Chapitre 1-4
Installer son environnement de travail

1. Introduction

Il ne s'agit ici que de CPython, l'implémentation de référence de Python, et non de PyPy ou Jython.
Quel que soit votre système d'exploitation, vous pouvez installer Python en lisant ce chapitre puis, dans un second temps, installer des bibliothèques tierces au gré de vos besoins (cf. section Installer une bibliothèque tierce) et vous pourrez créer des environnements virtuels (cf. section Créer un environnement virtuel).
Si vous souhaitez installer d'un seul coup Python ainsi que Jupyter (anciennement IPython) et la plupart des bibliothèques scientifiques ou d'analyse de données, vous pouvez aller directement à la section Installer Anaconda, pour installer celui-ci en lieu et place de Python. Vous disposerez alors d'autres méthodes pour gérer les environnements virtuels et pour installer des bibliothèques tierces.

2. Installer Python

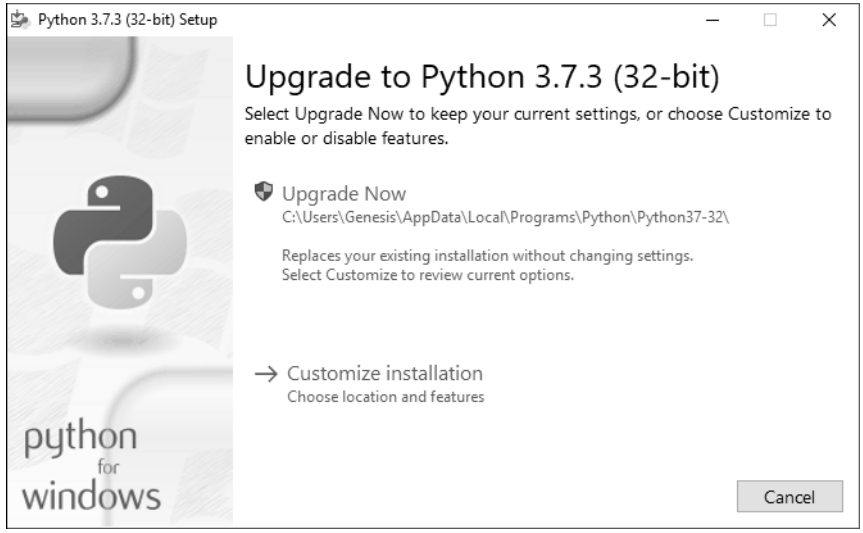
2.1 Pour Windows

Le système d'exploitation Windows requiert usuellement l'utilisation d'un installateur pour pouvoir installer un logiciel quel qu'il soit. Si vous disposez de Windows, vous devriez en avoir l'habitude. Python ne déroge pas à la règle.
Pour installer Python, vous devez donc aller sur le site officiel (https://www.python.org/downloads/) pour télécharger l'installateur adéquat. Comme vous pourrez le constater, on vous met en avant un accès rapide à la dernière version (au moment où ces lignes sont écrites, la 3.11.0), puis un accès aux dernières versions encore actives (actuellement la version 3.10 qui reçoit encore des corrections d'anomalies, puis les versions 3.9 à 3.7 qui reçoivent des corrections de sécurité uniquement).

Il est également possible de télécharger la toute dernière version de la branche 2.7 qui est en fin de vie (elle n'est plus mise à jour), car il existe encore de nombreux projets n'ayant pas encore migré.
Le support correctif dure 2 ans après la première sortie de la version et le support de sécurité dure 5 ans.
Pour notre part, nous vous conseillons la dernière 3.x, mais vous êtes libre d'installer celle que vous souhaitez ou même d'en installer plusieurs suivant vos contraintes, il n'y a pas d'objection à cela.
Une fois le téléchargement effectué, vous devez lancer l'installateur (et éventuellement passer quelques protections de votre système qui vous demande d'accorder votre confiance à cet installateur), pour observer l'écran suivant :

Comme vous pouvez le constater, il est possible de personnaliser l'installation en choisissant le chemin d'installation du logiciel ou en choisissant de ne pas sélectionner quelques fonctionnalités, mais nous ne le conseillons pas.
Nous vous recommandons en revanche de cocher la case **Add python.exe to PATH** afin de configurer la variable PATH du terminal pour rendre Python accessible plus facilement.

