Space Invaders - Documentation et Projets

Ce dépôt contient la documentation complète, le code source final et le projet de code à compléter pour le jeu "Space Invaders" créé avec Godot 3 et Python. Ce projet a été conçu comme un outil éducatif pour aider les enfants qui ont déjà une expérience avec Scratch à faire la transition vers Python.

Le jeu est une version simplifiée du classique Space Invaders, avec un vaisseau qui peut se déplacer à l'écran et tirer sur des météorites qui tombent. Le but du joueur est de détruire les météorites avant qu'elles n'atteignent le bas de l'écran.

Structure du Dépôt

- `documentation/` - Contient la documentation du projet, écrite en HTML et CSS. Elle explique comment installer Godot et Python, comment créer un nouveau projet dans Godot, et toutes les étapes pour créer le jeu de A à Z.
- `projet_final/` - Contient le code source complet du jeu.
- `projet_a_completer/` - Contient le jeu avec des parties du code à compléter pour les apprenants.

Utilisation

Pour commencer, dirigez-vous vers la partie installation de la documentation pour continuer.

Installation

Cette page vous guide à travers les étapes pour installer et configurer les outils nécessaires pour travailler sur le projet Space Invaders. Pour plus de détails sur l'installation de godot / python vous pouvez consulter la page github du projet à cette adresse.

La manière la plus simple d'installer Godot est d'utiliser l'AssetLib de Godot. Une fois Godot installé, ouvrez-le et cliquez sur l'onglet AssetLib. Recherchez "PythonScript" et cliquez sur le bouton "Installer" pour installer le plugin. Cependant cette méthode ne fonctionne pas toujours, c'est pourquoi vous pouvez suivre les étapes suivantes pour installer Godot qui consiste à compiler Godot à partir des sources.

Étape 1 : Installation de Python

La première étape consiste à installer Python sur votre machine. Vous pouvez le télécharger à partir de la commande suivante sous Linux : `$ apt install python3 python3-pip python3-venv build-essential`.

Après avoir téléchargé et installé Python, vous pouvez vérifier que l'installation a réussi en ouvrant un terminal et en tapant : `python --version` ou `python3 --version`

Étape 2 : Installation de Godot

Une fois Python installé, vous pouvez installer Godot. Pour ce faire, toujours sous Linux, cloner le projet avec la commande suivante : `git clone https://github.com/touilleMan/godot-python.git`

Ensuite, vous pouvez installer Godot en exécutant les commandes suivantes :
- `$ cd <godot-python-dir>`
- `$ python3 -m venv venv`
- `$ source venv/bin/activate`
- `$ pip install -r requirements.txt`
- `$ scons platform=x11-64 release` - `$ scons platform=x11-64 test`

Une fois l'installation terminée, dans le dossier build/x11-64/platforms se trouve un fichier Godot_vx.x.x-stable_x11.64. Lancez-le pour démarrer Godot.

Étape 3 : Installation du projet Space Invaders

Une fois Python et Godot installés, vous êtes prêt à commencer à travailler sur le projet Space Invaders. Il vous suffit de cloner le projet avec la commande suivante :
`git clone https://github.com/hugo-hamon/SpaceInvaders.git`

Le projet peut être utilisé de plusieurs manières :
- En utilisant l'éditeur Godot pour modifier / exécuter le projet déjà existant dans le dossier projet_final.
- En complétant le projet existant dans le dossier projet_a_completer et grâce à la documentation dans l'onglet "Compléter un jeu Existant".
- En créant un nouveau projet à partir de zéro en suivant la documentation dans l'onglet "Création du Projet".

Construction du Jeu

La partie "Construction du Jeu" explique le processus de développement des différentes fonctionnalités du jeu Space Invaders comprenant les météorites, les projectiles, le vaisseau spatial et le code principal du jeu. En suivant les instructions de cette section, vous serez en mesure de créer un Space Invaders complet et fonctionnel.

Développement des Météorites

Dans cette section, on va compléter les fonctionnalités des météorites, qui sont les ennemis du jeu. Dans un premier temps, on va compléter l'arborescence de la scène, puis on va développer le script qui gère les collisions avec les projectiles et le vaisseau spatial.

