Projet Space Invaders

SCOUARNEC Josselin
L2 Info

Avril 2022

Table des matières

1 Introduction

1.1 Amérliorations apportéss

— Explosions
— Tir et délai de rechargement
— Audio
— Scores
— Classe Canvas
— Classe GameObject

1.2 Problèmes et améliorations possibles

Redondance dans le code

Le code pour le tir de projectiles est presque le même pour `Defender` et `Fleet`. Pour résoudre le problème on pourrait par exemple les faire hériter d'une classe `Tireur`.

Taille de la fenêtre

Le canvas Tkinter ne permet pas de redimensionner les images directement, il est donc difficile de supporter plusieurs tailles de fenêtre sans que cela n'impacte le jeu. Par exemple avec une fenêtre plus haute les aliens mettraient plus de temps atteindre le bas de l'écran, rendant le jeu plus facile. En effet Tkinter n'est pas une bibliothèque graphique mais une bibliothèque d'interface utilisateur.

Audio portable

pas d'audio facilement portable sans utiliser un module tier.

Calcul du score

Il est assez facile d'obtenir le score maximal qui est de 16000 points en tuant tous les aliens. Il faudrait ajouter des paramètres supplémentaires au calcul des points comme par exemple des bonus et des malus pour les vies qu'il restantes, le nombre de balles ratées, la hauteur des aliens...

Animations

Les objets visibles ne disposent que d'une seule frame d'animation, c'est flagrant dans le cas des explosions qui

Temps d'invincibilité

Le jeu n'est pas très permissif : il est possible de se faire toucher par deux projectiles dans un court intervalle. Un attribut pourrait être ajouté à `Defender` pour indiquer quand il a été touché pour la dernière fois et le code qui gère les collisions pourrait être modifié en conséquence.

1.3 Intelligence Artificielle ?

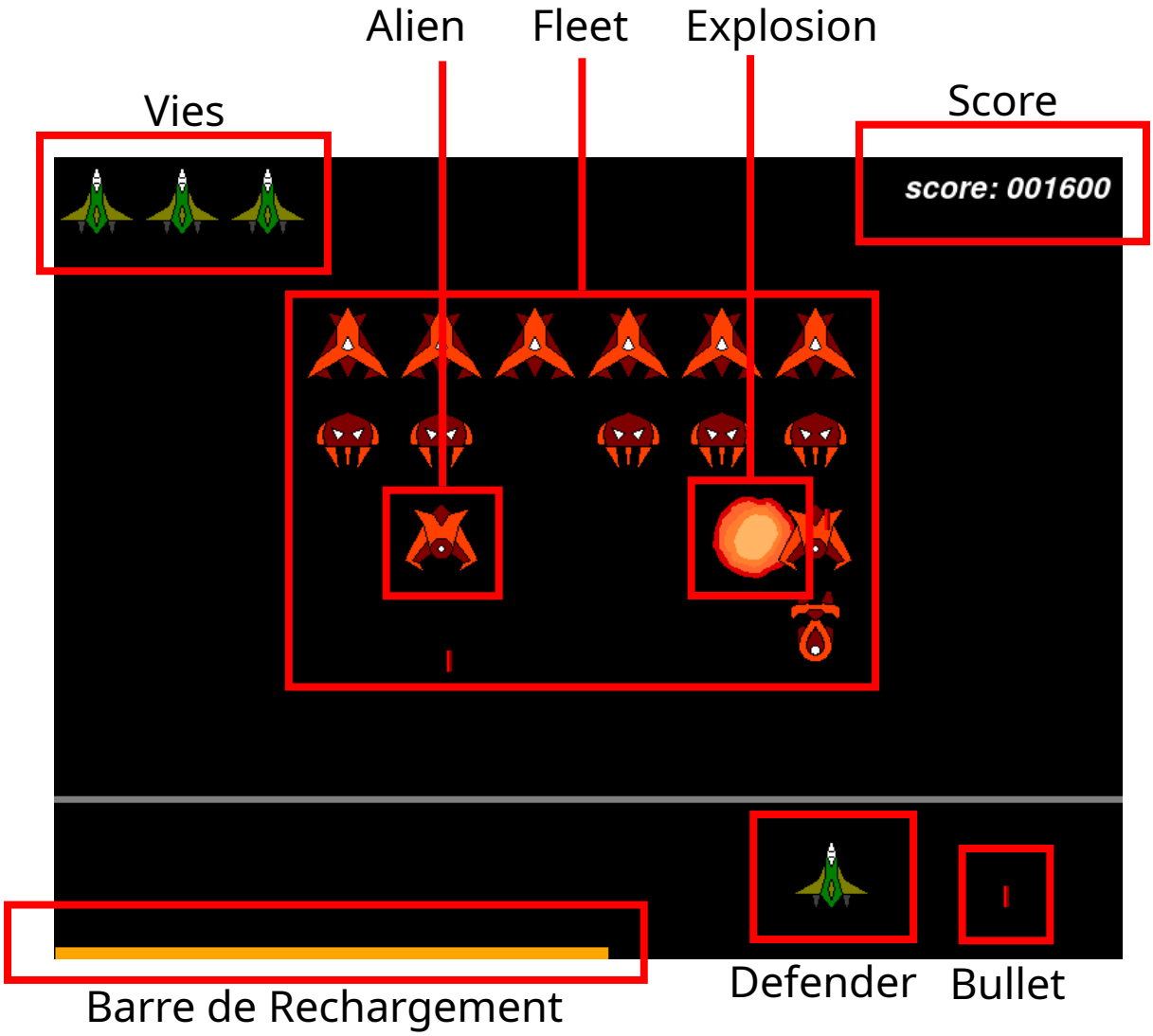
Dans l'état actuel, la flotte alien tire à la verticale depuis la position d'un alien choisi aléatoirement. Pour rendre le jeu plus intéressant on pourrait remplacer l'aléatoire par une fonction qui détermine l'alien qui est dans la meilleure position pour toucher le joueur.