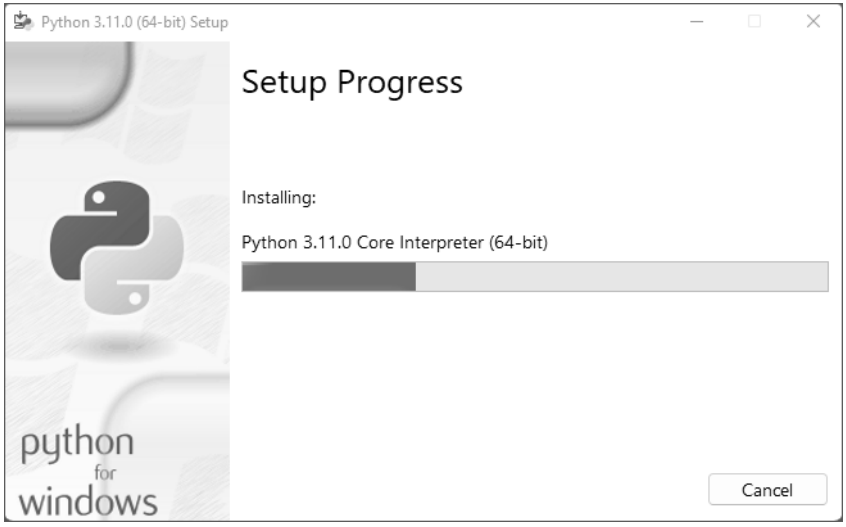
Si vous avez déjà une ancienne version de Python installée de la même branche (dans cet exemple, Python 3.7.2 est déjà installé), vous pourrez la mettre à jour à l'aide du même installeur :



Par contre, si vous avez déjà la version 3.7.1 et que vous installez la version 3.11, cette dernière ne viendra pas remplacer la précédente, mais s'installera à côté. Si vous souhaitez remplacer, il vous faudra donc désinstaller proprement la toute dernière version installée, ce qu'il est possible de faire en relançant l'installeur d'origine.

Nous vous encourageons à garder les installeurs sur votre PC, car ils pourraient devenir indisponibles au téléchargement si trop vieux.

Quel que soit le scénario, vous arriverez devant un écran vous montrant la progression de l'installation et vous n'aurez qu'à fermer la fenêtre une fois celle-ci terminée :



Vous êtes maintenant prêt à utiliser Python.

2.2 Pour Mac

Il faut savoir qu'une version de Python est déjà préinstallée sur Mac, car Mac OS X l'utilise pour ses propres besoins et Python est intégré à son propre cycle de développement. Cependant, si vous souhaitez une version différente de celle qui est déjà présente, vous pouvez l'installer, sachant qu' il n'y a pas de contre-indication à posséder plusieurs versions de Python sur la même machine.

Pour installer Python sur Mac OS X, la procédure à suivre est similaire à celle pour Windows. Il faut donc se rendre sur le site officiel (https://www.python.org/downloads/mac-osx/), télécharger un installateur correspondant à sa configuration et suivre les étapes.

Pour les utilisateurs de Mac, sachez que Python dispose d'une bonne intégration de ses spécificités, en particulier vis-à-vis de Objective-C, le langage de programmation avec lequel est développé Mac OS X, et Cocoa, interface de programmation de Mac OS X.

2.3 Pour GNU/Linux et BSD

Les différentes distributions libres utilisent nativement Python, notamment pour des parties sensibles. Python y est donc tout naturellement déjà installé, généralement sous la dernière version de la branche 2.x. Cependant, ici comme ailleurs, il n'y a pas d'objections à utiliser plusieurs versions de Python.
Le plus simple reste d'utiliser votre gestionnaire de paquets, ce qui peut se faire via un outil graphique, comme Synaptic pour Debian :

