kiss game engine

Version 1.1

mai 28, 2020

Contenu:

https://github.com/Fredkiss3/kge

KISS GAME ENGINE, alias kge, est le **premier moteur de jeu 2D** complet optimisé pour la création de jeux-vidéos avec le langage Python. Etant donné la lenteur de l'interprêteur python, le moteur a été optimisé en interne pour lancer vos jeux quelque soit leur taille. Ce moteur de jeu est basé sur un moteur de Jeu **PPB** créé par Piper Thunstrom.

CHAPITRE 1

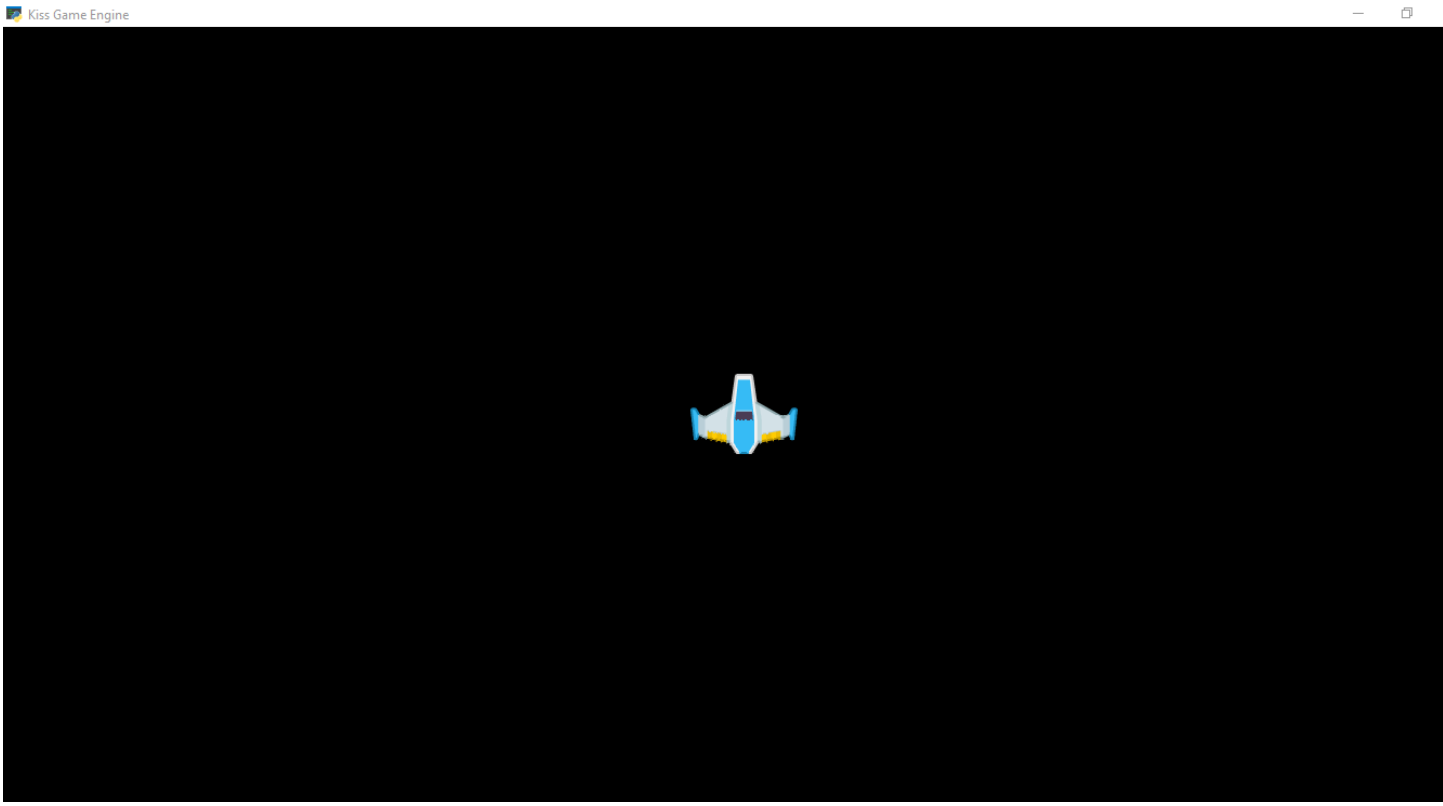
Une Bibliothèque optimisée et facile à utiliser

L'API du moteur utilise les *best practices* de la programmation orientée objet en python, avec une interface simple facile et à comprendre. Le moteur utilise un système d'évènements pour l'implémentation de la logique du jeu, des collisions, des évènements personnalisés, la liste complète des évènements se trouve dans le module `kge.core.events`. Le système utilise aussi une architecture d'' Entité Composant afin de pouvoir facilement ajouter des comportements personnalisés (Exemple : `Archer`, `Magicien`, `Goblin`).

