Projet Heaven

LEVIN Marvyn – MORETTI Tony – COLIN Cédric

« Les icônes sont réalisés par Freepick à partir de www.flaticon.com »

Les informations

- Le projet
- L'histoire
- Les lieux

Les informations | Le projet

But :
- **jeu vidéo** fonctionnel
- style **rétro**
- en **2D**
- un **RPG** (Role Play Game)
- des **quêtes**
- une **histoire** fantastique

Langage utilisé : Le **langage Python** sera utilisé pour le programme avec les **bibliothèques suivantes** :

Python 3.7.10 (Pygame - Maths - Random - Typing - Enum – Json – Memory-profiler)

Les informations | L’histoire

Histoire :

Kady, une épéiste talentueuse, part à la recherche de son ami enlevé dans un lieu maléfique. Elle affronte des ennemis redoutables (**myrtis**), y compris sa propre **mère Eris**, pour **protéger son village**.

Avec l'aide de son animal de compagnie, **Flocon**, et de son **père Gabriel**, elle réussit à ramener son ami **Léon** sain et sauf. Kady décide de former un groupe de défense local pour **protéger Astrelia**.

Bien qu'elle souhaite retrouver sa mère pour **obtenir des réponses**, elle sait que sa place est auprès de sa famille et de ses amis. Kady fait la **paix** avec son passé et se concentre sur la **protection** de ceux qu'elle aime.


Kady


Flocon


Léon


Gabriel


Eris


Myrtis

Les informations | Les lieux

Histoire :

Astrelia:

Nirvania:

Myrrhanda:

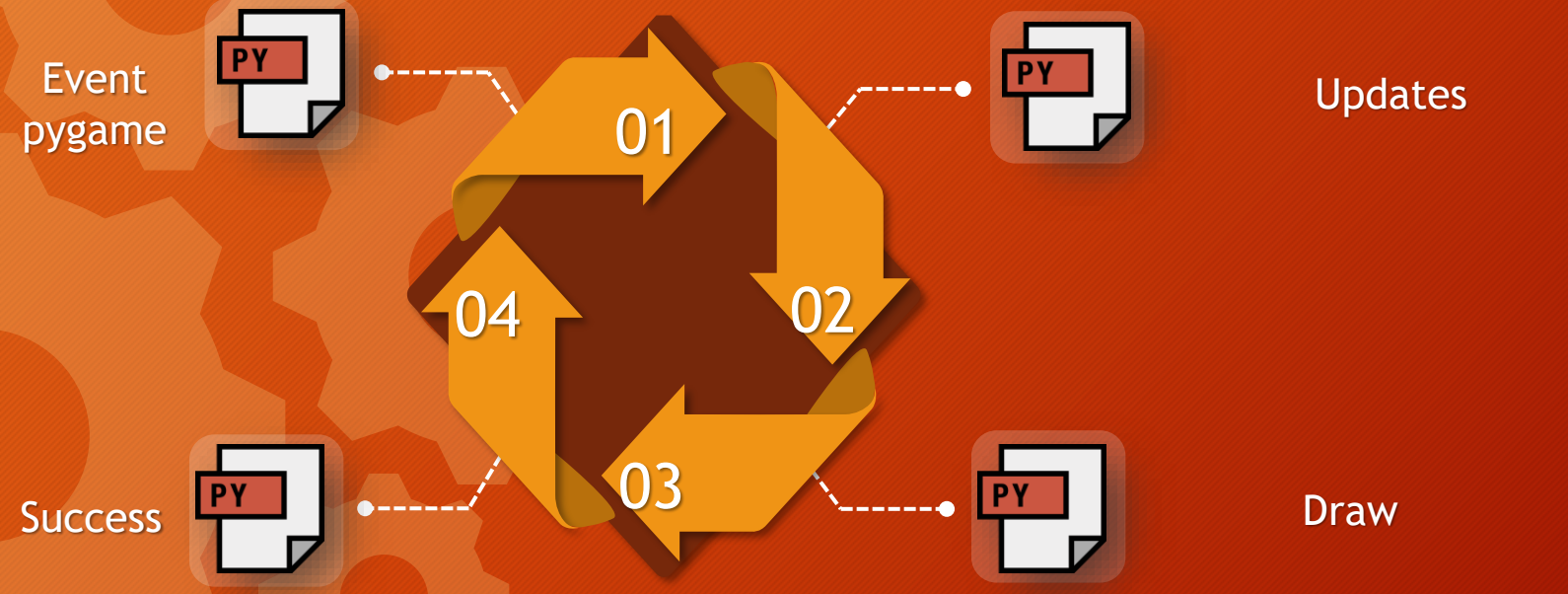
Le fonctionnement

- L’idée principale
- Le cœur
- Le delta
- Les dossiers

Le fonctionnement | L'idée principale

Fonctionnement :

Le **cœur du programme** relève du fichier **main.py** qui est comme la racine de notre projet.
Seulement pour que celui-ci fonctionne, nous avons d'**autres fichiers pythons.**



Et des **fichiers** sous d'**autres formats :**

Le fonctionnement | Le cœur

Fonctionnement :

Le **fonctionnement** du jeu se base sur une **boucle simple**, exprimée en langage naturel par :

```
jeu -> vrai
tant que jeu:
    mettre_a_jour_jeu()
    dessiner_jeu()
```

En effet, nous allons **alterner à l'infinie une boucle** :

```
... -> mise à jour -> dessin -> ...
```

Afin de lire les **divers évènements** (fermeture de fenêtres, pression de touches etc...), nous allons également insérer avant chaque mise à jour une **boucle qui va récupérer** les **évènements reçus** depuis la dernière mise à jour.

Le fonctionnement | Le delta

Fonctionnement :

Un des concepts les plus utilisés dans ce jeu est le concept de **delta**. Le delta est un **nombre flottant** qui correspond au **nombre de secondes** écoulées depuis la **dernière mise à jour**.

L'utilité est que le delta permet d'exprimer des valeurs en unités et d'être certains que si le nombre de mises à jour fluctue beaucoup, nous n'aurons pas de mouvements/animation saccadés.

Par exemple, la vitesse du joueur est exprimée en pixels/s, et est multipliée par le delta quand il est en mouvement. Ce qui fait que, si nous avons 60 mises à jour par secondes et que d'un moment nous passons à 127, alors les mouvements du joueur seront toujours fluides.

60 fps

127 fps

Le fonctionnement | Les dossiers

Fonctionnement :

Le jeu est organisé en plusieurs dossiers:

- **entities** : contient tout le code lié aux entités (parent > joueur, companion, mobs,)
- **categories** : contient le code source des catégories disponibles dans les menus (credits, load, options, success)
- **modules** : contient la logique du jeu (collisions, map, inventaires, save, interface, assets, camera, inventaire)
- **map** : contient les fichiers utilisé pour concevoir la carte (Avec le logiciel Tiled etn Paint.net)
- **algo** : contient des algorithmes qui ont été utilisés pour le jeu (Bug lié à Tiled)
- **assets** : contient les images utiles au jeu (Images.png)
- **poubelle** : le nom est transparent; ce dossier contient tous les fichiers bazars qui n'ont pas été utilisés/ne sont pas utiles dans l'immédiat (All files)

Le développement

- Les étapes
- Le code
- Les graphismes
- L’audio

Le développement | Les étapes

1. • Python - Pygame - Tiled

2. • Interfaces Menu - Game

3. • Catégories - Modules

4. • Graphismes - Audio - Optimisations

Le développement | Le code

Les collisions :

collisions.json > ...

```
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],[1,0,0,0,0,0,0,0,0,
),0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,0,0,0,0,0,
),0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
),1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
),0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,0,
),0,0,0,1,1,1,1,0,0,0,1,1,0,0,1,1,0,1,1,1,1,0,1,0,1],[1,
,0,1,1,0,0,0,0,0,0,1,1,1,1,0,0,1,0,0,0,0,1,0,1,0,0,1,1,
),0,1],[1,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,
),0,0,1,1,1,1,0,1,1,0,0,0,0,0,0,1,0,0,1,1,1,1,0,0,0,0,0,1,1,
),0,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
```