Notons v_P la vitesse des projectiles, v_D la vitesse du defender, x_D son abscisse et (x, y) la position de l'alien qui tire. On prend comme origine la position du defender. Le temps nécessaire pour que le projectile l'atteigne est $t = \frac{y}{v_P}$. Dans l'hypothèse où le defender a gardé sa vitesse constante et que le projectile a été tiré depuis une position optimale, au temps t le defender est touché. Comme le projectile a gardé la même abscisse, on a $x = x'_D = x_D + tv_D$.

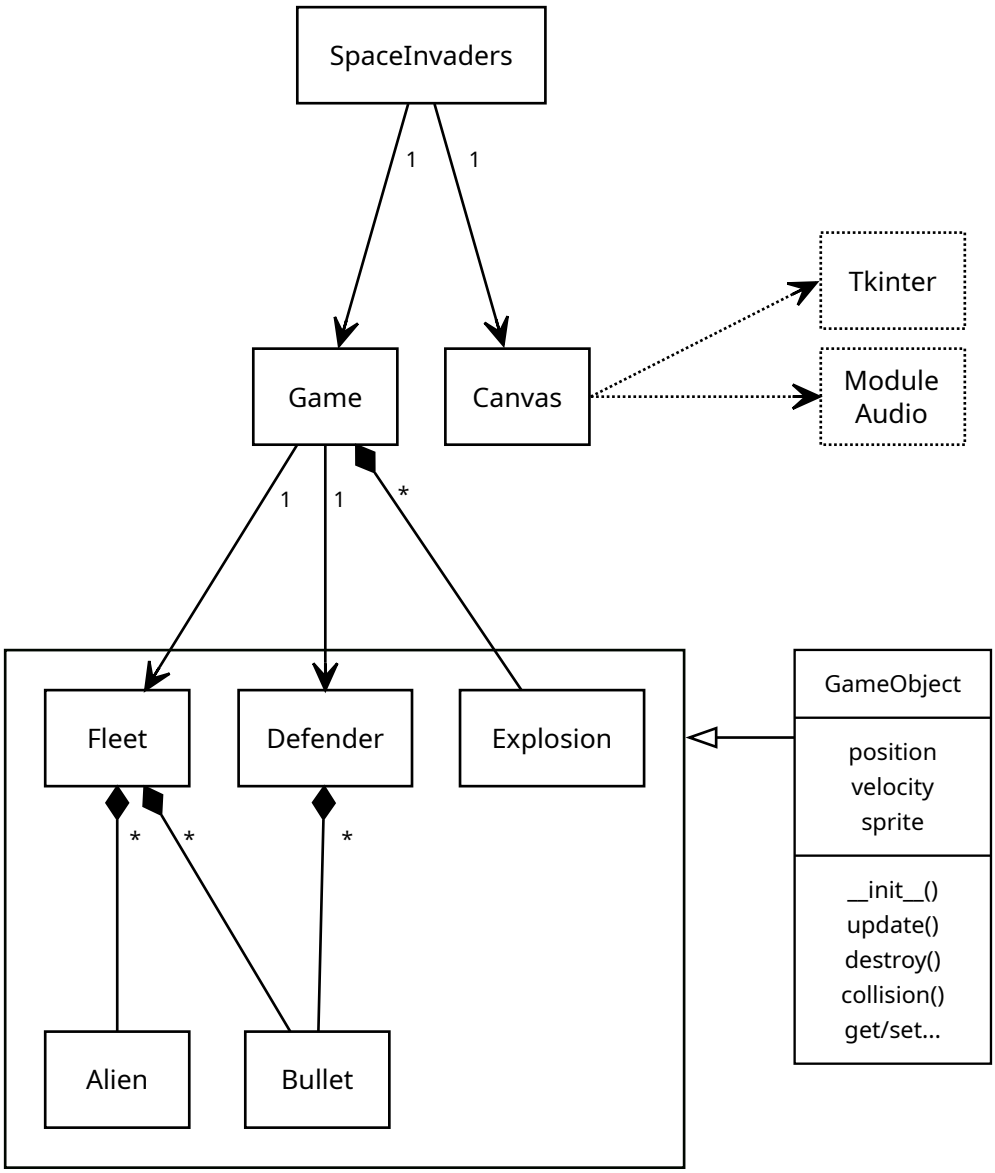
Il faut donc trouver un alien qui se rapproche d'une solution de l'équation linéaire ci-dessous, cependant rien n'empêche le joueur d'esquiver le projectile.

$$\boxed{x = x_D + y\frac{v_D}{v_P}}$$

2 Écran de jeu



3 Architecture



Les classes `Fleet, Defender, Explosion...` héritent de `GameObject`

4 Graphismes

Sprites pour les 4 types d'alien, les explosions, le joueur et les projectiles.

5 Conditions de game over

Il y a trois manière des finir la partie :
— Un des aliens a franchi une certaine ordonnée (`"finish_line"`)
— Le joueur a perdu toutes ses vies
— Tous les aliens ont été détruits

```
# Game.update()
if any(a.y + a.height >= self.finish_line for a in self.fleet) :
    self.transition_game_over()

elif not self.player.is_alive() :
    self.transition_game_over()

elif len(self.fleet) == 0 :
    self.add_score(10000)
    self.transition_game_over(win=True)
```

6 Gestion des collisions

6.1 Détection

On peut détecter de manière triviale si un objet sort de l'écran en connaissant ses coordonnées, sa taille et la taille de l'écran :

```
# GameObject
def screen_collision(obj) :
    return any([
        obj.x < 0, # left
        obj.y < 0, # top
        obj.x + obj.width  >= obj.canvas.get_width(), # right
        obj.y + obj.height >= obj.canvas.get_height() # bottom
    ])
```

