

Livret 1.4 Jeu VIDEO 2D PYTHON



Si tu es passionné par les jeux vidéo et que tu as toujours rêvé de créer le tien, alors ce livret pédagogique est fait pour toi. Imagine pouvoir concevoir et programmer ton propre jeu 2D en utilisant Python, et Pygame, une bibliothèque spécialement conçue pour créer des jeux. Nous irons encore plus loin en intégrant un outil puissant de création de niveaux appelé Tiled, qui te permettra de donner vie à tes idées de manière visuelle et intuitive.

Prépare-toi à plonger dans l'univers captivant du développement de jeux vidéo tout en apprenant des concepts de programmation et de conception de jeux. Es-tu prêt à relever le défi et à donner vie à ton imagination à travers la magie de la programmation ? Alors, allons-y et commençons cette aventure ensemble !"

À travers ce livret, tu découvriras les étapes pas à pas pour créer ton propre jeu, et je serai là pour t'accompagner à chaque étape, alors n'hésite pas à te plonger dans ce monde fascinant du développement de jeux vidéo en Python !

Livret 1.4 Jeu VIDEO 2D PYTHON

Table des matières

Mon premier écran	5
Dessiner un sol	7
Créer un joueur	9
Gérer la gravité sur le joueur	11
Gérer la vitesse de déplacement et la force du saut	11
Créer votre première plateforme	12
Comment monter sur la plateforme	13
Créer plusieurs plateformes	14
Modifier les conditions des déplacements	15
Création d'obstacles	16
Gérer la visibilité des objets graphiques entre eux.....	17
Gestion des collisions avec les obstacles.....	18
Animer le joueur	19
Création d'un soleil, objet fixe, illusion déplacement	21
Le logiciel TILED	22
Création d'un nouveau projet TILED	24
Création d'une carte TILED	25
Création d'un jeu de tuiles	26
Gérer les collisions.....	28
Re factoring pour la génération du player.....	31
Sonorisation du joueur	33
Affichage d'un texte à l'écran	34
Affichage d'une barre de vie	35
Génération de tokens	36
Attraper les tokens	38
Faire bouger les tokens.....	39
Faire bouger des plateformes.....	40
Création d'un monstre	41
Génération de plusieurs monstres	42

Livret 1.4 Jeu VIDEO 2D PYTHON

Bienvenue dans le monde magique de la programmation en Python !

Salut les aventuriers du code ! Vous êtes sur le point de partir pour une incroyable aventure où vous allez découvrir comment créer votre propre jeu vidéo en 2D. Imaginez-vous en train de concevoir des personnages, des décors et des défis qui rendront votre jeu unique et captivant. Avec Python et une bibliothèque spéciale, vous allez apprendre à donner vie à vos idées les plus créatives.

Pourquoi Python ?

Python est un langage de programmation super cool et facile à apprendre. Il est utilisé par des millions de personnes dans le monde entier pour créer des applications, des sites web, des jeux et même des programmes d'intelligence artificielle. Avec Python, vous pouvez écrire du code qui est clair et simple à comprendre, ce qui est parfait pour les débutants comme vous !

Qu'est-ce qu'une bibliothèque ?

Une bibliothèque en programmation, c'est comme une boîte à outils magique. Elle contient des fonctions et des objets tout prêts que vous pouvez utiliser pour rendre votre code plus puissant et plus facile à écrire. Pour notre jeu 2D, nous allons utiliser une bibliothèque appelée **Pygame**. Pygame est spécialement conçue pour créer des jeux, et elle vous permet de gérer les graphismes, les sons et les interactions de manière simple et amusante.

La programmation orientée objet

Vous allez également découvrir un concept très important en programmation : la programmation orientée objet. Imaginez que chaque élément de votre jeu (comme un personnage, un ennemi ou un objet) est un "objet" avec ses propres caractéristiques et comportements. En utilisant la programmation orientée objet, vous pouvez organiser votre code de manière logique et réutiliser facilement des morceaux de code pour créer des éléments similaires.

Ce que vous allez apprendre

Dans cette documentation, vous allez apprendre à :

- Installer Python et Pygame sur votre ordinateur.
- Créer des fenêtres de jeu et dessiner des formes.
- Animer des objets et gérer les interactions avec le clavier et la souris.
- Utiliser des classes et des objets pour organiser votre code.
- Ajouter des sons et des effets spéciaux pour rendre votre jeu encore plus amusant.

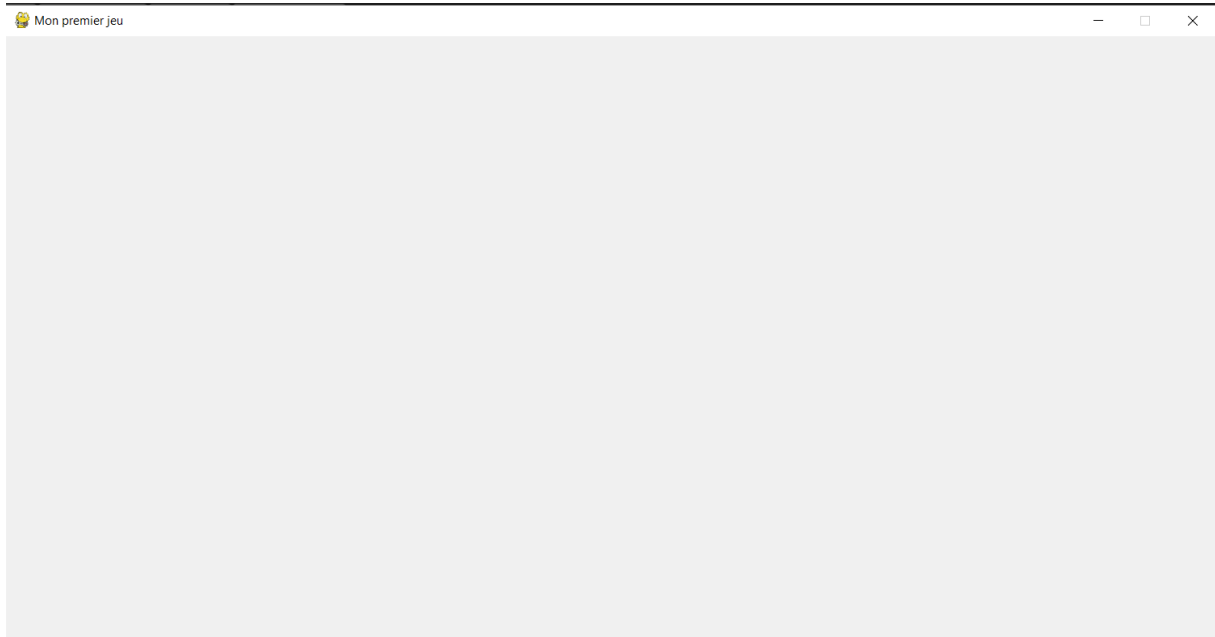
Prêts à devenir des créateurs de jeux vidéo ? Alors, allons-y ! Suivez les étapes de cette documentation et laissez libre cours à votre imagination. Qui sait, peut-être que votre jeu deviendra le prochain grand succès !