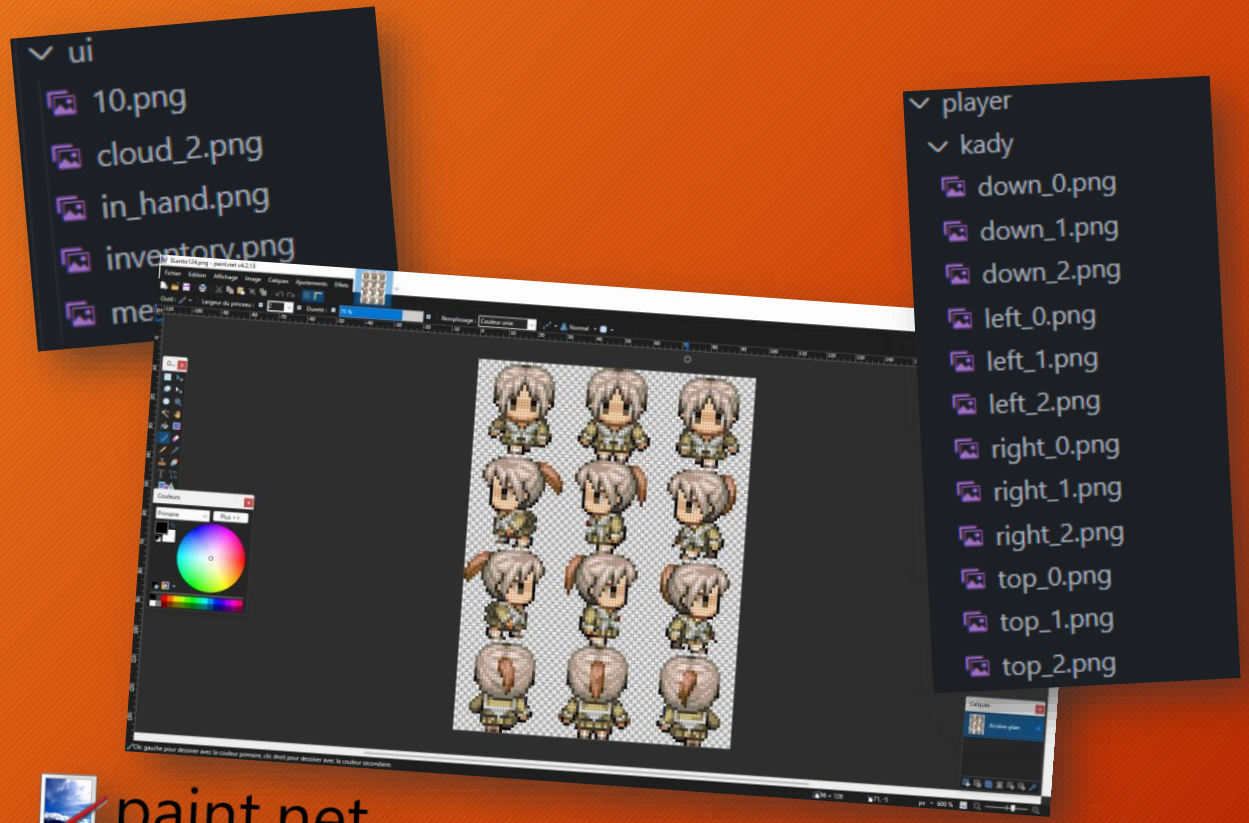
```python
def collisions_algo(collisions: List[List[int]], x: int, y: int) -> bool:
    """
    Prend en entrée un tableau 2D (symbolisant les collisions) et les coordonnées du joueur.
    Renvoi une boolean; true s'il y a collision, false sinon.
    """

    # Si la tuile n'existe pas, on ne veut pas faire planter le programme
    # Alors, on bloque toute erreur
    try:
        # On récupère la tuile sur laquelle est le joueur
        # `x` et `y` sont des nombres flottants.
        # Nous les divisons par les dimensions d'une tuile afin d'obtenir le nombre de la tuile
        tile = collisions[abs(floor((y / CASE_SIZE)))][abs(floor(x / CASE_SIZE))]
        # On renvoie le résultat de la fonction `check_col`
        return check_col(tile)
    except:
        return False
```

