. On y appelle les fonctions de mouvement tel que *move_bullets et move_aliens_fleet* pour animer le jeu. On y appelle également la fonction *manage_touched_aliens_by* de la classe Fleet pour gérer les éventuelles collision entre les aliens et les bullets. Si il reste des aliens sur le canvas et que le bas de la flotte d'aliens est plus basse que le defender on appelle la fonction *game_over* car la partie est finie et le jeu est perdu. Au contraire, si le nombre d'aliens restant est égal à 0, on appelle la fonction *winner* car la partie est finie et le jeu est gagné. Pour finir, la fonction *start_animation* permet d'activer la fonction animation.

Classe SpaceInvader
La classe SpaceInvader permet de démarer le jeu. Elle définit une frame et fait appel à la fonction *start_animation* de la classe Game pour commencer le jeu.

Problèmes rencontrés:

On a eu du mal à comprendre comment gérer la liste de bullets. Au début, au lieu de supprimer les bullets de la liste, on mettait sa case dans la liste à None. Mais c'était une mauvaise idée car on avait des problèmes dès qu'on voulait faire une boucle sur cette liste. Les objets de type None n'étant pas itérables.

Dans la classe Fleet, si un Alien était détruit, lorsqu'on parcourait la liste de tous les aliens, on se retrouvait avec des alien.id de type None. Dans la fonction *manage_touched_aliens_by*, on a dû installer une pré-condition pour vérifier si l'Alien était encore vivant avant de lancer la récupération des coordonnées de l'alien. En effet, on ne peut pas récupérer les coordonnées d'un objet de type None.

Pour définir les bords de la flotte d'aliens on a d'abord pris les coordonnées de deux aliens de chaque côté de la flotte. Dès qu'un de ces deux aliens touchaient un bord du canvas on repartait de l'autre côté. Mais lorsque ses aliens étaient touchés par un bullet ils étaient détruits, donc on ne pouvait plus récupérer leurs coordonnées et le jeu crachait. On a donc compté le nombre de déplacements nécessaires de la flotte pour aller d'un bord à l'autre du canvas.

On a une version du jeu avec des explosions lors de collisions bullet/alien, mais elle n'est pas aboutie. En effet, on a pas réussi à supprimer l'explosion du canvas après quelques millisecondes. On a essayé avec l'instruction "*canvas.after(1000, delete(explosion))*", mais l'explosion ne s'affichait pas du tout et le jeu entier se mettait en pause pendant une seconde. On n'a donc pas trouvé d'autre solution.

On a mis du temps à bien comprendre le fonctionnement de la fonction *find_overlapping()*.

Ainsi, dans notre jeu, nous avons un defender qui est capable de tirer des bullet, une flotte d'aliens, et nous avons géré la collision bullet/alien avec des explosions (version pas aboutie). Puis nous avons un affichage de fin de jeu qui comprend "game over" ou "You win",le nom du joueur, et le score du joueur. Nous aurions pu améliorer notre version en en faisant une page de démarrage du jeu avec la sauvegarde de tous les précédents scores des autres joueurs, et un bouton "start" pour démarrer le jeu. Nous aurions également pu ajouter des améliorations dans le design du jeu, comme des aliens de différentes formes, des bullets pour les aliens, ou encore des bunkers pour que le defender puisse s'abriter et des niveaux de difficultés(Alien se déplaçant plus vite, moins de Bullet en simultanés ect…)
Pour ce qui est d'une implémentation IA dans le jeux, il pourrait être intéressant de faire calculer à une IA, premièrement la position à laquelle le joueur à le plus tiré de bullet, et deuxièmement le nombre de déplacements moyen du defender vers la droite et vers la gauche. Ainsi cette implémentation pourrait ajouter une difficulté au joueur, en effet le jeu serait capable entre chaque niveau de déterminer une zone de probabilités de présence du joueur.