CODING CLUB

EPITECH LYON

Astroders

VERSION 1.2.0

DIVERSITY by EPITECH

25 Juin 7486.

Le monde a connu de grandes avancées technologiques, mais la planète Terre voit disparaître ses jungles et déserts, ses vents et marées, ses lacs et océans, et sa lumière naturelle provenant du soleil.

Les 12 signes du zodiaque, entités protégeant autrefois la Terre et dont chaque être humain est sous la protection, ont été capturés par les 4 éléments primordiaux venus se venger de l'humanité : l'air, le feu, la terre et l'eau.

C'est alors qu'une dernière entité, le Signe du Serpentaire, 13e signe du zodiaque, défie les 4 éléments afin de libérer les 12 autres Signes. Vous êtes choisi pour l'aider à faire face à cette menace. Serez-vous de taille ?

I : Les préparatifs

Le Serpentaire vous donne 4 fichiers : Classes.py, Astroders.py, Enemies.py et Player.py.

Il a d'abord besoin que tu l'aides à rejoindre l'espace !

Pour le moment, le contenu du programme permet de créer une fenêtre, de créer le joueur, de pouvoir arrêter le programme avec la touche « Echap », et enfin d'afficher cette fenêtre avec une image en fond.

Pour tester cela, effectue la commande `python3 Astroders.py` pour démarrer le programme.

On va commencer par aller dans le fichier « Player.py » afin de compléter la classe « Player » :

```
...
class Player(PlayerClass):
    def __init__(self):
        super().__init__()
        self.bullets = []
        self.bullet_cooldown = 0
        # code here
...
```

Ces lignes de codes permettent de créer une classe « Player » qui hérite de la classe « PlayerClass » déjà présente dans le fichier « Classes.py ». Hérité d'une classe signifie que tous les attributs et les méthodes de la classe parent appartiennent aussi à la classe enfant. Il nous montre aussi comment créer les attributs **bullets** et **bullet_cooldown**. Maintenant à vous de créer les attributs *hp*, *x* et *y* qui auront respectivement pour valeur 3, 0 et 800.

Maintenant que cela est fait, nous allons devoir créer la méthode permettant l'affichage du joueur, de ses points de vie ainsi que ses projectiles (pour lancer un projectile il faut appuyer sur la touche espace). Pour cela il va falloir compléter le code suivant dans le fichier « Player.py » :

```
...
def display(self, game):
    self.costume = (self.costume + 1) % (len(self.costumes) * 5)
    for bullet in self.bullets:
        # add a line here
    hp_text = game.font.render("HP: %d" % self.hp, True, (255, 255, 25
    # here too
    game.screen.blit(self.costumes[self.costume // 5], (self.x, self.y
...
```

Les lignes que vous avez à compléter doivent permettre l'affichage de chaque projectile et du texte des HP. Pour cela vous allez devoir utiliser la méthode blite de Pygame qui fonctionne que cela :

```
screen.blit(element_to_display, (x, y))
```

Pour que le code que vous venez de faire soit exécuter il faut rajouter cette ligne dans le fichier « Astroders.py » :

```
...
def display_game_elements(game):
    game.screen.blit(game.background, (0, 0))
    game.player.display(game) # this one
    pygame.display.update()
...
```

Une fois que notre joueur et ces projectiles peuvent s'afficher, il va falloir permettre à notre joueur de se déplacer. Pour cela nous allons devoir créer des méthodes pour chacun des mouvements possibles du joueur à la suite du fichier « Player.py »

Tu peux remarquer qu'il existe déjà une méthode permettant le déplacement vers la gauche :

```
...
def move_left(self):
    if self.x > 0:
        self.x -= self.player_speed
...
```

Tu auras remarqué que le mouvement ne s'effectue que lorsque la position du joueur à supérieur à 0 sur l'axe x. Pour les autres mouvements tu vas devoir appliquer la même logique mais avec des valeurs différentes :