Pour commencer ajoutez deux nodes à la scène :

- Une node animatedSprite pour représenter l'apparence visuelle de la météorite.
- Une node collisionShape2D pour représenter la zone de collision de la météorite.

Ensuite choisissez pour la node collisionShape2D une forme capsule et ajustez la taille de la zone de collision pour qu'elle corresponde à la taille de la météorite. Pour l'animatedSprite clicker dessus ensuite frame, ajouter SpriteFrames, ensuite cliquer dessus et ajouter une image de météorite présente dans le dossier assets.

Une fois ca fait diriger vous vers le script de la météorite et ajouter le code suivant dans la classe:

```
@exposed
class Mob(RigidBody2D):

    def _ready(self):
        pass

    def _process(self, delta):
        """Fonction appelée à chaque frame"""
        if collision := self.get_colliding_bodies() or self.position.y > 1000:
            self.get_parent().game_over()
            self.queue_free()
```

Il permet de vérifier si la météorite est en collision avec un autre objet ou si elle est sortie de l'écran. Si c'est le cas, la fonction game_over() est appelée pour terminer la partie et la météorite est supprimée.

Développement des Projectiles

Pour cette partie, on va compléter les fonctionnalités des projectiles, qui sont utilisés par le vaisseau spatial pour détruire les météorites. Les étapes sont exactement les mêmes que pour les météorites :

- Ajout de collisionShape2D pour représenter la zone de collision du projectile
- Ajout de animatedSprite pour représenter l'apparence visuelle du projectile

Une fois ca fait diriger vous vers le script du projectile et ajouter le code suivant dans la classe:

```
@exposed
class Projectile(Area2D):

 speed = export(int, 500)

 def _ready(self):
  self.spaceship = self.get_parent().get_node("Spaceship")

 def _physics_process(self, delta):
  self.position += Vector2(0, -self.speed * delta)
  if self.position.y < 0:
   self.queue_free()

 def _process(self, delta):
  # Check collision
  for body in self.get_overlapping_bodies():
   if body.is_in_group("Enemy"):
    self.spaceship.score += 1
    self.queue_free()
    body.queue_free()
    break
```

Ce script permet de déplacer le projectile vers le haut de l'écran et de vérifier si le projectile est en collision avec une météorite. Si c'est le cas, le score du joueur est incrémenté et la météorite est supprimée. De plus on remarque le groupe Enemy, il faut donc ajouter ce groupe aux météorites. Pour ce faire cliquer sur la météorite, ensuite cliquer sur Node, ensuite cliquer sur groupes et ajouter le groupe Enemy.

Développement du Vaisseau Spatial

Pour cette partie, on va compléter les fonctionnalités du vaisseau spatial, qui est le personnage contrôlé par le joueur. Les étapes sont un peu différentes des précédentes. Cependant l'arborescence de la scène reste la même à une exception près, on va ajouter un node Timer pour gérer le délai entre les tirs du vaisseau spatial. Les étapes sont donc les suivantes :

- Ajout de collisionShape2D pour représenter la zone de collision du vaisseau spatial
- Ajout de animatedSprite pour représenter l'apparence visuelle du vaisseau spatial
- Ajout d'une node timer que l'on renomme en "ProjectileTimer"