Pour concevoir ton jeu répond à ce [questionnaire](#)

Livret 1.4 Jeu VIDEO 2D PYTHON

Livret 1.4 Jeu VIDEO 2D PYTHON

Mon premier écran



```
from classes.Game2D import Game2D
from classes.common import Color
```

```
class MyGame(Game2D):
    def __init__(
        self,
        caption,                    # Titre du jeu
        game2D_width=6016,         # Espace du jeu en largeur en pixel
        game2D_height=640,         # Espace du jeu en hauteur en pixel
        screen_width=1200,         # Largeur de l'écran en pixel
        screen_height=600,         # Hauteur de l'écran en pixel
        fill_color=Color.LIGHTGRAY, # Couleur du fond de l'écran
        fps=20                      # Nombre d'images par seconde
    ):
        super().__init__(
            caption,
            game2D_width,
            game2D_height,
            screen_width,
            screen_height,
            fill_color,
```

Livret 1.4 Jeu VIDEO 2D PYTHON

```
        fps,
    )

    """
    Organiser tous les déplacements des objets de votre jeu
    """
    def myUpdate(self):
        pass

    """
    Dessiner tous les objets de votre jeu
    """
    def myDisplay(self, camera):
        pass

    """
    Initialisation du jeu
    Initialiser tous les objets dont vous aurez besoin pour votre jeu
    """
    def myInitialization(self):
        pass

    """
    Lancement du jeu
    - titre du jeu
    - largeur de l'espace du jeu en pixel
    - hauteur de l'espace du jeu en pixel
    - largeur de la fenêtre de jeu en pixel
    - hauteur de la fenêtre de jeu en pixel
    - couleur de fond de la fenêtre de jeu
    - FPS nombres d'images par seconde
    """
    myGame = MyGame("Mon premier jeu")
    myGame.run()

# Amuse-toi à modifier l'écran
myGame = MyGame("Mon premier jeu",
                6016, 640,
                1200, 640,
                (220, 220, 240))
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Dessiner un sol



```
from classes.common import Color, Shape
```

```
"""
```

```
Dessiner tous les objets de votre jeu
```

```
"""
```

```
# en arrière
```

```
def myDisplayBehind(self, camera, screen):
```

```
    pass
```

```
# devant
```

```
def myDisplayInFront(self, camera, screen):
```

```
    sol.draw(camera)
```

```
"""
```

```
Initialisation du jeu
```

```
Initialiser tous les objets dont vous aurez besoin pour votre jeu
```

```
"""
```

```
def myInitialization(self):
```

```
    global sol
```

```
    sol = Shape(0, Game2D.SIZE[1]-30,
```

```
               Game2D.SIZE[0], 50,
```

```
               Color.GREEN)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Tu vas représenter le sol par une surface rectangulaire de la largeur de l'espace de jeu et d'une hauteur de 50 pixels. Il commencera à la colonne 0 et débutera à 30 pixels au-dessus de la hauteur de l'espace de jeu. Tu fixes une couleur pour le sol.

Pour cela, tu vas appeler l'objet **Shape** de la manière suivante dans la fonction `myInitialization` :

```
sol = Shape(0, Game2D.SIZE[1] - 30, Game2D.SIZE[0], 50,  
Color.GREEN)
```

- origine colonne en pixel
- origine ligne en pixel
- taille largeur en pixel
- taille hauteur en pixel
- couleur du sol

N'oublie pas de dessiner le sol à l'écran dans la fonction `myDisplayInFront` :

```
sol.draw(camera)
```

- objet **camera** venant de la fonction `myDisplayInFront`

Livret 1.4 Jeu VIDEO 2D PYTHON

Créer un joueur



```
from classes.Game2D import Game2D, Player
```

```
def myDisplayInFront(self, camera, screen):  
    sol.draw(camera)  
    # dessiner le joueur à l'écran  
    player.draw(camera)
```

```
def myInitialization(self):  
    global sol  
    global player
```

```
    sol = Shape(0, Game2D.SIZE[1]-30,  
                Game2D.SIZE[0], 50,  
                Color.GREEN)
```

```
# Création du joueur  
player = Player(10, 10, 10, 20, Color.RED)  
# Indiquer au moteur le joueur  
self.setPlayer(player)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Tu vas représenter le joueur de ton jeu par un rectangle en définissant son origine, sa taille et sa couleur. Le joueur répondra aux déplacements effectués par le clavier.

Pour cela, tu vas appeler l'objet **Player** de la manière suivante dans la fonction myInitialization :

```
player = Player( 10,          # origine colonne  
                 10,          # origine ligne  
                 10,          # largeur en pixel  
                 20,          # hauteur en pixel  
                 Color.RED)   # couleur du joueur
```

N'oublie pas de dessiner le joueur à l'écran dans la fonction myDisplayInFront :

player.draw(camera)

- objet **camera** venant de la fonction myDisplayInFront

Livret 1.4 Jeu VIDEO 2D PYTHON

Gérer la gravité sur le joueur



```
def myInitialization(self):
    global sol
    global player

    sol = Shape(0, Game2D.SIZE[1]-30,
                Game2D.SIZE[0], 50,
                Color.GREEN)

    player = Player(10, 10, 10, 20, Color.RED)
    self.setPlayer(player)

    # Gestion de l'action de la gravité sur le joueur avec le sol
    self.gravity.addFloor(sol)
    self.gravity.addObject(player)
```

Gérer la vitesse de déplacement et la force du saut

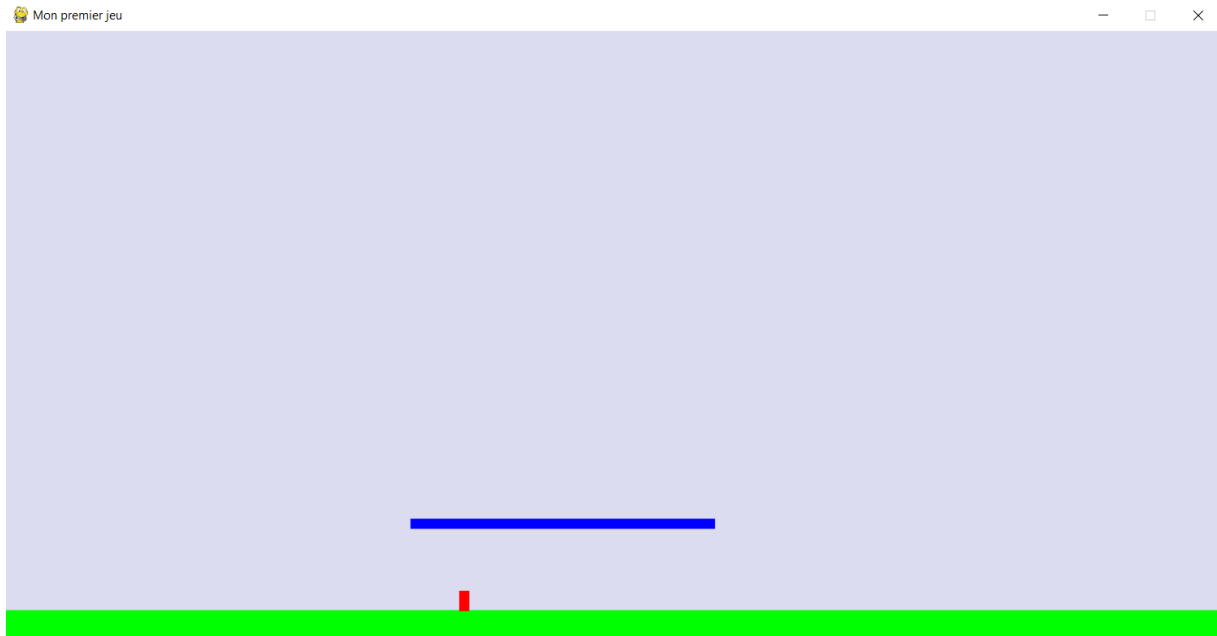
```
from classes.Game2D import Game2D, Player, Event
```