Il suffit alors de faire une recherche sur le mot-clé **`python`** pour voir les différentes versions (sur une ancienne Debian, ce sont Python 2.6, 2.7 et 3.2).
Par contre, tous les paquets python3-xxxxx que vous pouvez voir ici sont des bibliothèques tierces et non Python lui-même. Nous en parlerons plus tard dans ce chapitre.
Une fois les paquets souhaités sélectionnés, il ne manque plus qu'à les installer en cliquant sur le bouton **Appliquer**.
Notez que tout ceci peut se faire par la simple ligne de commande, toujours en utilisant votre gestionnaire de paquets qui peut être apt-get, aptitude, yum, emerge, pkg_add ou autre.
Voici par exemple pour une distribution Debian ou Ubuntu :

```
$ sudo aptitude install python3
```

Ceci ne permet cependant pas de choisir la version que l'on souhaite, à moins d'aller trouver des sources alternatives. Si l'on veut avoir la toute dernière version de Python, il faudra la plupart du temps passer par la compilation.

2.4 Par la compilation

Compiler Python n'est pas en soi une tâche très complexe. C'est par contre souvent une tâche imposée lorsque l'on ne travaille pas avec des conteneurs. En effet, en entreprise, on développe souvent des applications qui sont destinées à être hébergées. Il est alors impératif de travailler sur votre propre poste avec une version de Python qui soit la même que celle existante sur la machine de production.

Sous GNU/Linux, mais aussi sous d'autres systèmes, il est possible de compiler la version de Python que l'on souhaite. Après tout, Python n'est rien d'autre qu'un programme écrit en C. Pour ce faire, il faut aller télécharger le code source (https://www.python.org/downloads/source/), qui prend la forme d'une archive, puis décompresser celle-ci, se placer dans le répertoire ainsi obtenu et taper ces quelques commandes :

```
$ ./configure –prefix=/path/to/my/python/directory
$ make
$ sudo make altinstall
```

Notez que dans cette dernière ligne, nous n'utilisons pas la commande **`make install`**, qui aurait pour effet de remplacer votre Python système par le Python que vous compilez, ce qui pourrait avoir des conséquences indésirables voire désastreuses.

Notez également que vous choisissez lors de la configuration le chemin dans lequel vous placerez vos bibliothèques Python. En général, l'usage veut que l'on utilise **`/opt`**, mais il n'y a pas de règle, tout dépend des pratiques de votre entreprise ou votre expérience en la matière.

Si vous venez d'installer Python 3.5 par cette méthode, vous aurez alors maintenant accès à ce programme en l'appelant ainsi, depuis votre terminal :

```
$ python3.5
```

Par cette même méthode, vous pouvez installer les dernières versions (https://www.python.org/download/pre-releases/) de Python qui ne sont pas encore sorties (alphas ou betas), ce qui vous permet de les tester en avant-première !

Notons que, par cette méthode, toutes les bibliothèques de Python ne fonctionneront pas. En effet, lorsqu'elles ont besoin d'autres bibliothèques C, il faut effectuer des compilations croisées et utiliser les différents en-têtes de ces bibliothèques. C'est le cas par exemple pour faire fonctionner Curses, ReportLab (génération de fichiers PDF) ou encore PyUSB (accès aux ports matériels USB).

Dans ce cas-là, la commande **`./configure`** devra recevoir des arguments supplémentaires et vous devrez trouver un tutoriel en ligne pour vous indiquer la démarche, laquelle peut être plus ou moins complexe.

2.5 Pour un smartphone

Installer une machine virtuelle Python sur un smartphone est possible. Pour Android, la procédure est assez simple puisqu'il existe un produit dédié (http://qpython.com/), tout comme sur Windows Phone (https://apps.microsoft.com/store/detail/python-39/9P7QFQMJRFP7). Pour iOS, c'est une autre paire de manches (https://github.com/linusyang/python-for-ios) étant donné que l'utilisateur est enfermé dans un système sur lequel il n'a aucun contrôle.

3. Installer une bibliothèque tierce

Remarque

Si vous abhorrez le terminal, sachez que vous pouvez installer une bibliothèque tierce depuis votre IDE, ce qui sera probablement plus aisé pour vous.