Une fois ca fait comme pour avec les autres objets diriger vous vers le script du vaisseau spatial et ajouter le code suivant dans la classe:

```
from godot import exposed, export
from godot import *
import random

@exposed
class Spaceship(KinematicBody2D):
```

```
velocity = export(int, 300)
direction = export(Vector2, Vector2(0, 0))
time = export(float, 0.5)
score = export(int, 0)
sprite_size = Vector2(0, 0)

def _ready(self):
    self.screen_size = self.get_viewport().size
    self.sprite = self.get_node("AnimatedSprite")
    self.sprite_size = self.get_sprite_size()

    self.projectile_scene = ResourceLoader.load("res://scenes/Projectile.tscn")

    self.set_position(Vector2(self.screen_size.x / 2, self.screen_size.y - self.sprite_size.y))
    self.projectile_timer = self.get_node("ProjectileTimer")
    self.projectile_timer.start()

def get_sprite_size(self):
    sprite_size = self.sprite.get_sprite_frames().get_frame("Idle", 0).get_size()
    scale = self.sprite.get_transform().get_scale()
    return Vector2(sprite_size.x * scale.x, sprite_size.y * scale.y)

def _physics_process(self, delta):
    self.direction = Vector2(0, 0)

    self.process_input()
    self.check_screen_colision()

    self.direction = self.move_and_slide(self.direction, Vector2.UP)

def process_input(self) -> None:
    if Input.is_action_pressed("right"):
        self.direction.x = self.velocity
    if Input.is_action_pressed("left"):
        self.direction.x = -self.velocity
    if Input.is_action_pressed("up"):
        self.direction.y = -self.velocity
    if Input.is_action_pressed("down"):
        self.direction.y = self.velocity
    if Input.is_action_pressed("shoot"):
        if self.projectile_timer.is_stopped():
            self.shoot()
            self.projectile_timer.start(self.time)

def check_screen_colision(self) -> None:
    if self.get_position().x > self.screen_size.x - self.sprite_size.x / 2:
        self.set_position(Vector2(self.screen_size.x - self.sprite_size.x / 2, self.get_position().y))
    if self.get_position().x < self.sprite_size.x / 2:
        self.set_position(Vector2(self.sprite_size.x / 2, self.get_position().y))
    if self.get_position().y > self.screen_size.y - self.sprite_size.y / 2:
        self.set_position(Vector2(self.get_position().x, self.screen_size.y - self.sprite_size.y / 2))
    if self.get_position().y < self.sprite_size.y / 2:
        self.set_position(Vector2(self.get_position().x, self.sprite_size.y / 2))

def shoot(self):
    b = self.projectile_scene.instance()
    self.owner.add_child(b)
    b.set_position(self.get_position() + Vector2(0, -40))
```

Ce script permet de déplacer le vaisseau spatial dans les quatre directions et de tirer des projectiles vers le haut de l'écran. On a ajouter un timer pour gérer le délai entre les tirs. On a

aussi ajouter une variable score qui sera incrémenter à chaque fois qu'une météorite est détruite.

Développement du Code Principal

L'on va maintenant se concentrer sur le développement du code principal du jeu. Ce code va gérer les mécanismes de jeu tels que la création des météorites, la gestion des collisions et le score. Voici les étapes pour pouvoir y arriver :

- Ajouter un timer et nommez le "MeteoriteTimer". Il nous permettra de créer des météorites à intervalle régulier.
- En sélectionnant le background dans le dossier assets, glisser le dans la scène de manière à ce qu'il recouvre toute la scène.
- Ajouter une node "Label" et nommez la "ScoreLabel". Elle nous permettra d'afficher le score du joueur. Vous pouvez modifier la police et la taille du texte dans les propriétés de la node. De plus déplacer la en haut à gauche de l'écran et modifier le texte à "Score : 0".
- Finalement, à côté du petit plus en dessous de scènes il y a un bouton instancier une autre scène, cliquer dessus et ajouter la scène Spaceship.

Maintenant que nous avons tous les éléments nécessaires pour le jeu, nous allons pouvoir commencer à développer le code principal. Voici le code pour le script principal :

```
from godot import exposed, export
from godot import *
import random

@exposed
class Level(Node):

    def _ready(self):
        self.mob_scene = ResourceLoader.load("res://scenes/Meteorite.tscn")
        self.score_label = self.get_node("ScoreLabel")
        self.player = self.get_node("Spaceship")
        print(self.player)
        self.screen_size = self.get_viewport().size
        self.spawn_timer = self.get_node("MeteoriteTimer")
        self.spawn_timer.connect("timeout", self, "_on_spawn_timer_timeout")
        self.spawn_timer.start()

    def _process(self, delta):
        self.score_label.set_text(f"Score: {self.player.score}")

    def _on_spawn_timer_timeout(self) -> None:
        self.spawn_mob()
        new_time = max(0.5, -1 / 50 * self.player.score + 2)
        self.spawn_timer.set_wait_time(random.uniform(new_time, new_time + 0.25))

    def spawn_mob(self) -> None:
        mob = self.mob_scene.instance()
        self.add_child(mob)
        position = Vector2(random.uniform(50, self.screen_size.x - 50), -100)
        mob.set_position(position)
        mob.set_linear_velocity(Vector2(0, random.randint(150, 200)))

    def game_over(self) -> None:
        # stop the spawn timer
        self.spawn_timer.stop()
        # delete all mobs
        for mob in self.get_children():
            if str(mob.get_name()) == "Meteorite" or str(mob.get_name()).startswith("@Meteorite"):
                mob.queue_free()
```

```
    # resete the score
    self.player.score = 0
    # move the player back to the start position
    self.player.set_position(Vector2(self.screen_size.x / 2, self.screen_size.y - 100))
    # start the spawn timer
    self.spawn_timer.start()
```