Vous verrez vos jeux prendre vie en quelques lignes de code, la preuve :

```
import kge
from kge import *

def setup(scene):
    scene.add(Sprite(image=Image('player.png')))

kge.run(setup)
```

Résultat :



Nous plongeons dans le contenu de ce code dans les sections suivantes, j'espère que ce petit bout de code vous a mis l'eau à la bouche.

1.1 Démarrage

1.1.1 Pré-requis

kge ne marche que sur les versions de python avec `python 3.7` sur windows pour l'instant.

1.1.2 Installation

kge est installable depuis PyPI :

Mais si vous voulez l'installer depuis les sources, Vous pouvez le faire avec la commande sur le terminal :

Tester si le module a été installé

Afin de savoir si l'installation a été un succès, vous tester la commande suivante dans votre interpréteur python :

```
>>> import kge
>>> kge.version
1.0
>>>
```

1.1.3 Créer votre premier Jeu

Afficher le joueur

Une fois que l'installation est un succès, nous pouvons dès à présent créer notre premier jeu. Pour cela, nous devons créer un nouveau dossier et nous rendre dans celui-ci, ouvrez votre terminal et saisissez :

L'étape suivante est de créer un nouveau fichier. Si vous utilisez un IDE, Ouvrez votre projet dans celui-ci et créer un nouveau fichier nommé `game.py`. Si vous utiliser un simple éditeur de texte, vous devez créer un nouveau fichier et l'enregistrer sous le nom de `game.py`.

Note : `game.py` *est juste un nom de convention, vous êtes libre de le nommer tel que vous le voulez.*

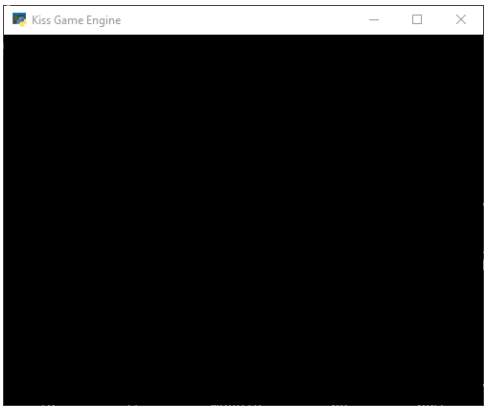
Dans votre code, ajoutez ceci :

`game.py` :

```
import kge

kge.run()
```

Enregistrez votre fichier et lancer le programme via la commande sur le terminal :

Vous devriez avoir le résultat suivant :



Ajoutons une image, pour rendre tout cela plus intéressant :

Nous nous baserons sur cette image (Il est mignon, mon petit vaisseau pas vrai ?) :

Copiez le fichier dans le dossier de votre jeu, et ajouter ceci dans votre code :

`game.py` :

```
import kge
from kge import *

def setup(scene):
    scene.add(Sprite(image=Image('player.png')))
```

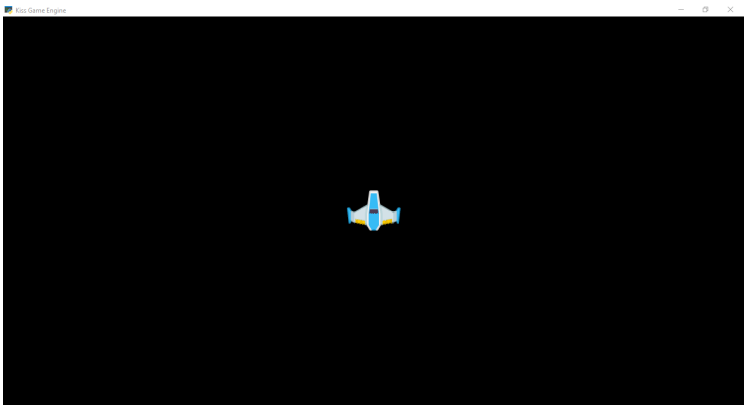
(suite sur la page suivante)

(suite de la page précédente)

```
kge.run(setup)
```

Le résultat ne devrait pas trop vous surprendre :

Résultat :



Basiquement, ce que fait le programme c'est d'ajouter une image à la `scene`, cela représente un niveau dans le jeu. Tous les éléments qui doivent être visibles, sont ajoutés dans la `scene`.