3.1 À partir de Python 3.4

Pour installer une bibliothèque tierce, vous devez simplement connaître son nom. Celui-ci est généralement assez intuitif. Par exemple, la bibliothèque permettant de communiquer avec un serveur Redis s'appelle redis.
Il peut y avoir des variations. Par exemple, la bibliothèque de référence pour traiter du XML est lxml et, plus complexe, celle pour BeautifulSoup est bs4. En recherchant comment répondre à un besoin sur le Net ou sur PyPi (https://pypi.python.org/pypi), vous trouverez rapidement une bibliothèque de référence.
Sur des sujets plus confidentiels, il vous arrivera de trouver plusieurs petites bibliothèques. Vous pourrez alors les tester et choisir celle que vous utiliserez pour votre projet.
Sachez que vous pouvez aussi conduire une recherche directement depuis votre terminal :

```
$ pip search xml
$ pip search soup
```

Cela vous donnera une liste de bibliothèques accompagnée d'une courte description, à la manière de ce que font les gestionnaires de paquets sous Linux (lesquels sont écrits en Python, au passage).
Sachez que **`pip`** existe quel que soit votre système d'exploitation (vous devez être familier avec le terminal de votre système, cependant) et que depuis la version 3.4 de Python, il est installé automatiquement avec celui-ci. Si ce n'est pas votre cas, consultez la section suivante : Pour une version inférieure à Python 3.4.
`pip` est un outil formidable. Si vous utilisez une version de Python qui est celle du système, vous utiliserez alors la commande **`pip`** pour gérer les bibliothèques. Si vous utilisez une autre version, telle que Python 3.5, alors vous utiliserez la commande **`pip-3.5`**. Pour Python 3.3, ce sera **`pip-3.3`**. Dans les exemples suivants, il vous faudra prendre en compte cette particularité.
Cet outil vous permettra d'installer une bibliothèque à sa dernière version ainsi que toutes les bibliothèques dépendantes. En effet, il n'est pas rare qu'une bibliothèque de Python ait besoin d'une autre bibliothèque (ou de plusieurs) pour fonctionner. Par exemple, l'installation de redis se fait par cette commande :

```
$ pip install redis
```

On peut aussi choisir la version à installer :

```
$ pip install -Iv redis==2.10.5
```

Ou mettre à jour la bibliothèque à une version précise :

```
$ pip install -U redis==2.10.5
```

Ou à la dernière version :

```
$ pip install -U redis
```

Et on peut la désinstaller :

```
$ pip uninstall redis
```

Une fonctionnalité très importante permet d'obtenir la liste des bibliothèques déjà installées (quelle que soit la manière dont elles ont été installées) :

```
$ pip freeze
```

Ce que l'on peut mettre dans un fichier :

```
$ pip freeze > requirements.txt
```

Pour installer tous les paquets ainsi listés, il faut procéder ainsi :

```
$ pip install -r requirements/base.txt
```

Cette méthode est particulièrement utile dans le cadre d'un environnement virtuel ; nous y reviendrons.
Il est possible de retrouver des informations sur un paquet déjà installé :

```
$ pip show django-redis
---
Name: django-redis
Version: 4.3.0
Location: /path/to/my/env/lib/python3.4/site-packages
Requires: redis
```

On voit ici que le paquet **`django-redis`** a une dépendance vers **`redis`** : en l'installant, on installe automatiquement **`redis`**.
Mettre à jour ce paquet met à jour automatiquement les dépendances :

```
$ pip install -U django-redis
```

Si on ne veut pas mettre à jour les dépendances, on peut procéder ainsi :

```
$ pip install -U --no-deps django-redis
```

On peut aussi installer plusieurs bibliothèques en même temps :

```
$ pip install django-redis==4.3.0 bs4 lxml
```

Cette commande installera donc automatiquement redis s'il n'est pas installé, car il est déclaré comme dépendance.
Cette commande a cependant des limites. En effet, si vous installez une bibliothèque tierce qui utilise une bibliothèque C, vous devrez disposer des en-têtes C correspondants (paquets **`dev`** pour Debian ou **`devel`** pour Fedora). Il faut donc avoir un peu de pratique dans ce genre de situation pour savoir déjouer ces pièges.