Pour qu'il y ait collision entre deux objets `o1` et `o2`, il faut réunir 4 conditions :
- Le bord gauche de `o1` est à droite du bord gauche de `o2`
- Le bord droit de `o1` est à gauche du bord droit de `o2`
- Le bord haut de `o1` est plus bas que que le bord haut de `o2`
- Le bord bas de `o1` est plus haut que que le bord haut de `o2`

```
# GameObject
def collision(o1, o2) :
    return all([
        o1.x + o1.width >= o2.x,  # left
        o1.x <= o2.x + o2.width,  # right
        o1.y + o1.height >= o2.y, # top
        o1.y <= o2.y + o2.height, # bottom
    ])
```

La classe `Fleet` utilise une variante de ces fonctions puisque détecter les collisions avec un rectangle qui englobe tous les aliens serait peu convainquant. Plutôt, on teste si au moins un des aliens est en collision avec l'écran ou un objet.

```
# Fleet
def screen_collision(fleet) :
    return any(a.screen_collision() for a in fleet)

def collision(fleet, obj) :
    return any(a.collision(obj) for a in fleet)
```

6.2 Collisions `Fleet-Écran`

Lorsque la flotte détecte qu'il y a collision avec un bord de l'écran, sa vitesse horizontale est multipliée par l'opposé de son accélération : cela correspond à changer de sens et à augmenter la vitesse absolue. La vitesse de tous les aliens est alors mise à cette valeur et leur hauteur est incrémentée.

```
# Fleet.update()
if self.screen_collision() :
    self.vx *= -self.acc

    for alien in self :
        alien.set_vx(self.vx)
        alien.add_y(self.vy)
```

6.3 Collisions `Defender-Écran`

Pour éviter que le joueur ne puisse sortir de l'écran, il n'est possible pour lui de se déplacer d'un côté que si il n'a pas déjà dépassé le bord de l'écran de ce côté. Ainsi, même si il se retrouvait en dehors de la zone de jeu (impossible), il pourrait toujours revenir.

```
# Defender.update()
if self.keys["left"] and self.x > 0 :
    ...

if self.keys["right"] and self.x + self.width < self.canvas.get_width()
↪   :
    ...
```

6.4 Collisions `Bullet-Defender`

Gérer les collisions entre le joueur et les projectiles de la flotte se fait en deux étapes. D'abord ces projectiles sont recueillis dans une liste. Ensuite pour chacun d'entre eux une explosion est créée à la position du joueur, un message lui est envoyé pour qu'une vie soit retirée et le projectile est supprimé.

```
# Game.update()
collisions = [b
    for b in self.fleet.bullets
    if self.player.collision(b)
]

for b in collisions :
    self.boom(self.player.get_x(), self.player.get_y())
    self.player.hit()
    self.fleet.remove_bullet(b)
```

6.5 Collisions `Bullet-Fleet/Alien`

Pour les collisions entre les aliens et les projectiles du joueur, on trouve d'abord les couples `(Alien, Bullet)` en collision.

```
# Game.update()
collisions = [(a, b)
    for a in self.fleet
    for b in self.player.bullets
    if a.collision(b)
]
```

Les projectiles impliqués sont ensuite supprimés. Un projectile peut potentiellement être en collision avec deux aliens, pour ne pas le supprimer deux fois on itère sur l'ensemble (« `{}` ») des projectiles présents dans la liste des collisions.

```
# Game.update()
for b in {b for _, b in collisions} :
    self.player.remove_bullet(b)
```