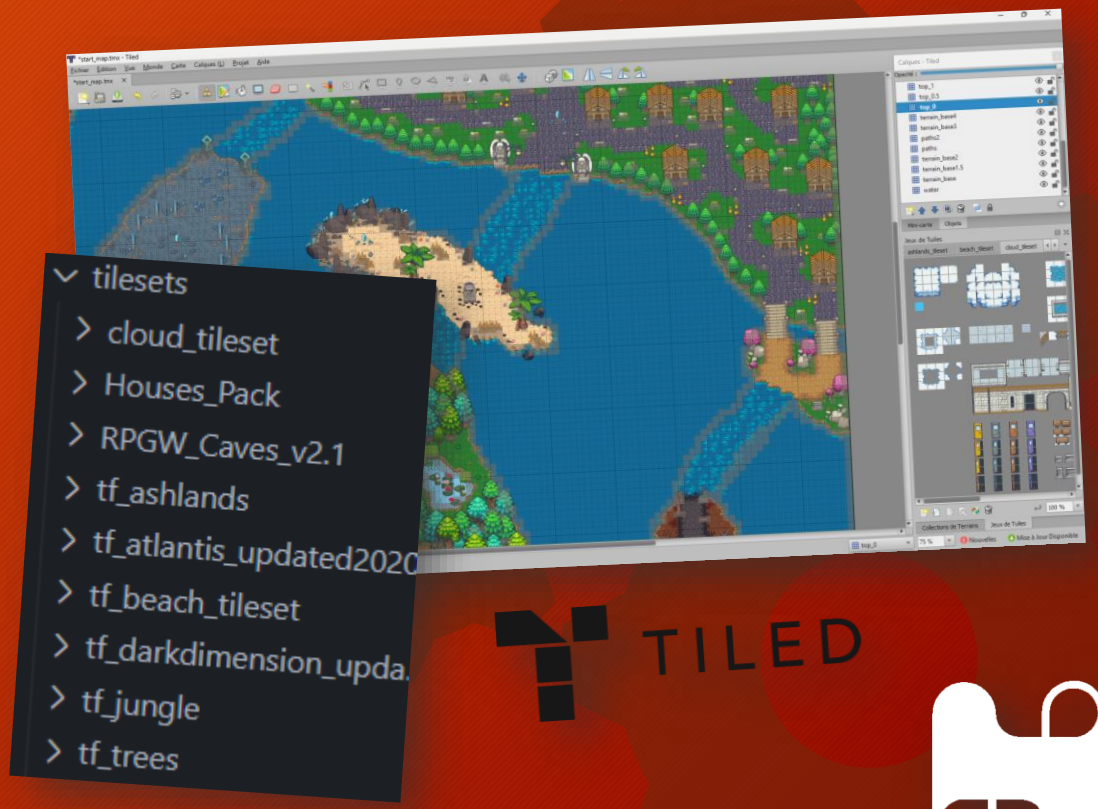
```
with open('./algo/collisions.json', 'r', encodin
    # Ce tableau contiendra nos collisions de la
    START_COLLISIONS = json.load(f)  # json.load
```

```python
def check_col(col: int) -> bool:
    """
    Fonction très simple
    vérifie si `col ` est 1, alias une collision pleine
    """
    return col == 1
```

