CODING CLUB

EPITECH

Asteroids

DIVERSITY by EPITECH

Asteroids

Introduction

Le but de ce projet est de te faire découvrir la programmation orientée objet en réalisant un jeu vidéo. Tu vas devoir coder un jeu de type Asteroids en utilisant le langage Python.

Objectifs

- Consolider tes acquis en Python
- T'initier au paradigme de la programmation orientée objet
- S'amuser !

Prérequis

- Même si le sujet tend à être accessible, il est préférable d'avoir déjà une première expérience en programmation.
- Avoir installé Python sur ta machine ainsi que la librairie Tkinter.
- Avoir la volonté de découvrir (***c'est même le plus important***).

Ressources & communication

Le sujet peut sembler ne pas vous donner beaucoup d'instructions, car votre objectif est de faire les recherches nécessaires pour comprendre le code et mettre en œuvre la logique du jeu par vous-même. Ce projet n'est pas une évaluation, vous devez communiquer avec vos camarades de classe pour terminer le projet. Gardez à l'esprit que ce projet est difficile par conception, demandez à Google, demandez à vos camarades de classe, demandez-nous, mais n'abandonnez pas !

Good luck soldier !

Instructions

Tout au long du sujet, les parties de code à compléter seront indiquées par des commentaires `# TODO`.
Si la coloration syntaxique de votre éditeur de texte ne vous permet pas de les voir, vous pouvez utiliser la fonction de recherche de texte de votre éditeur pour les trouver.

Pensez bien à lire tous les commentaires, ils sont là pour vous aider.

Variables & types

Comme vous le savez sûrement, Python utilise un système à typage faible. Cela signifie que vous n'avez pas besoin de déclarer le type d'une variable avant de l'utiliser. Le type d'une variable est déterminé par la valeur que vous lui attribuez. Cependant, il n'est pas recommandé d'utiliser cette fonctionnalité car cela peut entraîner des confusions et des bugs. C'est pourquoi je vous recommande toujours de déclarer le type de vos variables. Pour ce faire, vous pouvez utiliser la syntaxe suivante :

```
variable: type = value
```

Pour les fonctions, vous pouvez déclarer le type des arguments et la valeur de retour comme ceci :

```
def function(argument: type) -> type:
    return value
```

Même si Python ne forcera pas le type, cela vous aidera à comprendre votre code et à éviter les erreurs.
Je vous recommande vivement d'utiliser cette fonctionnalité dans ce projet.

Installation & lancement

1. Pour démarrer le projet, il te suffit d'ouvrir le dossier `Participants`
2. Pour démarrer le projet, tu peux utiliser la commande suivante:

```
python3 main.py
```

Si tu n'a pas accès à un terminal, exécute le fichier `main.py` avec ton IDE Python.

Naviguer dans un termial
Si tu utilise un terminal, les commandes suivantes peuvent t'aider à naviguer dans les dossiers:
- `ls` : lister les fichiers et dossiers
- `cd ..` : remonter d'un dossier
- `cd <nom_dossier>` : entrer dans un dossier
- `pwd` : afficher le chemin du dossier courant (*ou qu'est ce que tu es*)

Programmation Orientée Objet

En Programmation Orientée Object (*on dira OOP*), on définit chaque élément du programme comme un object.
Un object c'est: - Un état (des variables, dit attributs) - Un comportement (des fonctions, dit méthodes) - Une identité (un nom) - Une classe (un modèle)

En parallèle de cette définition, l'OOP apporte plusieurs concepts:

Encapsulation : C'est le fait de regrouper les données et les méthodes qui agissent sur ces données dans une même entité. Cela permet de protéger les données et de les manipuler uniquement à travers les méthodes.
Dans ce projet, cela est représenté par la classe `Wrapper` qui encapsule les fonctions de Tkinter

Héritage : C'est le fait de créer une nouvelle classe à partir d'une classe existante. La nouvelle classe hérite des attributs et des méthodes de la classe existante.

Polymorphisme : C'est le fait de redéfinir une méthode héritée d'une classe parente dans une classe enfant. Cela permet de modifier le comportement de la méthode sans modifier la classe parente.

Abstraction : C'est le fait de cacher les détails d'implémentation d'une classe pour ne montrer que les informations essentielles. Cela permet de simplifier l'utilisation de la classe.

La plupart de ces concepts étant difficile à comprendre et à mettre en place, ils ne seront que très peu abordés dans ce projet.
Si tu veux en savoir plus, n'hésite pas à nous poser des questions!

1. Now the fun begins

1.0. Les bases

On appel "Point d'entrée" le fichier ou la fonction qui est exécutée en premier lors du lancement du programme.
Dans ce projet, le point d'entrée est le fichier `main.py`.

Ce fichier va donc être en charge de créer les ressources nécéssaires au programme, et de lancer la boucle de jeu.
Ta première mission est de réparer le point d'entrée du programme.

1.1. Les vecteurs

Maintenant que le point d'entrée est réparé, tu vas devoir t'attaquer à la base du jeu: les vecteurs.

> Pas de stress si tu n'es pas féru de mathématique!

Le fichier `Vector2D.py` contient la base d'une classe `Vector2D` qui représente un vecteur à deux dimensions, x & y.

Dans ce fichier, tu devras implémenter des **surcharge d'opérateurs**.
Les surcharges d'opérateurs permettent de redéfinir le comportement des opérateurs pour une classe donnée.

Par exemple, pour un type abstrait T, on peut redéfinir l'opérateur `+` pour que l'addition de deux objets de type T renvoie un nouvel objet de type T.

```
class T:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return T(self.value + other.value)
```

Pour s'assurer que la surcharge est correcte, tu peux utiliser cet assert:

```
def are_methods_overloaded(cls: type) -> bool:
    return cls.__str__ is not object.__str__
        and cls.__add__ is not object.__add__
        and cls.__sub__ is not object.__sub__

assert are_methods_overloaded(Vector2D), "Le vecteur n'est pas encore réparé"
```

Cet exercice est simple, mais marque la base de l'OOP.
Assure toi de bien avoir compris avant de passer à la suite.

Il est normal de bloquer sur cet exercice, n'hésite pas à demander de l'aide à tes camarades ou à nous!

1.2. Astéroïdes

La class `Asteroid`, définie dans le fichier du même nom est déjà complète.
Prend le temps de bien la lire pour comprendre son fonctionnement, ça te sera utile pour la suite.

1.3. Above and beyond

Maintenant que la base du projet est réparée, tu es prêt à passer aux choses sérieuses.
Voici les fichiers qu'il te reste à corriger: - `Bullet.py` - `Player.py` - `Wrapper.py`

S'ils peuvent tous être réparés en parallèle, je te conseille de les faire dans l'ordre proposé ci-dessus.

N'hésite pas à nous demander si tu es bloqué!

1.3. The final countdown

Si tu arrives à cette étape, c'est que tu as réparé le jeu, félicitation!

Mais après avoir joué un peu, tu as dû te rendre compte que le jeu est un peu... vide. C'est maintenant que tu peux laisser libre court à ton imagination! Tu peux implémenter de nouvelles fonctionnalités, de nouveaux éléments, ou même de nouveaux niveaux!

N'hésite pas à nous montrer ce que tu as fait!

Glossaire

- **Paradigme** : Un paradigme est un modèle de programmation, c'est-à-dire une manière de programmer. Il existe plusieurs paradigmes de programmation, dont la programmation orientée objet, la programmation impérative, la programmation fonctionnelle, etc.