Ce script permet de créer des météorites à intervalle régulier. Il permet aussi de gérer le score du joueur et de réinitialiser le jeu lorsque le joueur perd.

Quelques ajustements

Maintenant que tout est en place quelque ajustement sont encore à faire. Premièrement, dirigez vous vers la scène Meteorite et modifier les choses suivante:

- RigidBody2D:
  - Gravity Scale = 0 (Car le mouvement des météorites est géré par le script)
  - contact monitor = true (Pour détecter les collisions)
  - contacts Report = 2 (Pour détecter les collisions avec le vaisseau spatial et les projectiles)
  - collision / Mask : décocher tout pour que les météorites ne puissent pas entrer en collision avec elle même.

Maintenant dirigez vous vers la scène Spaceship et modifier les choses suivante:

- ProjectileTimer:
  - Wait Time = 0.5 (Pour que le joueur ne puisse pas tirer trop rapidement)
  - One shot = True
- AnimatedSprite:
  - Renommer l'animation "Default" en "Idle".

Une fois ces ajustements fait, vous pouvez tester votre jeu. Dans le cas où vous auriez des erreurs, vérifier que vous avez bien suivi toutes les étapes. Si cela ne fonctionne toujours pas, vous pouvez comparer avec le projet final.

Documentation du jeu Space Invaders

Création du Projet

La partie "Création du Projet" explique comment configurer un nouveau projet Space Invaders dans l'éditeur Godot, créer les différentes scènes nécessaires et ajouter des scripts pour les fonctionnalités du jeu.

Explication

Avant de commencer, il est important de comprendre les différents éléments du projet et à quoi ils servent :

- **Scènes**: Les scènes sont les éléments principaux du jeu. Dans Space Invaders, on a besoin de scènes pour le vaisseau spatial, les météorites, les projectiles, etc.
- **Scripts**: Les scripts sont utilisés pour ajouter des fonctionnalités et des comportements aux objets du jeu. On aura besoin de scripts pour gérer les mouvements du vaisseau spatial, la création des météorites, la détection des collisions, etc.
- **Taille de la fenêtre**: Il est important de paramétrer la taille de la fenêtre du jeu pour qu'elle convienne aux différents sprites. On définira une taille qui correspond à la résolution attendue du jeu.
- **Contrôles**: On aura besoin de définir les contrôles pour les mouvements du vaisseau spatial. Par exemple on pourra utiliser les flèches pour se déplacer ou bien ZQSD.

Création des Scènes et Scripts

Dans les étapes suivantes, on va créer les scènes et les scripts nécessaires pour le jeu.