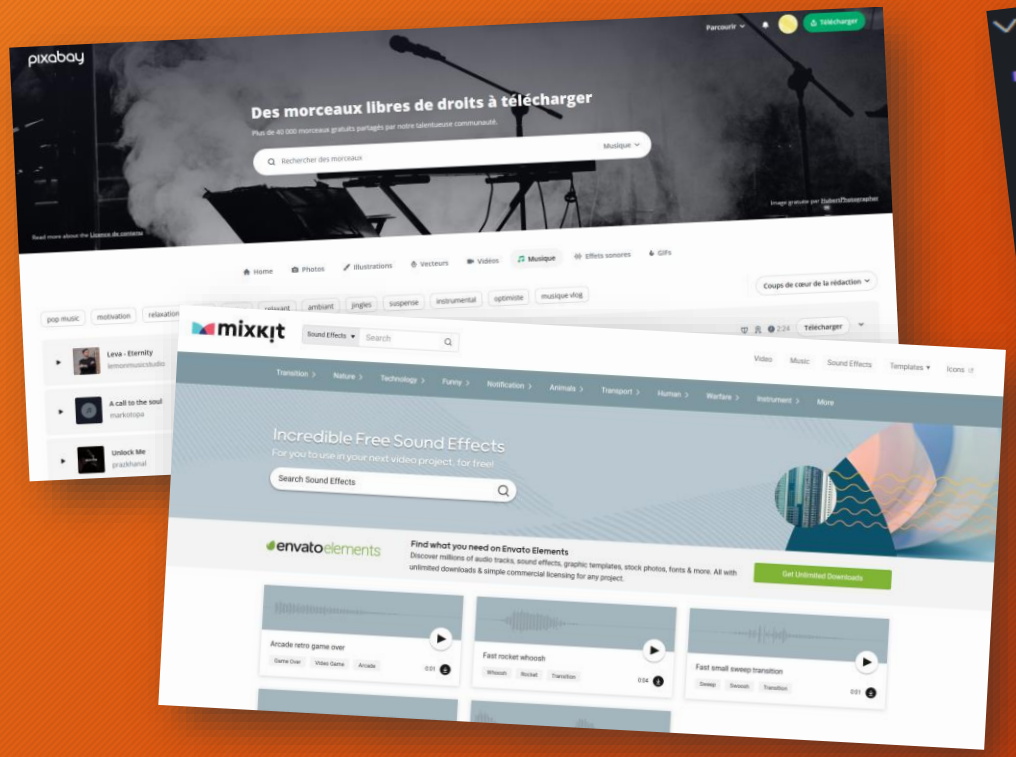
```
artir d'un string
```

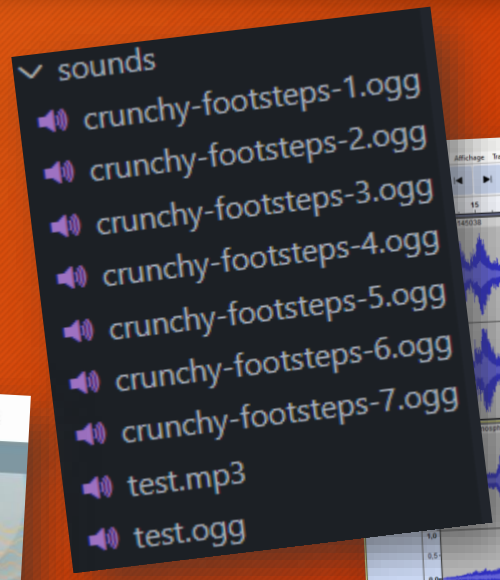
Le développement | Les graphismes

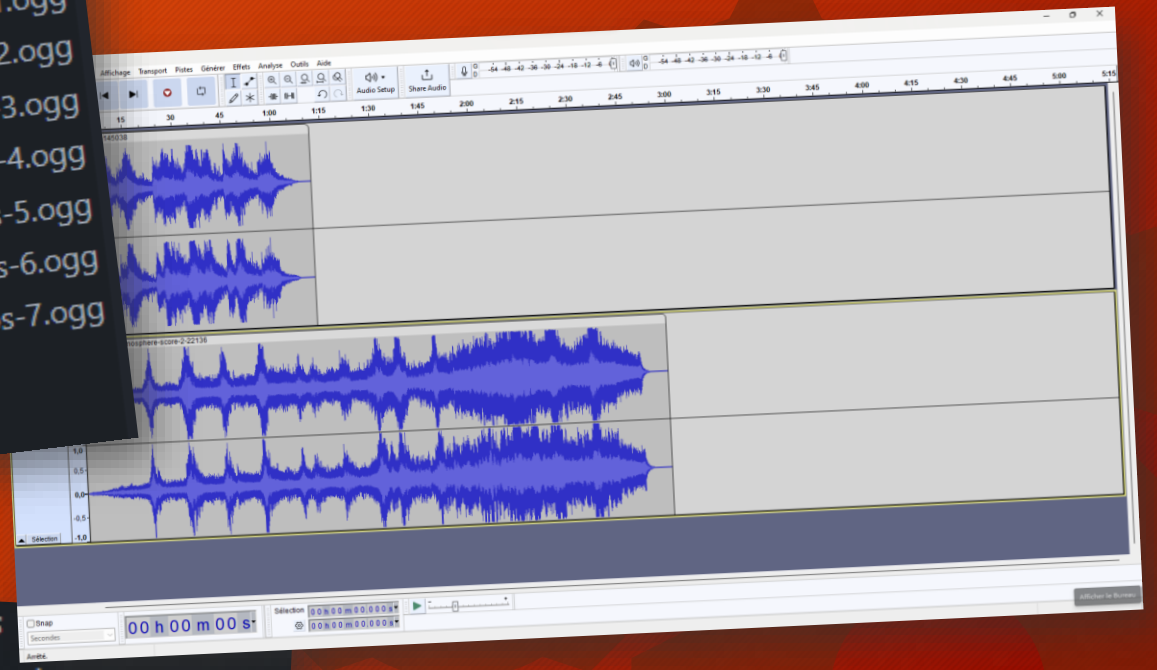


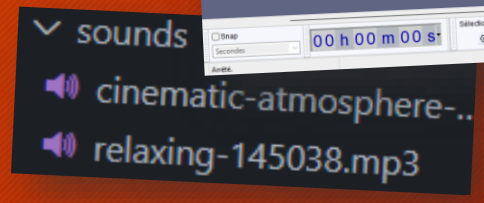


Le développement | L’audio









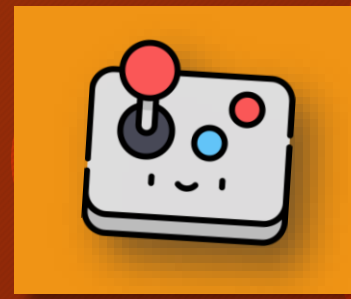
pixabay

mixkit

Audacity

Le jeu

- Le programme
- La répartition du travail



L. Marvyn

- Apprendre à manipuler **Pygame**
- Réaliser la **première interface** (menu.py)
- Réaliser **les catégories** (success.py – credits.py)
- Réaliser **les différents fichiers annexes** (achievement.py – collisions.py – fonts.py)
- **Aide** Tiled et Pygame (Graphisme + Interfaces)
- Trouver **les failles** (Corrections)

C. Cédric

- Apprendre à manipuler **Pygame**
- Réaliser **la logique** et **la deuxième interface** (main.py et game.py)
- Réaliser **les catégories** (load.py – options.py)
- Réaliser **les différents fichiers annexes** (cinematic.py – camera.py – inventory.py – save.py – ui.py)
- **Aide** Pygame (Interfaces)
- Trouver **les failles** (Corrections)

M. Tony

- Apprendre à manipuler **Tiled**
- Réaliser/Trouver **les Tilesets** (graphismes)
- Réaliser/Implémenter **la map**
- Réaliser **les différents fichiers annexes** (assets.py – map.py – sound.py)
- **Aide** Pygame (Interfaces)
- Trouver **les failles** (Corrections)

« L'histoire de Kady a été écrite par Cédric et la diapositive a été réalisé par Marvyn. »

Projet Heaven

LEVIN Marvyn – MORETTI Tony – COLIN Cédric

« Les icônes sont réalisés par Freepick à partir de www.flaticon.com »