Déplacer le joueur

Afin de prendre le contrôle de notre vaisseau, il va falloir créer une nouvelle classe héritant de `Sprite` afin de personnaliser son comportement :

```
class Player(Sprite):
    def __init__(self):
        self.image = Image('player.png')
```

Donc notre fichier `game.py` devient :

```
import kge
from kge import *

class Player(Sprite):
    def __init__(self):
        self.image = Image('player.png')

def setup(scene):
    scene.add(Player())

kge.run(setup)
```

Si vous relancez le programme, vous ne constaterez aucun changement, mais là nous avons la base pour ajouter des comportements à notre vaisseau. Pour ce faire, définir notre logique dans la fonction `on_update` du `Player` que nous devons créer, sachez que vous allez définir presque tous les comportements de vos entités dans cette fonction :

```
class Player(Sprite):
    def __init__(self):
        self.image = Image('player.png')
```

(suite sur la page suivante)

(suite de la page précédente)

```
    def on_update(self, event: Update, dispatch):
        pass
```

Cette fonction prend deux arguments en paramètres, `event` contient certains attributs importants que nous verrons plus tard et `dispatch` qui est une fonction que vous n'aurez pas tout le temps à utiliser. pour plus d'informations, veuillez lire la documentation dans la section **'Evènements'_**.

Ainsi, ajoutons un peu de code pour gérer les entrées du clavier, pour cela **KGE** vient avec la classe `Inputs` qui permet de gérer facilement cela :

`game.py` :

```
import kge
from kge import *

class Player(Sprite):
    def __init__(self):
        self.image = Image('player.png')

    def on_update(self, event: Update, dispatch):
        if Inputs.get_key_down(Keys.Left):
            # déplacer à gauche
            self.position += Vector.Left()

        elif Inputs.get_key_down(Keys.Right):
            # déplacer à droite
            self.position += Vector.Right()

def setup(scene):
    scene.add(Player())

kge.run(setup)
```

Lancer le programme et vous devez avoir pris le contrôle du joueur ! ! Vous devez avoir remarqué que le joueur était bien trop rapide ! Ceci est tout à fait normal, pour régler ce problème, il va falloir faire quelques modifications à notre programme :

```
# ... Les différents 'import'

class Player(Sprite):
    # ... Code d'initialisation

    def on_update(self, event: Update, dispatch):
        if Inputs.get_key_down(Keys.Left):
            # déplacer à gauche
            # Multiplication par 'delta_time'
            self.position += Vector.Left() * event.delta_time

        elif Inputs.get_key_down(Keys.Right):
            # déplacer à droite
            self.position += Vector.Right() * event.delta_time
```

Tout marche ! ! Mais le joueur est bien trop lent ! ! Pour cela il va falloir définir une vitesse de déplacement pour le joueur. Ainsi notre code se voit modifié encore :

```
# ... Les différents 'import'
```

(suite sur la page suivante)

(suite de la page précédente)

```
class Player(Sprite):
    def __init__(self):
        # ... Code d'initialisation

        # Ajouter la variable pour la vitesse
        self.speed = 5

    def on_update(self, event: Update, dispatch):
        if Inputs.get_key_down(Keys.Left):
            # déplacer à gauche
            # Multiplication par 'delta_time'
            # Multiplication par la vitesse
            self.position += Vector.Left() * event.delta_time * self.speed

        elif Inputs.get_key_down(Keys.Right):
            # déplacer à droite
            self.position += Vector.Right() * event.delta_time * self.speed
```

Cette fois-ci notre vaisseau devrait pouvoir se déplacer à une vitesse convenable. Soyez libre de jouer avec la valeur de la vitesse afin de voir l'effet que ça pourrait produire. A ce stade, notre fichier devrait ressembler à ceci :