Chapitre 3

Concepts du jeu vidéo et premiers pas à propos de Pygame

1. Introduction

Le présent chapitre se propose à la fois de présenter différents concepts relatifs à Pygame d'une part, et d'autre part de commencer à expliquer comment Pygame apporte des solutions pratiques face à ces concepts. Il nous faut notamment expliquer ce qu'est une boucle de jeu, puis présenter la notion de gestion des collisions. Ce dernier aspect est particulièrement fastidieux à gérer sans Pygame. Nous prendrons un petit exemple d'un jeu dont la gestion des collisions est réalisée sans Pygame. Celui-ci nous permettra ainsi de démontrer à quel point Pygame est pratique pour la gestion de cet aspect si central en développement de jeux vidéo.

2. La boucle de jeu

Avant d'aller plus loin, il faut définir une notion fondamentale du développement de jeux vidéo : la boucle de jeu. Parfois appelée « boucle d'animation », elle se retrouve dans (presque) tous les jeux vidéo et constitue à ce titre une sorte de colonne vertébrale.

Elle correspond à une boucle infinie. Évidemment, il faut pouvoir l'interrompre. Pour cela, on code une action utilisateur (par exemple, appuyer sur la touche [Esc] du clavier). Un appui sur cette touche interrompt la boucle infinie et donc la partie en cours.

Que fait donc cette boucle à chaque itération ? Peu ou prou, les étapes suivantes :

1. Vérifier que les conditions d'arrêt ne sont pas atteintes et, si elles le sont, interrompre la boucle.

2. Mettre à jour les ressources nécessaires pour l'itération courante.

3. Obtenir les entrées soit issues du système, soit issues de l'interaction avec le joueur.

4. Mettre à jour l'ensemble des entités qui caractérisent le jeu.

5. Rafraîchir l'écran.

À la fin d'une itération, la boucle infinie se poursuit et on passe à l'itération suivante.

Idéalement, chaque itération doit avoir la même durée que toutes les autres pour permettre une certaine fluidité dans le déroulement du jeu. Comme nous le verrons par la suite, c'est aussi l'un des avantages de Pygame qui offre des outils de gestion du temps. Pour information, on travaille en général avec une modalité de 30 images par seconde avec Pygame, ce qui revient à voir la boucle infinie du jeu réaliser 30 itérations par seconde.

3. Présentation de Pygame

Cette section est avant tout là pour contextualiser Pygame et expliquer comment ce framework est conçu, pourquoi et dans quels contextes.

Pygame est une bibliothèque logicielle, c'est-à-dire un ensemble de ressources logicielles, dédiée au développement de jeux vidéo temps réel. Cette bibliothèque a également la spécificité et le grand intérêt d'être un logiciel libre, dans la mesure où elle est distribuée sous licence GNU LGPL.

Ce qu'il faut bien comprendre, c'est que pour faire un jeu, il faut nécessairement que des éléments de code interagissent avec des parties « bas niveau » de l'ordinateur, c'est-à-dire avec le matériel, que ce soit l'écran, la carte son, le clavier, la souris, voire d'autres périphériques dédiés au jeu. Ces interactions « bas niveau » sont complexes, en général développées avec des langages comme le langage C, et utilisent la bibliothèque SDL (*Simple DirectMedia Layer*) qui n'est pas toujours très simple à prendre en main.

La bibliothèque Pygame constitue une couche qui se situe au-dessus de SDL et qui simplifie l'accès et la manipulation de ces différents périphériques de manière simple et intuitive, qui plus est en développant avec le langage Python.

Pygame offre des modules simplifiant tous les aspects du jeu : entre autres le graphisme, le son, les interactions avec la souris ou le clavier, la gestion des événements.

4. Installation de Pygame

Le moyen le plus simple d'installer Pygame est d'utiliser le programme `pip`. Ainsi, la ligne de commande suivante permet d'ajouter la dernière version de Pygame.

```
--------------------------------------------------------------------------------

pip install pygame

--------------------------------------------------------------------------------
```

Une fois installé, vous pouvez immédiatement vérifier la version de Pygame, avec la commande suivante lancée dans un environnement Python :

```
import pygame

--------------------------------------------------------------------------------
```

On obtient :

```
> pygame 2.1.2 (SDL 2.0.18, Python 3.8.8)
```

Il est par ailleurs possible d'installer Pygame avec des programmes incluant de nombreux autres outils, comme Anaconda par exemple.