```
Event.JUMP = 150
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Event.VELOCITY = 3

Créer votre première plateforme



Pour créer votre première plateforme, initialiser un objet **Shape** dans la fonction **myInitialization** :

```
global platform01
```

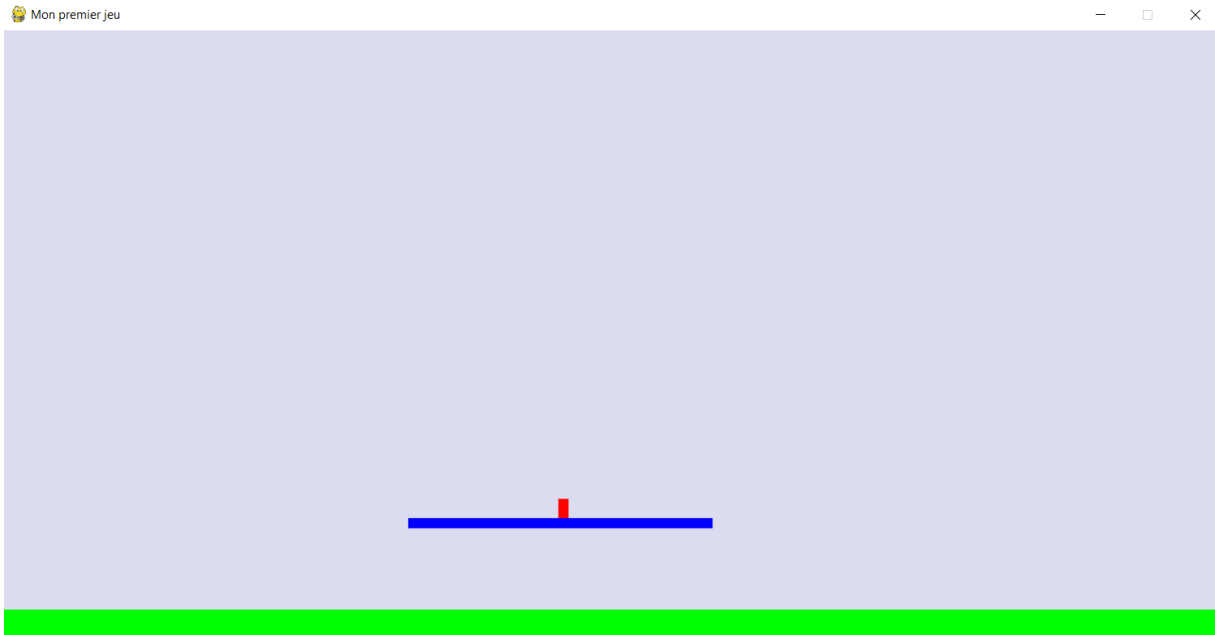
```
platform01 = Shape(400, 480, 300, 10, Color.BLUE)
```

Puis n'oubliez pas de la dessiner dans la fonction **myDisplayInFront** :

```
platform01.draw(camera)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Comment monter sur la plateforme

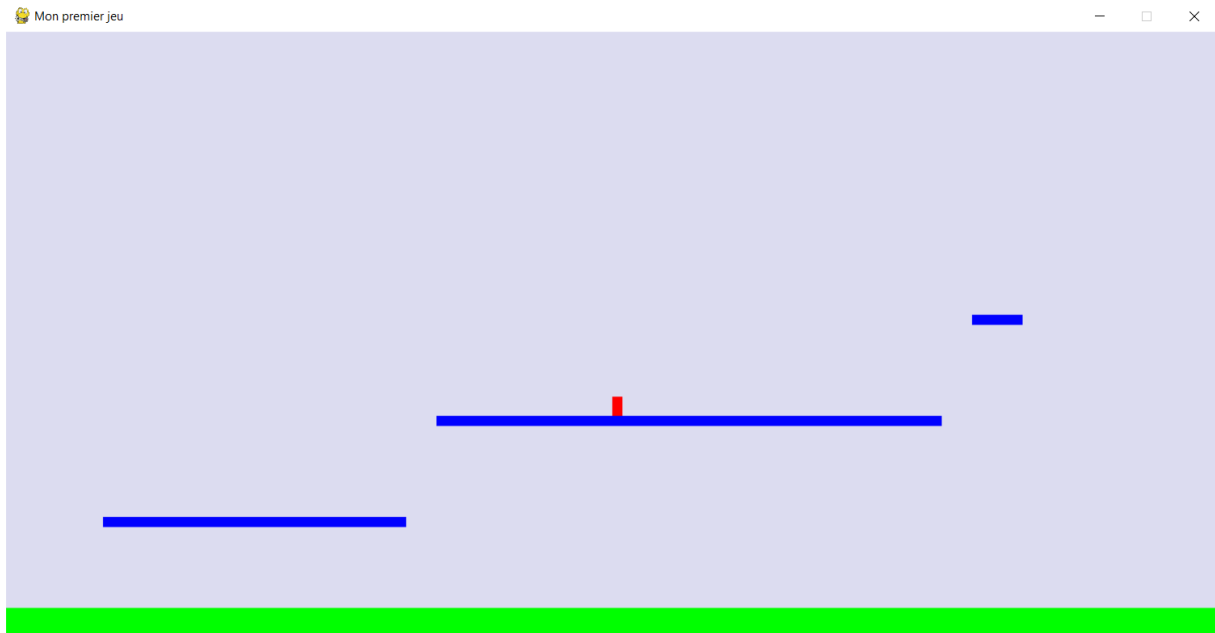


Rajoutez la plateforme dans la liste des objets de type sol :

```
self.gravity.addFloor(sol)
self.gravity.addFloor(platform01)
self.gravity.addObject(player)
self.setPlayer(player)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Créer plusieurs plateformes



Pour créer vos plateformes dans la fonction **myInitialization** :

```
global platform01
global platform02
global platform03
...
platform01 = Shape(400, 480, 300, 10, Color.BLUE)
platform02 = Shape(730, 380, 500, 10, Color.BLUE)
platform03 = Shape(1260, 280, 50, 10, Color.BLUE)
...
self.gravity.addFloor(platform01)
self.gravity.addFloor(platform02)
self.gravity.addFloor(platform03)
```

Puis n'oubliez pas de les dessiner dans la fonction **myDisplayInFront** :

```
platform01.draw(camera)
platform02.draw(camera)
platform03.draw(camera)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Modifier les conditions des déplacements