`game.py` :

```
import kge
from kge import *

class Player(Sprite):
    def __init__(self):
        self.image = Image('player.png')
        self.speed = 5

    def on_update(self, event: Update, dispatch):
        if Inputs.get_key_down(Keys.Left):
            # déplacer à gauche
            # Multiplication par 'delta_time'
            # Multiplication par la vitesse
            self.position += Vector.Left() * event.delta_time * self.speed

        elif Inputs.get_key_down(Keys.Right):
            # déplacer à droite
            self.position += Vector.Right() * event.delta_time * self.speed


def setup(scene):
    scene.add(Player())

kge.run(setup)
```

Ajoutons quelques enemis

Tout bon jeu, nécessite un enemi à détruire pour être fun, nous allons les ajouter. Pour cela, nous allons ajouter quelques images :

Dans notre code, nous ajoutons l'enemi :

```
import kge
from kge import *

class Enemy(Sprite):
    def __init__(self):
        self.image = Image('enemy.png')

        # Ajout du composant permettant de répondre aux collisions
        self.addComponent(
            BoxCollider(sensor=True)
        )

    def on_collision_enter(self, event, dispatch):
        # Si l'enemi est touché, le détruire
        self.destroy()
```

Ce code permet de créer un enemi basique qui lorsqu'il est touché par quelque chose se détruit. Nous lui avons ajouté un composant `BoxCollider` afin de répondre aux collisions, que nous avons mis à `sensor=True` afin de ne pas repousser les collisions. Nous avons aussi mis le code de destruction de l'enemi dans la fonction `on_collision_enter` afin d'être exécuté lorsque le projectile rentrera en collision avec l'enemi.

Note : Vous pouvez remarquer que la signature est la même que la fonction `on_update`.

Ajoutons du code pour le projectile :

```
class Bullet(Sprite):
    def __init__(self):
        self.image = Image('bullet.png')

        # Ajout d'un composant pour le déplacement par la physique
        rb = kge.RigidBody()

        # Empêcher la gravité de faire effet
        rb.gravity_scale = 0

        # Vitesse
        rb.velocity = Vector.Up() * 2

        self.addComponent(rb)

        # Ajout du composant pour répondre aux collisions
        self.addComponent(
```

(suite sur la page suivante)

(suite de la page précédente)

```
            BoxCollider(sensor=True)
        )

    def on_collision_enter(self, event, dispatch):
        # Détruire aussi le projectile, s'il touche l'enemi
        self.destroy()
```

Afin de tirer le projectile, il faut appuyer le bouton espace donc :

```
class Player(Sprite):
    #... Code d'initialisation

    def on_update(self, event: Update, dispatch):
        #... Code pour se déplacer

    def on_key_down(self, event, dispatch):
        # Nous pouvons aussi gérer les saisies de clavier dans cette méthode
        if event.key is Keys.Space:
            # Tirer !!
            event.scene.add(Bullet(), position=self.position)
```

Vous pouvez remarquer que l'argument `event` a pour attribut `scene`, qui représente le niveau actuel. la fonction `add` prend aussi en argument la position où mettre l'élément. Ajoutons donc les enemis à notre niveau et :

```
def setup(scene):
    scene.add(Player())

    # Ajout des enemis
    for i in range(-7, 9, 2):
        scene.add(Enemy(), position=Vector(i, 4))
```

D'où le code complet :