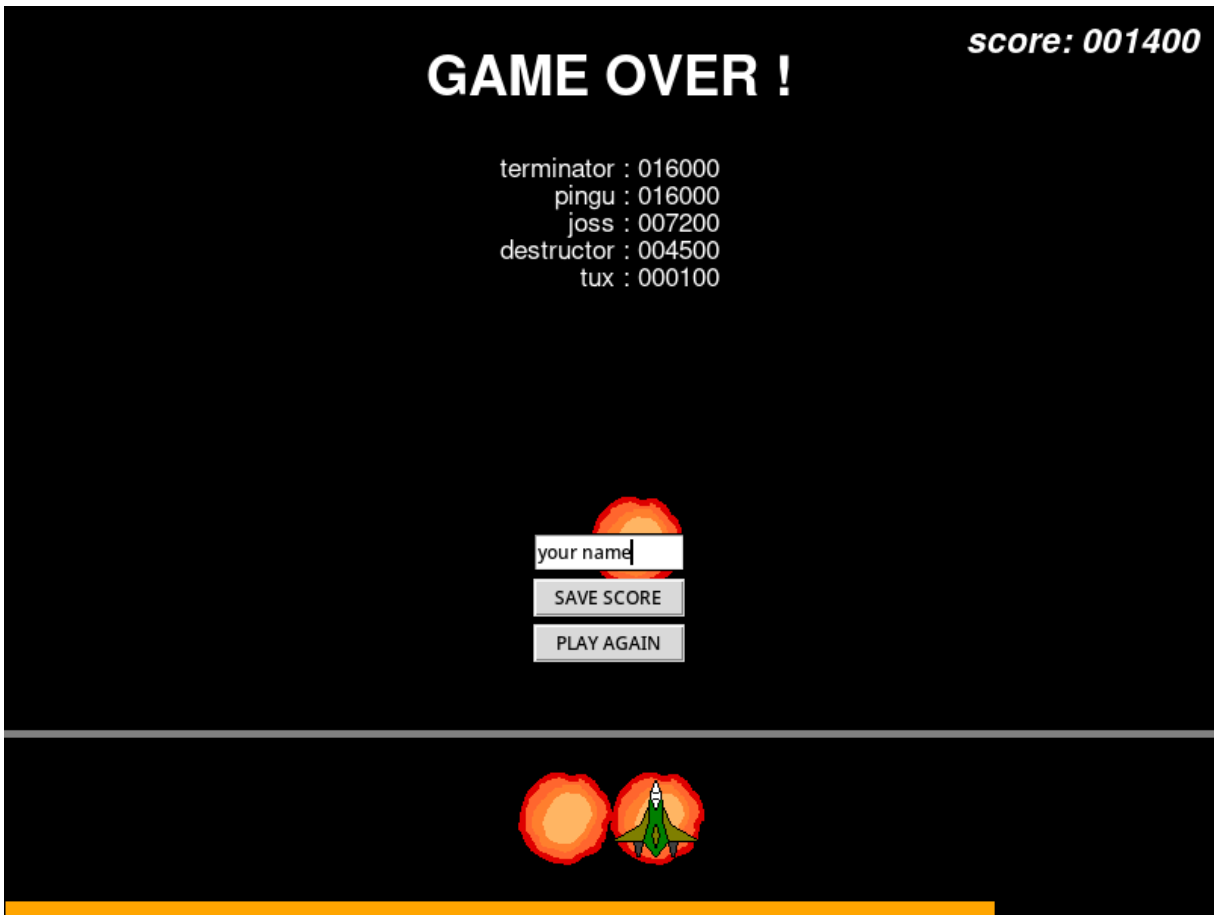
De la même manière, pour chacun de ces aliens, le score est incrémenté, une explosion est créée et l'alien est détruit.

```
# Game.update()
for a in {a for a, _ in collisions} :
    self.add_score(a.get_value())
    self.boom(a.get_x(), a.get_y())
    self.fleet.remove(a)
```

7 Gestion des scores

7.1 Interface utilisateur

Le score courant est indiqué en haut à gauche de l'écran. Une fois la partie terminée, les scores sont affichés dans l'ordre. Le joueur peut alors rajouter le sien si il n'a pas déjà eu un score plus élevé. Le formulaire est ensuite grisé pour empêcher l'ajout de scores supplémentaires.