```
from classes.Game2D import Game2D, Player, Event
```

```
...
```

```
self.setPlayer(player)
```

```
Event.JUMP = 20
```

```
Event.VELOCITY = 1
```

```
Event.VELOCITY_UP = 3
```

```
Event.VELOCITY_MAX = 20
```

```
...
```

```
myGame = MyGame("Mon premier jeu",  
                6000, 600,  
                1200, 600,  
                (220, 220, 240),  
                60)
```

ou

```
myGame = MyGame("Mon premier jeu",  
                6000, 600,  
                1200, 600,  
                (220, 220, 240),  
                2)
```

Valeurs par défaut :

```
Event.JUMP = 10
```

```
Event.VELOCITY = 1
```

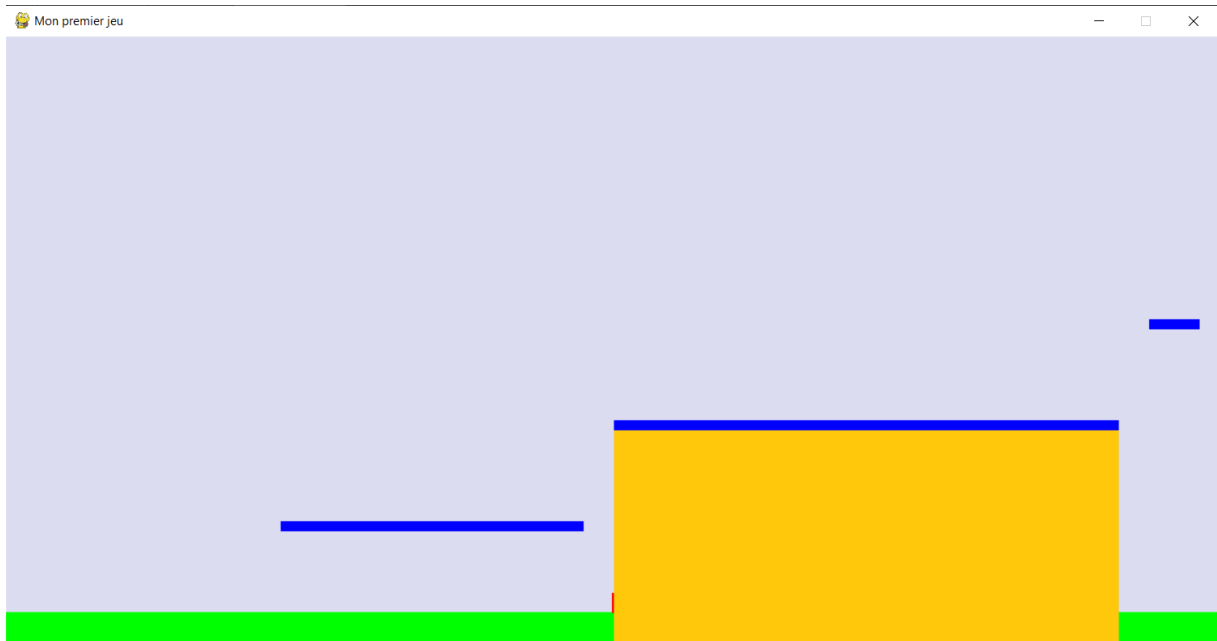
```
Event.VELOCITY_MAX = 20
```

```
Event.VELOCITY_UP = 1.1
```

Valeur par défaut du nombre d'images par seconde : **fps=20**

Livret 1.4 Jeu VIDEO 2D PYTHON

Création d'obstacles



Pour créer vos obstacles dans la fonction **myInitialization** :

global wall01

...

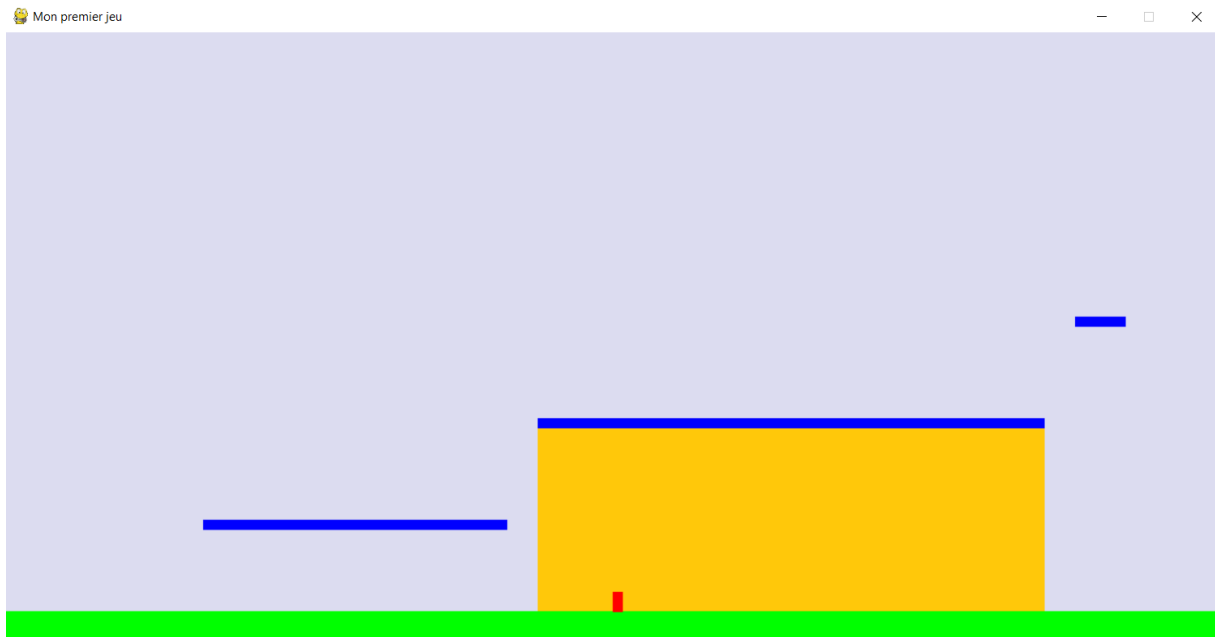
```
wall01 = Shape(730, 390,  
               500, 700,  
               (255,200,10))
```

Puis n'oubliez pas de les dessiner dans la fonction **myDisplayInFront** :

wall01.draw(camera)

Livret 1.4 Jeu VIDEO 2D PYTHON

Gérer la visibilité des objets graphiques entre eux



Dans la fonction **myDisplayInFront** vous pouvez gérer la priorité d’affichage des différents objets :

```
def myDisplayInFront(self, camera, screen):  
    # Dessin des obstacles  
    wall01.draw(camera)  
    # Dessin du sol  
    sol.draw(camera)  
    # Dessin des plateformes  
    platform01.draw(camera)  
    platform02.draw(camera)  
    platform03.draw(camera)  
    # Dessin du joueur  
    player.draw(camera)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Gestion des collisions avec les obstacles

Dans la fonction **myInitialization**, vous pouvez initialiser la gestion la collision entre les différents objets de votre jeu :

```
# Gestion des collisions
```

```
# Déclaration de tous les murs ou objets contre lesquels on peut  
rentrer en collision
```