- move_right, ssi la position x du joueur est strictement inférieur à 1150.
- move_up, ssi la position y du joueur est strictement supérieur à 600.
- move_down, ssi la position y du joueur est strictement inférieur à 800

Maintenant que nous avons fait les méthodes associées à chacun des mouvements, nous devons gérer les évènements relatifs à ceux-ci affins que nous puissions déplacer le joueur avec les touches du clavier. Pour cela nous allons modifier la fonction « key_inputs » du fichier « Astroders.py ».

```
...
def key_inputs(player):
    keys = pygame.key.get_pressed()
    if keys[K_ESCAPE]:
        exit()
    if keys[K_SPACE]:
        player.shoot()
    # code here
...
```

Il ne vous reste plus qu'à implémenter les cas qu'il manque. Tu peux maintenant relancer le programme, et appuyer sur la flèche de droite. Le Serpentaire devrait maintenant bouger !

II. À l'attaque !

Bravo ! Le Serpentaire peut maintenant se déplacer librement dans la fenêtre. Mais pour l'instant aucun ennemis en vue...

Pour faire apparaitre les ennemis il vous suffit de rajouter la ligne suivante dans le fichier « Astroders.py » :

```
...
def display_game_elements(game):
    game.screen.blit(game.background, (0, 0))
    game.player.display(game)
    game.enemies.display(game) # this one
    pygame.display.update()
...
```

Une fois cela fait, il va falloir permettre aux ennemis de se déplacer en rejoutant la ligne suivante :

```
def update_game_elements(game):
    game.clock.tick(60)
    game.player.update()
    game.enemies.update(game) # this one
```

Maintenant, il va falloir permettre aux ennemis d'attaquer. Pour cela on va créer une fonction dans le fichier « Astroders.py » :

```
...
def attack_on_tick(game):
...
```

Cette fonction devra : - incrémenter la valeur de *game.frame* de 1 - ssi *game.frame* est strictement supérieur à 50, alors *game.frame* est remis à zéro et on appel *game.enemies.attack(game.screen)*

Une fois cela fait, il ne faut pas oublier d'appeler cette nouvelle fonction dans la boucle du jeu :

```
...
while game.running:
    check_events(game)
    attack_on_tick(game) # here
    update_game_elements(game)
    display_game_elements(game)
...
```

III. Attention aux choques

Mais vous arez remarqué que tes projectiles, comme ceux de tes ennemis n'atteignent leur cible. Pour qu'ils soient pris en compte, il va falloir créer une méthode **check_attacks** dans le fichier « Enemies.py »

```
...
def check_attacks(self, player):
    for attack in self.attacks:
        # code here
...
```

Maintenant à vous de compléter la boucle pour qu'à chaque *attack*, ces conditions soient vérifiées :

- ssi la coordonnée y de l'*attack* est strictement supérieur à 850, alors on la supprime de la liste des attack (c'est-à-dire de *attacks*). Et on passe au tour de boucle suivant.
- ssi la condition suivante est vrai, alors un supprime l'*attack* de la liste *attacks*. Et on passe au tour de boucle suivant.

```
if attack.x <= player.x + 100 and player.x <= attack.x + 50
and attack.y + 50 >= player.y and player.y + 100 >= attack.y:
```

Il vous suffit maintenant d'appeler la méthode que vous venez de créer dans le fichier « Astroders.py » :

```
...
def check_hit_boxes(game):
    game.enemies.check_attacks(game.player)
...
while game.running:
    check_events(game)
    attack_on_tick(game)
    update_game_elements(game)
    check_hit_boxes(game)
...
```

Vous pouvez maintenant démarrer à nouveau le programme, et le joueur devrait normalement perdre des points de vie lorsqu'une attaque l'atteint, mais ces attaques à lui n'ont toujours aucun effet !