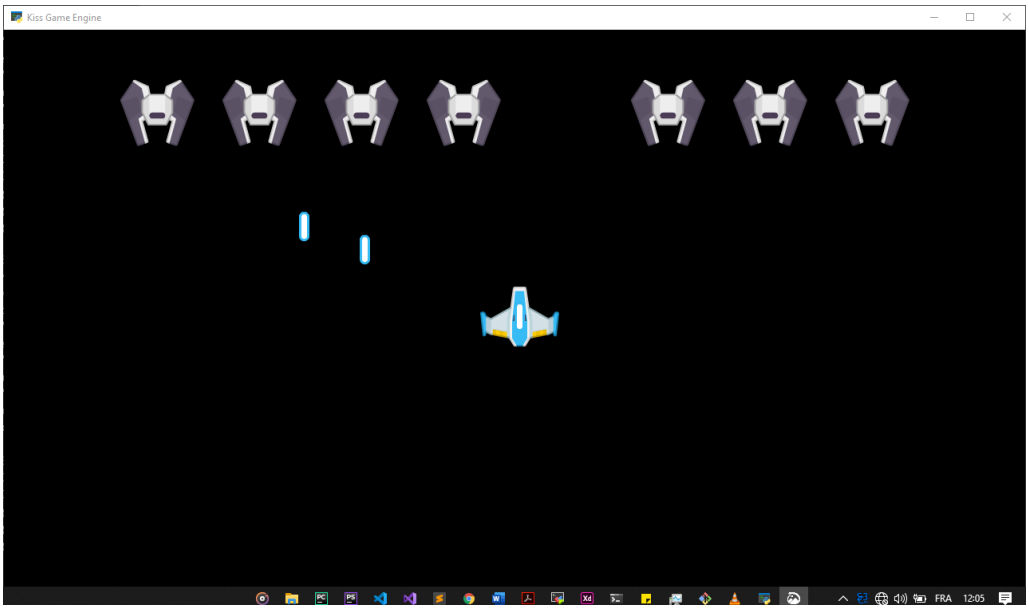
`game.py` :

```
import kge
from kge import *

class Enemy(Sprite):
    def __init__(self):
        self.image = Image('enemy.png')

        # Ajout du composant permettant de répondre aux collisions
        self.addComponent(
            BoxCollider(sensor=True)
        )

    def on_collision_enter(self, event, dispatch):
        # Si l'enemi est touché, le détruire
        self.destroy()


class Bullet(Sprite):
    def __init__(self):
        self.image = Image('bullet.png')

        # Ajout d'un composant pour le déplacement par la physique
```

(suite sur la page suivante)

(suite de la page précédente)

```
        rb = kge.RigidBody()

        # Empêcher la gravité de faire effet
        rb.gravity_scale = 0

        # Vitesse
        rb.velocity = Vector.Up() * 2

        self.addComponent(rb)

        # Ajout du composant pour répondre aux collisions
        self.addComponent(
            BoxCollider(sensor=True)
        )

    def on_collision_enter(self, event, dispatch):
        # Détruire aussi le projectile, s'il touche l'enemi
        self.destroy()

class Player(Sprite):
    def __init__(self):
        self.image = Image('player.png')
        self.speed = 5

    def on_update(self, event: Update, dispatch):
        if Inputs.get_key_down(Keys.Left):
            # déplacer à gauche
            # Multiplication par 'delta_time'
            # Multiplication par la vitesse
            self.position += Vector.Left() * event.delta_time * self.speed

        elif Inputs.get_key_down(Keys.Right):
            # déplacer à droite
            self.position += Vector.Right() * event.delta_time * self.speed

    def on_key_down(self, event, dispatch):
        # Nous pouvons aussi gérer les saisies de clavier dans cette méthode
        if event.key is Keys.Space:
            # Tirer !!
            event.scene.add(Bullet(), position=self.position)


def setup(scene):
    scene.add(Player())

    # Ajout des enemis
    for i in range(-7, 9, 2):
        scene.add(Enemy(), position=Vector(i, 4))

kge.run(setup)
```



Voilà ! Nous avons un jeu fonctionnel. Si vous ressortez de ce tutoriel sans avoir compris certaines parties, ne vous en faites pas, nous allons les aborder dans les prochains chapitres.

Un petit plus pour pouvoir y voir plus clair et afficher les boîtes de collisions, il suffit d'ajouter ces lignes à votre code :

`game.py` :

```
import kge
from kge import *
# Importer cette bibliothèque pour le déboggage
import logging

#... Code du jeu

kge.run(setup,
        # Ajouter cette ligne
        log_level=logging.DEBUG
        )
```

1.2 Manuel d'utilisation du moteur de jeu KISS GAME ENGINE

1.2.1 Les concepts

Le moteur de jeu KGE dispose de plusieurs concepts qui vous permettront d'être à l'aise avec la programmation de jeux-vidéos 2D.

1.2.2 Cette partie est encore en construction.

1.3 Référence des classes de kge

Ci-dessous la liste des classes du moteur.