7.2 Calcul des points

Chaque alien tué rapporte un nombre de points suivant une formule plus ou moins arbitraire : les aliens les plus bas (type 0) rapportent 100 points, ceux de la deuxième ligne en partant du bas (type 1) 200 points, etc... De plus, remporter la partie donne un bonus de 10000 points.

```
def get_value(alien) :
    return 100 * (alien.type+1)
```

7.3 Stockage

Les meilleurs scores sont enregistrés dans le fichier `scores` au format CSV, associant le nom des joueurs à leur score.

```
player,score
destructor,4500
joss,7200
terminator,16000
tux,100
pingu,16000
```

8 Module audio

8.1 Portabilité ?

Le module défini une fonction `play_wav(fname)` qui joue de manière asynchrone le fichier audio `fname` et `stop_all()` qui arrête tous les sons en cours. Il n'y a pas de module Python par défaut qui permet de jouer d'audio de manière portable, il faut donc séparer les implémentations, comme seule l'implémentation Linux a pu être testée, celle pour Windows est laissée en commentaire dans le code.

```
# audio.py
import platform
os = platform.system()

if os == "Linux" :
    def play_wav(fname) :
        ...

    def stop_all() :
        ...

else : # Non supporté
    def play_wav(fname) :
        pass

    def stop_all() :
        pass
```

8.2 Implémentation Linux

Cette implémentation utilise le module python `subprocess` et la commande `ffplay` (qui est généralement livré avec `ffmpeg`). Un processus est créé pour jouer le fichier audio et est ajouté à une liste de processus. Les sorties sont redirigées pour éviter les messages superflus.

```
def play_wav(fname) :
    p = subprocess.Popen(
        ['ffplay', '-nodisp', '-autoexit', '-volume', '75', fname],
        stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL,
    )
    processes.append(p)
```

Pour arrêter tous les sons il suffit de vider la liste en terminant chaque processus.

```
def stop_all() :
    while processes :
        processes.pop().kill()
```

9 Classe SpaceInvaders

Cette classe correspond au point d'entrée de l'application. À l'initialisation la fenêtre est créée (une instance de `Canvas`) et une partie est initialisée (instanciation de `Game`). La méthode `start` lance la boucle d'événements de la fenêtre (gérée par Tkinter) et en parallèle la boucle principale de jeu qui prend la forme d'appels successifs à la méthode `frame_event`.

Pour avoir la même expérience de jeu même avec un ordinateur très lent, au lieu de mettre un temps fixe de 1/60 de seconde, le temps écoulé depuis la dernière itération est calculé avec le module `time`. Le jeu est ensuite mis à jour avec cette valeur.

```
def frame_event(self) :
    now = time.time()
    dt = now - self.lastTime
    self.lastTime = now

    self.game.update(dt)
    ...
```

Si la partie est terminé, c'est à dire le joueur a atteint l'écran game over et a cliqué sur rejouer, alors la fenêtre est nettoyée et une nouvelle partie est initialisée.

```
    ...
    if self.game.is_done() :
        self.game.destroy()
        self.game = Game(self.canvas)
    ...
```

Enfin, si l'application est toujours active, un nouvel appel à `frame_event` est programmé pour dans 1/60 de seconde. Pour plus de fluidité il faudrait mesurer et soustraire le temps d'exécution de `game.update`. Sinon, la fenêtre est détruite si elle ne l'a pas déjà été.

```
    ...
    if self.running :
        self.canvas.after(1000 // 60, self.frame_event)

    else :
        self.canvas.destroy()
```

Pour arrêté la boucle depuis l'extérieur on utilise la méthode `stop`. D'ailleurs, le canvas est configuré dans `SpaceInvaders.__init__` pour appeler cette méthode lorsque la fenêtre est détruite ou bien lorsque la touche échap est pressée.

```
def stop(self) :
    self.running = False
```