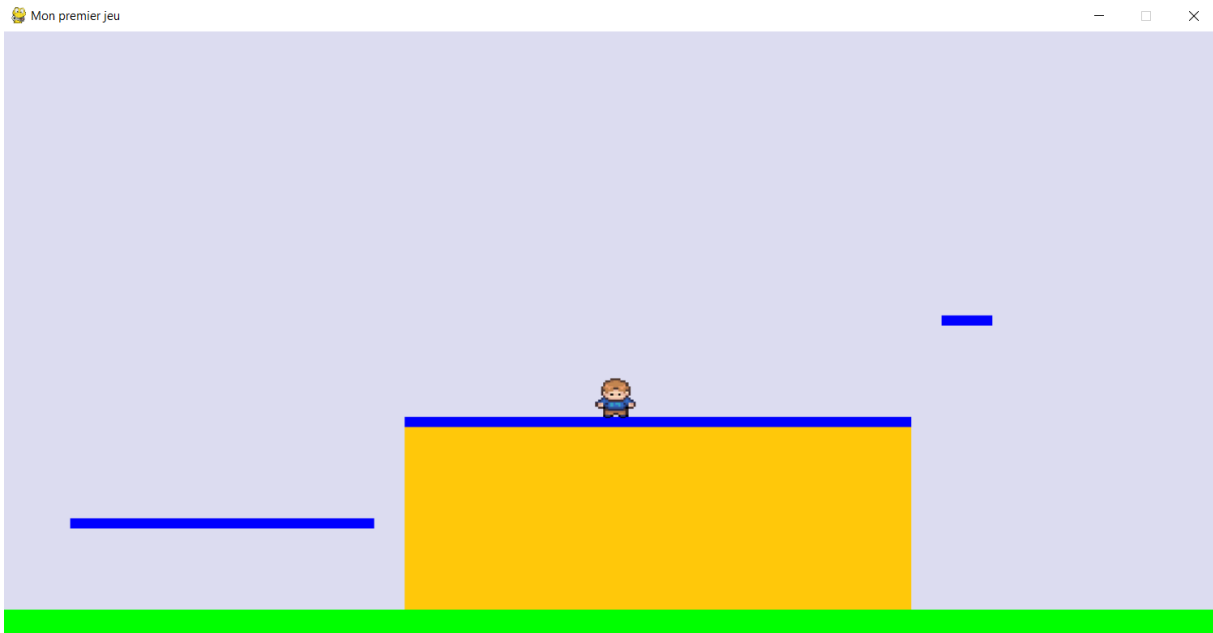
```
self.collisionsManager.addCollider(wall01)
```

```
# Déclaration des objets en mouvement qui peuvent rentrer en  
collision
```

```
self.collisionsManager.addMovingObject(player)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Animer le joueur



from classes.Game2D import Game2D, Player, Event, **Animation**

Dans la fonction **myInitialization**, déclare l'animation du joueur en décrivant les images pour chaque mouvement du joueur dans un tableau :

```
ANIMATIONS_PLAYER = {"DOWN" : ["devant01.png",  
                                "devant02.png",  
                                "devant03.png"],  
                      "UP" : ["dos01.png",  
                              "dos02.png",  
                              "dos03.png"],  
                      "LEFT" : ["cotegauche01.png",  
                                "cotegauche02.png",  
                                "cotegauche03.png"],  
                      "RIGHT" : ["cotedroit01.png",  
                                "cotedroit02.png",  
                                "cotedroit03.png"],  
                      "IDLE" : ["attendredevant.png"],  
                      "IDLEDOWN" : ["attendredevant.png"],  
                      "IDLEUP" : ["attendredos.png"],  
                      "IDLELEFT" : ["attendregauche.png"],  
                      "IDLERIGHT" : ["attendredroit.png"]}]
```

Livret 1.4 Jeu VIDEO 2D PYTHON

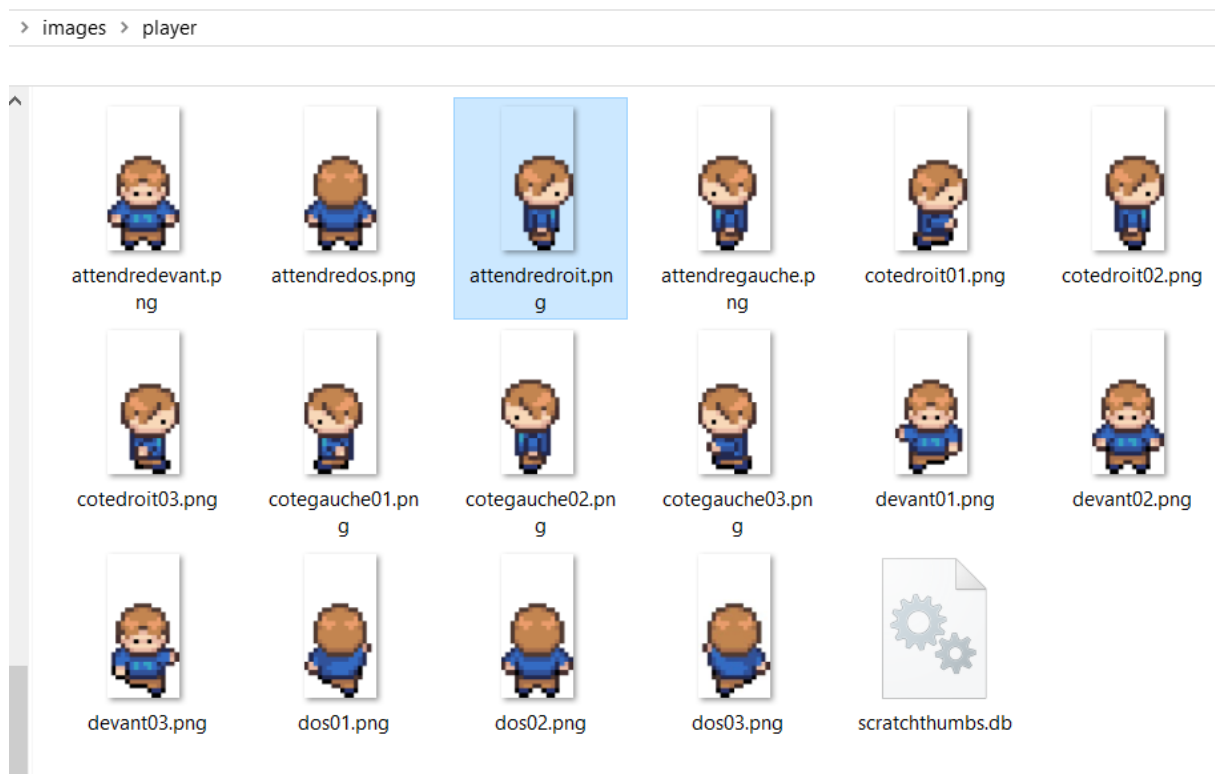
Puis génère une animation avec le tableau des images :