En effet, nous ne vérifions pas que les projectiles du joueur atteignent leur cible. Pour cela on va compléter la boucle contenue dans la fonction **check_bullets** du fichier « Enemies.py »

```
...
 def check_bullets(self, width, bullets):
    for line_index, line in enumerate(self.enemies):
        for enemie_index, enemy in enumerate(line):
            if not enemy.alive:
                continue
            top = self.enemies_top + line_index * 80
            left = (self.lines_pos[line_index] + enemie_index * 80) %
            right = (self.lines_pos[line_index] + enemie_index * 80 +
            bottom = self.enemies_top + line_index * 80 + 50
            for bullet in bullets:
                # code here
...
```

Vous devrez ajouter les conditions suivantes : - ssi la coordonnée y de *bullet* est strictement inférieure à la valeur de *top* **OU** si cette coordonnée est strictement supérieur à la valeur de *bottom*, alors passer au prochain tour de boucle. - ssi la coordonnée x de *bullet* est supérieur ou égale à la valeur de *left* **ET** que qu'elle est aussi inférrieur ou égale à la valeur de *right*, alors on met l'attribue *alive* de l'*enemy* à False et on suprime cet élément du tableau des projectiles

Avant de relancer notre programme pour vérifier s'il fonctionne, n'oubliez pas d'appeler cette méthode dans le fichier « Astroders.py » :

```
...
def check_hit_boxes(game):
    game.enemies.check_attacks(game.player)
    game.enemies.check_bullets(game.window_width, game.player.bullets)
...
```

IV. Pour la victoire !

Maintenant que le joueur peut perdre des points de vie et détruire ses ennemis, il faut mettre en place les conditions de victoire et de défaite. Pour ce faire nous allons de nouveau modifier le fichier « Astroders.py » pour y ajouter la fonction suivante :

```
...
def check_end_conditions(game):
...
```

À vous maintenant d'y implémenter les conditions suivantes :

- si le nombre de points de vie du joueur (*game.player.hp*) est inférieur ou égale à 0 **OU** si la valeur de *game.enemies.enemies_top* est strictement supérieur à *game.window_height* - 150, alors on met la variable *game.running* à False.
- si la valeur de *game.enemies.count_enemies()* est strictement égale à 0, alors met la variable *game.running* à False et la variable *game.win* à True.

Bien évidemment, on n'oubliera pas d'appeler notre nouvelle fonction dans la boucle su jeu :

```
while game.running:
    check_events(game)
    attack_on_tick(game)
    update_game_elements(game)
    check_hit_boxes(game)
    display_game_elements(game)
    check_end_conditions(game) # don't forget me
```

Maintenant que tu peux gagner ou perdre, à toi de mettre en place des messages de victoire ou de défaite :

```
...
while game.running:
    check_events(game)
    attack_on_tick(game)
    update_game_elements(game)
    check_hit_boxes(game)
    display_game_elements(game)
    check_end_conditions(game)
if game.win:
    # add a line here
else:
    # here too
...
```

Il vous suffit alors d'appeler la fonction *print()* qui permet d'afficher un message :

```
print("Hello you !")
```

Output :

```
Hello you !
```

V. The end

Tout d'abord : bravo !

Maintenant, libre à toi de modifier le jeu selon tes désirs ! Tu as normalement dû apprendre certaines choses, et tu peux modifier à ta guise des parties du programme ou ajouter de nouvelles choses facilement comme du texte, de nouveaux projectiles, etc... Voici quelques exemples d'ajouts sympathiques ! :

- Faire en sorte que chaque élément ajoute un bonus selon son type, comme plus de PV, augmentation de la vitesse de déplacement, augmentation de la vitesse de tir, et un bonus aléatoire entre ces trois-là
- De la musique et des sons.
- Un score et un écran de Game Over.
- Un menu
- De nouveaux ennemis ou personnages jouables.

N'hésite surtout pas à demander de l'aide à un Cobra qi tu as du mal à appliquer certaines de tes idées !