Au moment de la rédaction de ce livre, la version de Pygame (comme indiqué ci-dessus) est la version 2.1.2, incluant la version de SDL 2.0.18.

Il peut être intéressant de commencer par visiter le site officiel du projet Pygame (https://www.pygame.org), d'une part pour découvrir les différentes ressources qui pourront vous être utiles par la suite, mais d'ores et déjà pour vérifier quel est le numéro de la version la plus récente. Par ailleurs, le site officiel propose plusieurs alternatives d'installation selon votre système d'exploitation.

Que vous soyez sous Windows, Linux, ou macOS, l'installation de Pygame est extrêmement simple. Rendez-vous pour commencer à l'adresse web suivante :
https://www.pygame.org/download.shtml

La page de téléchargement est organisée selon le système d'exploitation utilisé :

- Sur Windows, téléchargez le programme d'installation (.msi) adéquat, puis exécutez-le.
- Sur Linux, choisissez la ressource correspondant à votre distribution (Ubuntu, Debian, Fedora, etc.), téléchargez-la puis exécutez-la avec la commande d'installation indiquée.
- Sur macos, téléchargez la ressource adéquate (.dmg), puis exécutez le programme d'installation inclus (.mpkg).

L'installation de Pygame à laquelle vous avez procédé a également et de façon transparente installé SDL.

Même si l'installation de Pygame semble s'être bien déroulée (pas de message d'erreur à l'horizon), il s'agit tout de même de vérifier que tout est correctement en place. Faites l'import de Pygame dans une ligne de commande Python pour vérifier qu'une version de Pygame est bien proposée et qu'elle est cohérente avec ce que vous avez pu lire sur le site web.

Si problème, il se peut qu'il soit dû soit à un problème de cible 32 ou 64 bits, soit à une version de Pygame ne correspondant pas à celle de Python sur votre machine.

5. Les modules composant Pygame

Cette section recense les différents modules qui composent Pygame, ce qui donne un premier aperçu des possibilités. Avec seulement quatre ou cinq de ces modules (ou classes) qu'on utilise le plus souvent en général (*event*, *image*, *draw*, *surface*, *time*, par la suite *sprite*), il est possible de réaliser des jeux vidéo avancés.

- `cdrom` : gestion des lecteurs CD/DVD.
- `cursors` : gestion du curseur de la souris.
- `display` : configuration de la surface d'affichage.
- `draw` : dessin de formes surfaciques.
- `event` : gestion des événements graphiques ou relatifs aux périphériques.
- `font` : gestion des polices de caractères.
- `image` : gestion des images.
- `joystick` : gestion des périphériques de jeu.
- `key` : gestion du clavier.
- `mixer` : gestion générale du son.
- `mouse` : gestion de la souris.
- `movie` : gestion des aspects vidéo.
- `music` : gestion de la musique.
- `overlay` : gestion avancée de la vidéo.
- `pygame` : gestion de la bibliothèque Pygame elle-même.
- `rect` : gestion spécifique de la forme rectangulaire.
- `sndarray` : gestion de données sonores.
- `sprite` : gestion des sprites, objets de haut niveau utilisés dans Pygame.
- `surface` : gestion avancée d'images.
- `surfarray` : gestion et manipulation d'images.

- `time` : gestion du temps et du rafraîchissement de l'écran.
- `transform` : transformation et déplacement des images.

6. Réalisation d'un premier jeu graphique : fusée et planètes

L'idée ici est de réaliser un petit jeu dont le but est de piloter un engin volant et d'éviter certains obstacles : des planètes, deux en l'occurrence, tombant du ciel. La fusée doit les éviter en allant à gauche ou à droite. Ce sera là l'action du joueur : déplacer à gauche ou à droite (latéralement uniquement) la fusée pour éviter les planètes. Si une planète touche la fusée, c'est évidemment perdu.