10 Classe Canvas

Cette classe permet de construire la zone graphique en héritant de `tkinter.Canvas` et en instanciant `tkinter.Frame` et `tkinter.Canvas`.

```
def __init__(self, width, height, title) :
    super().__init__() # tk.Tk

    self.title(title)
    self.width  = width
    self.height = height

    self.tkframe  = tk.Frame(self)
    self.tkcanvas = tk.Canvas(self.tkframe, width=self.width,
    ↪  height=self.height, ...)
    ...
```

Une instance de cette classe peut être manipulée à la fois comme `tkinter.Tk` par héritage et comme `tkinter.Canvas` par un ensemble d'attributs-méthodes. De la même manière un accès au module audio (global) est défini.

```
    ...
    self.create_image     = self.tkcanvas.create_image
    self.create_rectangle = self.tkcanvas.create_rectangle
    self.create_line      = self.tkcanvas.create_line
    self.create_text      = self.tkcanvas.create_text
    ...
    self.play_wav = audio.play_wav
    self.stop_all = audio.stop_all
```

11 Classe GameObject

Classe générique pour les objets qui ont une position, une vitesse et un sprite. Ces attributs peuvent êtres initialisés avec les paramètres optionnels du constructeur. Les membres `width` et `height` sont déduits des dimensions de l'image si elle n'est pas `None` (0 dans ce cas).

```
def __init__(self, canvas, x=0, y=0, vx=0, vy=0, image=None) :
    ...
```

La méthode `update` permet de calculer la nouvelle position de l'objet en connaissant sa vitesse et le temps écoulé depuis la dernière mise à jour (intégration par méthode d'Euler). Une classe héritant de `GameObject` pourra redéfinir `update`.

```
def update(self, dt) :
    self.x += self.vx * dt
    self.y += self.vy * dt
    self.update_sprite()
```

`update_sprite` utilise `moveto` pour déplacer l'image à l'écran à la bonne position.

```
def update_sprite(self) :
    if self.sprite is not None :
        self.canvas.moveto(self.sprite, int(self.x), int(self.y))
```

Enfin, une méthode pour retirer le sprite de l'écran lorsque l'on veut détruire l'objet.

```
def destroy(self) :
    self.canvas.delete(self.sprite)
    self.sprite = None
```

Les méthodes de détection de collisions ont déjà été décrites plus haut.

```
def collision(o1, o2) :
    ...

def screen_collision(self) :
    ...
```

12 Classe Defender

12.1 Contrôles

Tkinter permet d'appeler une fonction quand une touche est pressée. Pour savoir si une touche est actuellement enfoncée ou non, les méthodes `key_up` et `key_down` de `Defender` sont configurées pour gérer les événements clavier.

```
def key_down(self, e) :
    if e.keysym == "space" :
        self.keys["fire"] = True
    elif ...

def key_up(self, e) :
    if e.keysym == "space" :
        self.keys["fire"] = False
    elif ...
```

Ainsi on peut savoir si on doit déplacer le defender ver la droite ou vers la gauche.

```
if self.keys["left"] and self.x > 0 :
    self.x -= self.vx * dt

if self.keys["right"] and self.x+self.width < self.canvas.get_width() :
    self.x += self.vx * dt
```

12.2 Tir

À chaque mise à jour un timer est décrémenté. Lorsqu'il atteint 0 et que la touche de tir est enfoncée, il est réinitialisé et la méthode `fire` est appelée. Cette dernière ajoute une instance de `Bullet` dans la liste des projectiles tirés. La valeur du timer est utilisée pour calculer la largeur de la barre de rechargement en bas de l'écran.

```
self.timer -= dt
    if self.keys["fire"] and self.timer <= 0 :
        self.timer = self.interval
        self.fire()
```

```
def fire(self) :
    x = self.get_x() + self.get_width() / 2
    y = self.get_y() - self.get_height()
    vy = -350

    self.bullets.append(
        Bullet(self.canvas, x, y, vy, dosfx=True))
```

Lorsque les projectiles de `Defender` sont mis à jour, ceux qui sont sorti de l'écran sont retiré du canvas et de la liste avec la méthode `remove_bullet`.

```
for b in self.bullets :
    b.update(dt)

    if b.screen_collision() :
        self.remove_bullet(b)
```