```
animationPlayer = Animation(ANIMATIONS_PLAYER,  
"images/player", delay=5)
```

- tableau dictionnaire : mouvement, image
- chemin pour trouver les images sur le disque
- délai entre chaque image

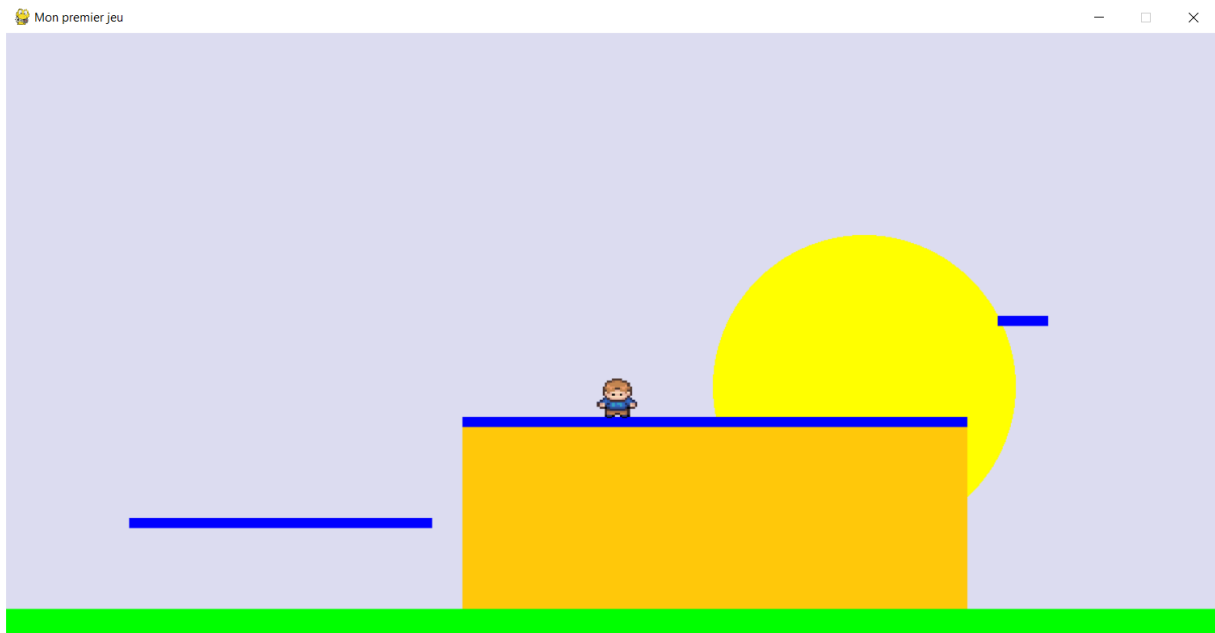
```
player.loadAnimations(animationPlayer, "IDLE", 15, 40)
```

- animation du personnage
- position du personnage au début de l'animation
- taille de l'animation en pixel



Livret 1.4 Jeu VIDEO 2D PYTHON

Création d'un soleil, objet fixe, illusion déplacement



```
from classes.common import Color, Shape, Circle
```

Pour créer ton soleil tu vas générer un cercle jaune dans la fonction **myInitialization** :

```
global sun  
...  
sun = Circle(700, 200, 150, (255,255,0))
```

Puis pour dessiner en arrière-plan et de manière fixe dans la fonction **myDisplayBehind**:

Ainsi le soleil est dessiné derrière les décors et les personnages