Pour cela, nous allons écrire un petit code en Pygame en détaillant le plus possible chaque étape.

Le code est muni d'un fichier requirements.txt qui vous autorise à créer et activer un environnement virtuel. Ici, il n'y a qu'un seul module, en l'occurrence Pygame.

L'un des intérêts de cette démarche est d'expliquer et d'illustrer simplement deux grands aspects du développement Pygame, à savoir :

- le système de coordonnées dans une fenêtre Pygame,
- la gestion des collisions, c'est-à-dire le code qui permet de gérer la rencontre de deux objets graphiques (si l'on veut par exemple simuler graphiquement le rebond d'une balle sur le sol).

La suite du livre détaille avec précision chaque aspect abordé. Mais ce premier développement offre une vue globale de ce qu'est Pygame et de ce qu'il permet de faire.

6.1 Les images utilisées

L'engin volant est défini par une image au format JPEG ou au format PNG. Vous pouvez utiliser une de vos photographies ou une de vos créations graphiques réalisées avec un logiciel de dessin ou encore une image trouvée sur le Web, sous réserve qu'elle soit sous licence permettant une telle réutilisation gratuite et éventuellement sans nécessité d'attribution. C'est le cas de l'image de fusée utilisée ici. Elle est stockée dans le répertoire du projet sous le nom FUSEE.png. Le véhicule volant a une orientation verticale, en vue d'un défilement automatique du jeu orienté vers le haut. Idéalement enfin, l'image choisie doit avoir un fond transparent.

L'image de planète que la fusée doit éviter est stockée sous le nom PLANETE.png.

6.2 La fenêtre du jeu

Créer la fenêtre du jeu est la première étape. Il s'agit de la fenêtre qui accueille les éléments graphiques qui composent le jeu.

Sans surprise, on commence par importer Pygame :

```
import pygame
```

Puis, on initialise Pygame :

```
pygame.init()
```

Les deux lignes précédentes sont systématiquement présentes dès lors que l'on utilise Pygame.

Définissons maintenant les dimensions de la fenêtre :

```
HAUTEUR_FENETRE = 600
LARGEUR_FENETRE = 600
```

Puis, nous définissons la couleur de fond de la fenêtre :

```
COULEUR_FOND = (255, 255, 255)
```

Et enfin, nous utilisons la commande d'affichage de la fenêtre :

```
ECRAN = pygame.display.set_mode((LARGEUR_FENETRE, HAUTEUR_FENETRE))
```

Si vous lancez le programme, la fenêtre du jeu s'affiche avec un fond blanc. Aucune indication sur la durée de l'affichage n'est apportée, on la voit donc s'ouvrir et se fermer le temps d'une fraction de seconde.

6.3 La boucle du jeu

Pour définir la boucle du jeu, nous commençons par définir un booléen : si sa valeur est `True`, la boucle se poursuit, elle s'interrompt sinon.

```
ARRET_JEU = False
```

La fonction `pygame.event.get()` permet d'intercepter tous les événements entrants notamment depuis le clavier, la souris, etc. Si on appuie sur la touche [Esc], alors le jeu est interrompu. Ceci est permis en utilisant les événements du clavier suivants : touche appuyée (`pygame.KEYDOWN`) et touche [Esc] ou [Echap] (`pygame.K_ESCAPE`).

On définit donc ainsi la boucle de jeu :

```
while not ARRET_JEU:
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                ARRET_JEU = True
```

À ce stade, le code est le suivant :

```
import pygame
pygame.init()
HAUTEUR_FENETRE = 600
LARGEUR_FENETRE = 600
COULEUR_FOND = (255, 255, 250)
ECRAN = pygame.display.set_mode((LARGEUR_FENETRE, HAUTEUR_FENETRE))
# booléen de gestion de la boucle
ARRET_JEU = False
while not ARRET_JEU:
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                ARRET_JEU = True
```

La fenêtre s'affiche et l'affichage s'interrompt dès qu'on appuie sur [Esc]. Nous sommes pour le moins encore assez loin de quelque chose d'amusant. Nous allons donc entrer dans le vif du sujet et commencer à parler de fusée et de planètes.