12.3 Vies

Les icônes qui montrent le nombre de vies à l'écran sont stockées dans une liste. Le nombre de vies est la longueur de cette liste. Lorsque que le defender se fait toucher et que la liste est vide, son membre `alive` est mis à `False`, sinon une vie est retiré de la liste et du canvas.

```
def hit(self) :
    if self.lives :
        L = self.lives.pop()
        self.canvas.delete(L)

    else :
        self.alive = False
```

13 Classe Fleet

13.1 Initialisation des aliens

Les aliens sont stockés dans une liste unidimensionnelle initialisée dans le constructeur. Les deux `for` permettent de placer les aliens dans une formation rectangulaire avec un certain un nombre de lignes et de colonnes.

```
self.aliens = [
    Alien(self.canvas,
        x=i * self.spacing,
        y=j * self.spacing + 64,
        vx=self.vx,
        type=self.rows-j-1 # weakest at the bottom
    )
    for i in range(self.cols)
    for j in range(self.rows)
]
```

13.2 Tir

Comme pour `Defender` à chaque mise à jour un timer est décrémenté. Lorsqu'il atteint 0 il est réinitialisé et la méthode `fire` est appelée.

```
self.timer -= dt
    if self.timer <= 0 :
        self.timer = self.interval
        self.fire()
```

Lorsque l'ordre de tirer est donné, un alien est désigné au hasard comme tireur et un projectile est créé à sa position. Puis comme pour `Defender`, le projectile est supprimé en atteignant le bord de l'écran.

```
def fire(self) :
    a = random.choice(self.aliens)

    x = a.get_x() + a.get_width() / 2
    y = a.get_y() + a.get_height()
    vy = 350

    self.bullets.append(
        Bullet(self.canvas, x, y, vy, dosfx=False))
```

13.3 Itération

La méthode `__iter__` est définie pour permettre d'itérer sur `Fleet` comme une collection d'aliens.

```
def __iter__(self) :
    return iter(self.aliens)

...

fleet = Fleet()
for alien in fleet :
    ...
```

14 Remarques sur les autres classes

14.1 Game

Cette classe gère la logique principale du jeu avec les différents éléments vus précédemment : gestion des scores, résolution des collisions `Bullet-Fleet` et `Bullet-Defender`, et détection conditions de game over. Lorsqu'il y a game over, d'abord pour plus de clarté tous les sons sont stoppés et les aliens sont cachés. Ensuite le texte « You win » ou « Game over » avec le tableau des highscores et le formulaire sont affichés pendant qu'un son de victoire ou de défaites est joué.

Lorsque sa méthode `update` est appelée, les méthodes `update` respectives de ses instances de `Defender`, `Fleet` et `Explosion` sont appelée.

14.2 Explosion

Lorsqu'une explosion est instanciée, un son parmi les 5 possibles est joué. Un timer est initialisé et est décrémenté à chaque update. Une fois que le timer atteint 0, le sprite de l'explosion est retirée du canvas et son membre `alive` est mis à `False`. Les explosions ne se déplacent pas donc ce n'est pas utile d'appeler `super().update`.

```
def update(self, dt) :
    self.timer -= dt

    if self.timer <= 0 :
        self.alive = False
        self.destroy()
```

14.3 Alien

Comme les tirs sont gérés par `Fleet`, la classe `Alien` diffère de `GameObject` que par la méthode `get_value` décrite dans le chapitre sur les scores, et par et le chargement des images. La méthode `update` héritée de `GameObject` s'occupe de mettre à jour la position des aliens.

14.4 Bullet

Comme pour `Explosion`, un son est joué à l'instanciation, et comme `Alien` la méthode `update` n'a pas besoin d'être redéfinie. La vitesse verticale doit être passée en paramètre au constructeur ce qui permet d'avoir des projectiles qui vont vers le haut ou vers le bas.