```
...  
# en arrière  
def myDisplayBehind(self, camera, screen):  
    sun.draw(screen)  
...
```

- **screen** permet un affichage fixe sur l'écran quelque soit les déplacement du joueur.

Livret 1.4 Jeu VIDEO 2D PYTHON

Le logiciel TILED

Tiled est un éditeur de carte 2D très utile pour la création de jeux vidéo.

Voici 10 raisons qui en font un outil précieux :

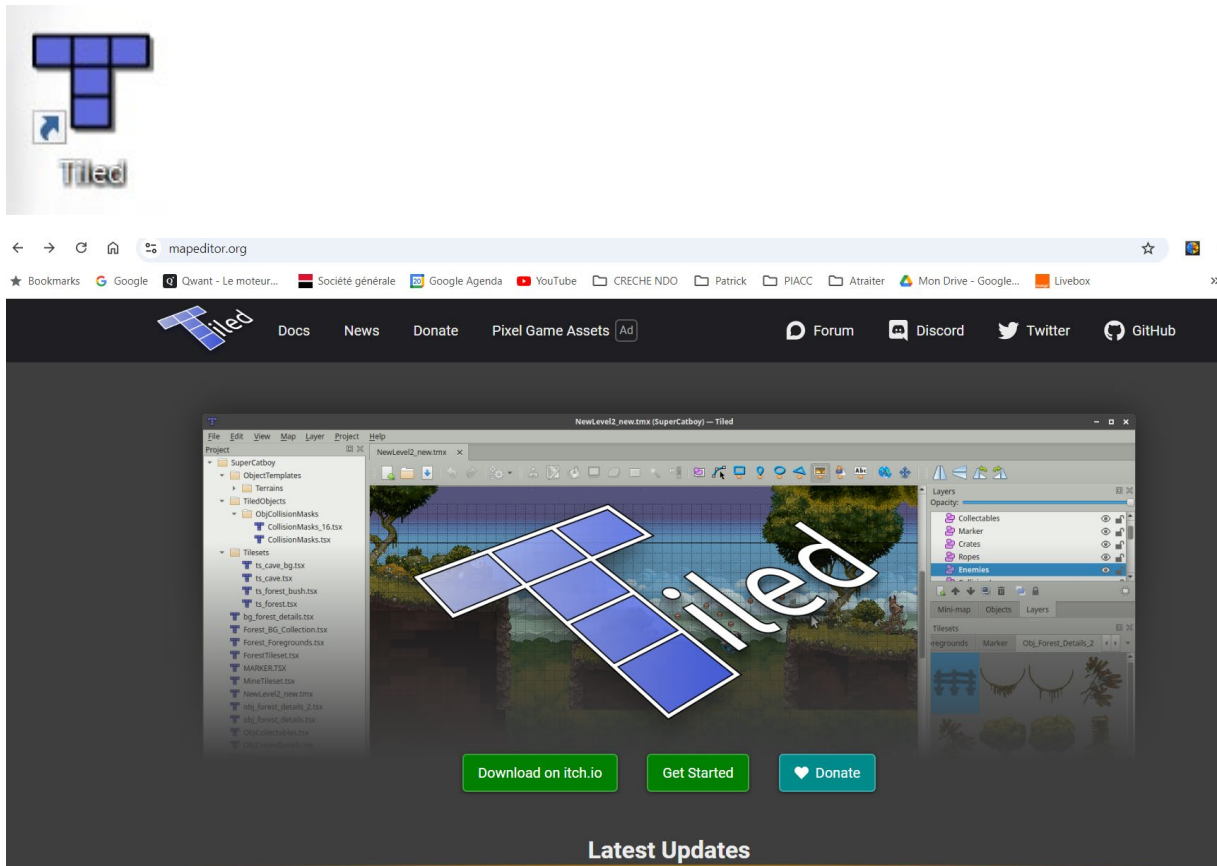
1. Cartes modulaires : Tiled permet de concevoir des cartes de jeu modulaires en divisant le terrain de jeu en tuiles, ce qui facilite la création d'environnements complexes.
2. Facilité d'utilisation : Son interface conviviale rend la création de niveaux et de cartes plus accessible, même pour les débutants.
3. Économie de temps : En utilisant des tuiles pré-conçues, Tiled accélère le processus de conception des niveaux, permettant ainsi de gagner du temps de développement.
4. Gestion des calques : Il offre la possibilité de superposer des calques pour organiser les éléments du jeu, permettant ainsi une meilleure gestion et manipulation des différents éléments graphiques.
5. Compatibilité avec Pygame : Tiled est compatible avec Pygame et d'autres moteurs de jeu couramment utilisés pour les jeux 2D, permettant une intégration aisée des cartes dans le jeu.
6. Édition collaborative : Il facilite le travail en équipe grâce à la possibilité de partager et collaborer sur les cartes, ce qui est idéal pour les projets de développement de jeux en équipe.