1. Ouvrez Godot et créez un nouveau projet en sélectionnant "Nouveau Projet" dans le menu principal. Et choisissez un nom pour le projet, par exemple "Space Invaders". Une fois le projet créé, rendez-vous dans le dossier du projet et créez les dossiers suivants : assets, scenes et scripts.
2. Créez une nouvelle scène en sélectionnant "Node2D" dans les scènes proposées. Une fois créée, renommez la scène en Main et enregistrez-la dans le dossier scenes du projet. Cette scène sera la scène principale du jeu. Elle contiendra tous les autres éléments du jeu commme les météorites, les projectiles, etc.
3. Créez maintenant une scène pour le vaisseau spatial, en sélectionnant en haut à gauche scène / Nouvelle Scène / Node2D / Autre Noeud / KinematicBody2D. Une fois créée, renommez la scène en Spaceship et enregistrez-la dans le dossier scenes du projet. Cette scène contiendra le vaisseau spatial.
4. De la même manière, créez une scène pour les météorites avec comme nom Meteorite et avec la Node Rigidbody2D. Finallement, créez une scène pour les projectiles avec comme nom Projectile et avec la Node Area2D.
5. Avant de passer à l'étape suivante, en haut diriger vous dans l'onglet AssetLib et chercher le package "PythonScript" et l'installer puis redémarrer Godot. Ce package permet d'utiliser le langage python pour les scripts.
6. Maintenant que les scènes sont créées, il faut créer les scripts pour les fonctionnalités du jeu. Pour cela, sélectionnez une scène, clique droit sur la scène et sélectionnez attacher un script. Sélectionnez le langage python et le dossier scripts. Enregistrez le script avec le même nom que la scène. Par exemple,

pour la scène Spaceship, le script sera Spaceship.py. Faite ca pour toutes les scènes.

Paramétrage du Projet / Contrôles

Pour paramétrer le projet et définir les contrôles du jeu, suivez ces étapes :

1. Dans Godot, accédez aux paramètres du projet en sélectionnant "Paramètres du Projet" dans le menu "Projet".
2. Dans les paramètres du projet, accédez à la section "Display / Window" et définissez la taille de la fenêtre à 700x1000 et décocchez la case "Resizable".
3. Dans la section "Contrôles", définissez les contrôles du jeu en ajoutant des actions pour les mouvements du vaisseau spatial. Ajouter les actions suivantes :
    - up
    - down
    - left
    - right
    - shoot
    - escape

Continuer la construction du jeu

Maintenant que les scènes et les scripts sont créés, il faut continuer la construction du jeu en ajoutant les éléments du jeu dans les scènes et en ajoutant les fonctionnalités dans les scripts. Pour cela dirigez-vous vers la partie "Construction du Jeu".

Documentation du jeu Space Invaders

Compléter un Jeu Existant

Il est possible de coder le jeu space invader en passant par une interface. La liste des commandes sont disponibles ci-dessous. Pour pouvoir les utiliser, il suffit d'ajouter ces commandes / codes dans le fichier edit.py dans la fonction run.

Les étapes à suivre sont les suivantes :

- Gérer les déplacements du vaisseau et les tirs du vaisseau
- Gérer les collisions entre les projectiles et les ennemis ainsi qu'avec les bords de l'écran
- Faire en sorte que si un astéroïde atteint le bas de l'écran, le joueur perd.

Class: Interface

__init__

```
__init__(self, planer) -> None
```

No description !

Move_spaceship

```
Move_spaceship(self, user_input: UserInput) ->
None
```

Déplace le vaisseau dans la direction donnée (up, down, left, right)

Shoot

```
Shoot(self) -> None
```

Tire un projectile

Get_spaceship_position

```
Get_spaceship_position(self) -> Tuple[int,
int]
```

Retourne la position du vaisseau

Get_spaceship_size

```
Get_spaceship_size(self) -> Tuple[int,
int]
```

Retourne la taille du vaisseau

Get_spaceship_speed

```
Get_spaceship_speed(self) -> int
```

Retourne la vitesse du vaisseau

Get_asteroid_position

```
Get_asteroid_position(self) -> List[Tuple[int, int]]
```

Retourne la position de l'astéroïde

Get_screen_size

```
Get_screen_size(self) -> Tuple[int,
int]
```

Retourne la taille de l'écran

Is_input_pressed

```
Is_input_pressed(self, user_input: UserInput) ->
bool
```

Retourne True si la touche direction est pressée, False sinon

Get_score

```
Get_score(self) -> int
```

Retourne le score actuel

Increase_score

```
Increase_score(self) -> None
```

Augmente le score de 1

Spaceship_collides

```
Spaceship_collides(self) -> bool
```

Retourne True si le vaisseau est en collision avec un ennemi, False sinon

Reset_game

```
Reset_game(self) -> None
```

Réinitialise le jeu