7. Support de multiples formats : Tiled prend en charge plusieurs formats de fichier, comme TMX, JSON, XML, permettant une intégration flexible avec divers moteurs de jeu.
8. Création de mondes complexes : Grâce à sa capacité à gérer des cartes de grande taille, Tiled permet de créer des mondes de jeu vastes et détaillés.
9. Paramétrage des propriétés : Il offre la possibilité de définir des propriétés personnalisées pour les tuiles et les objets, offrant ainsi une grande flexibilité dans la conception du jeu.
10. Communauté active : Tiled bénéficie d'une communauté active, offrant des tutoriels, des ressources et un support continu pour les développeurs de jeux 2D.

Livret 1.4 Jeu VIDEO 2D PYTHON

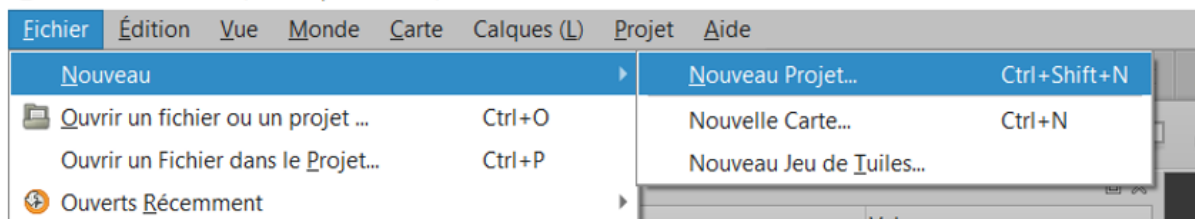
En somme, Tiled est un outil inestimable pour la création de jeux vidéo 2D grâce à sa simplicité d'utilisation, son efficacité dans la conception de cartes et son intégration harmonieuse avec les moteurs de jeu populaires.

Livret 1.4 Jeu VIDEO 2D PYTHON

Création d'un nouveau projet TILED




Jeu2D Level01.tmx (Jeu 2D plateForme) - Tiled

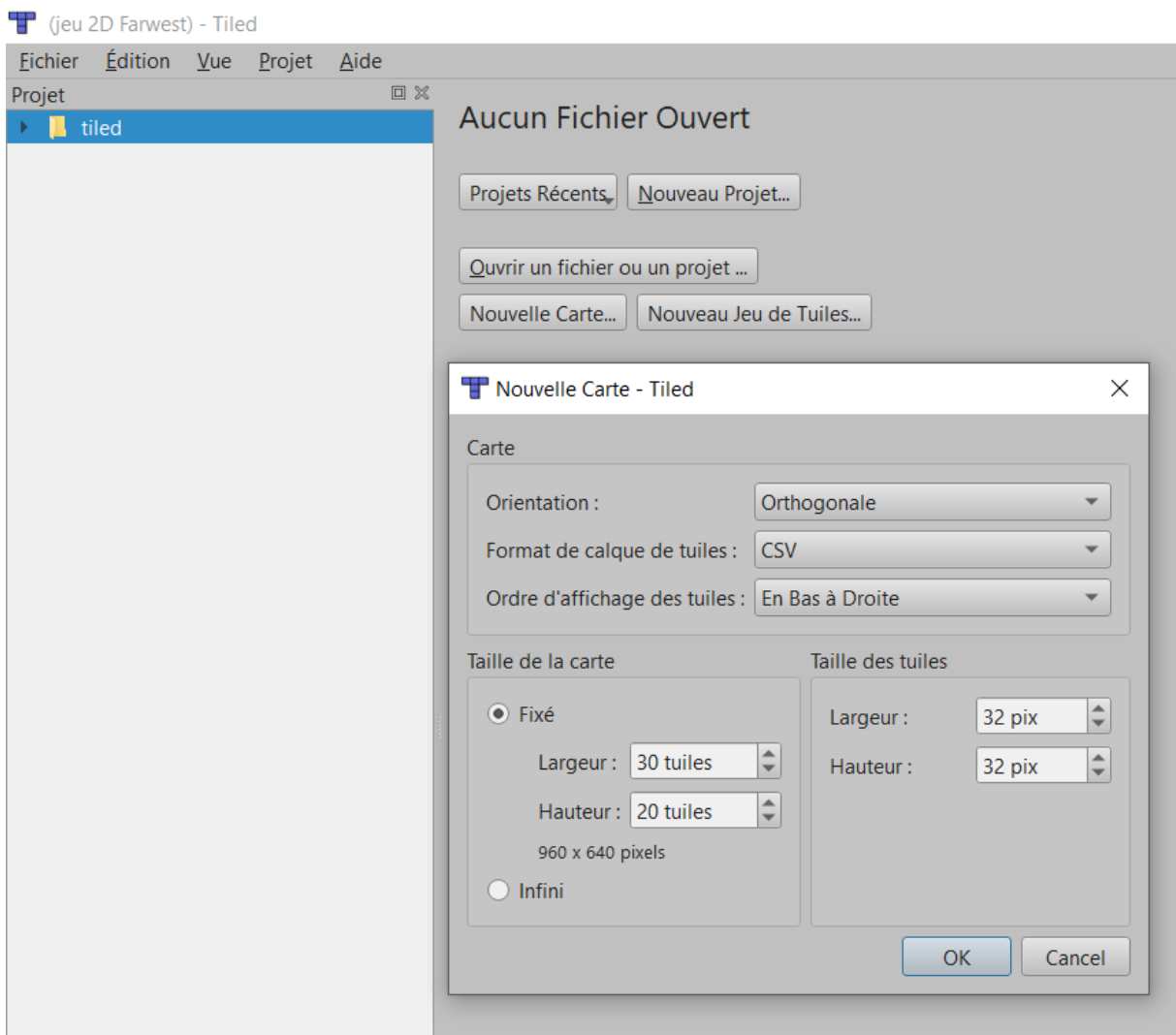
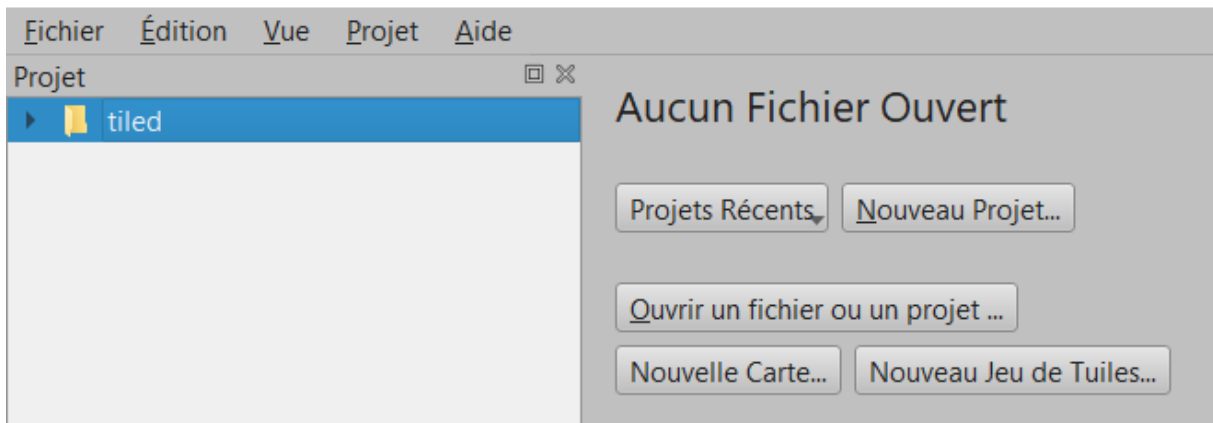


Donne un nouveau nom au projet et choisis son emplacement.

Livret 1.4 Jeu VIDEO 2D PYTHON

Création d'une carte TILED

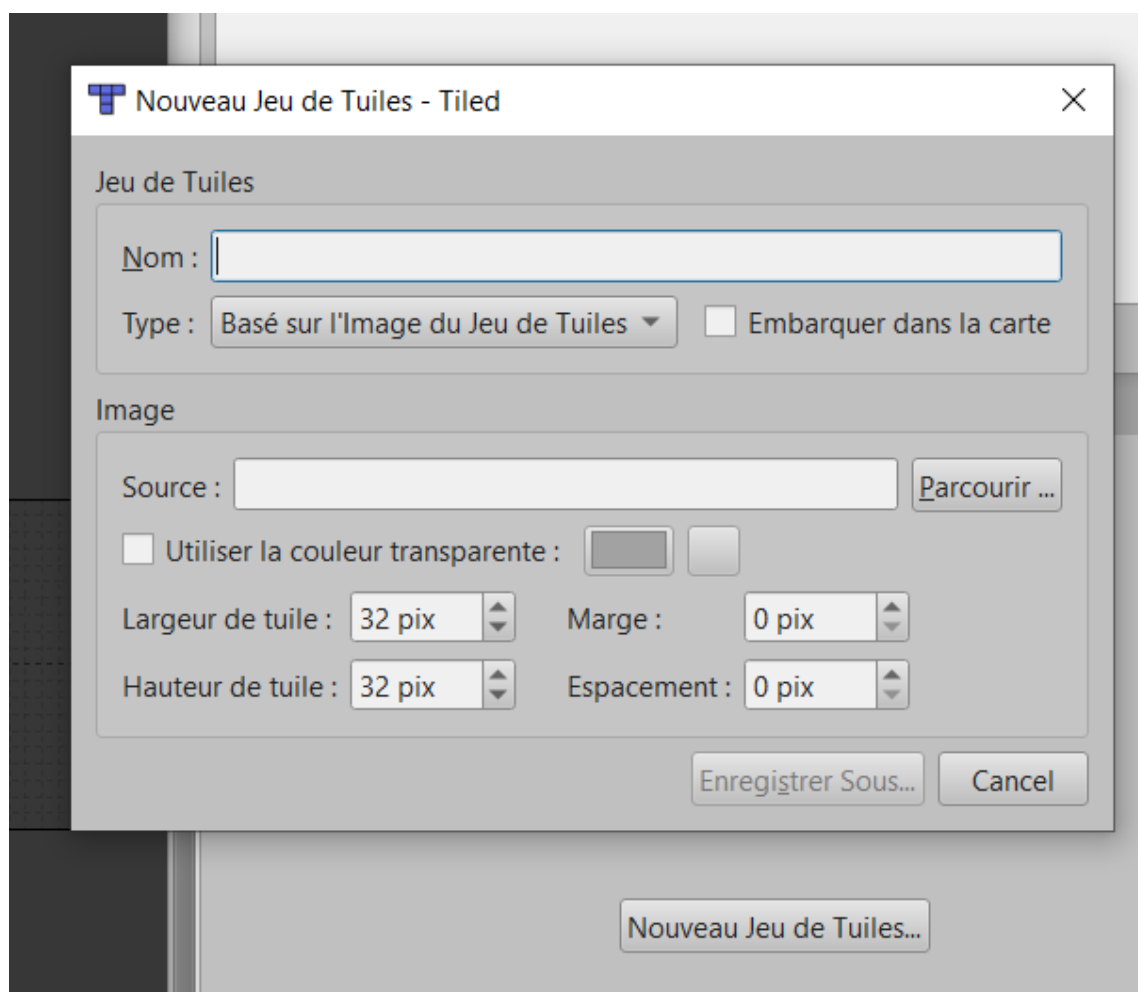
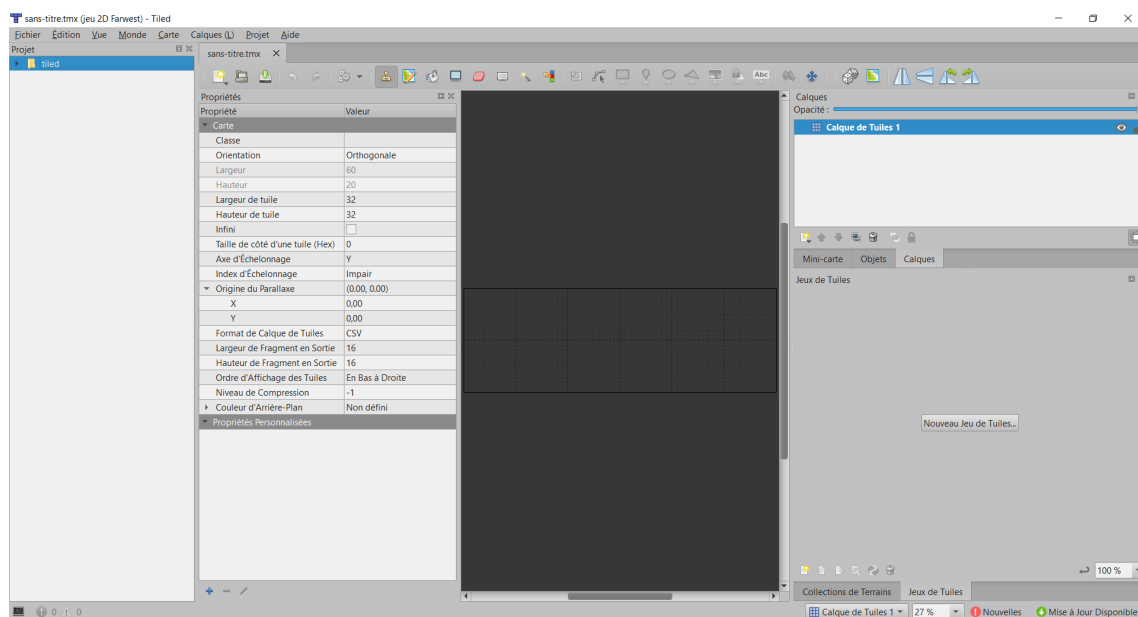
 (jeu 2D Farwest) - Tiled



Choisis la taille des tuiles en pixels (par exemple 32 par 32) puis la taille de la carte en nombre de tuiles.

Livret 1.4 Jeu VIDEO 2D PYTHON

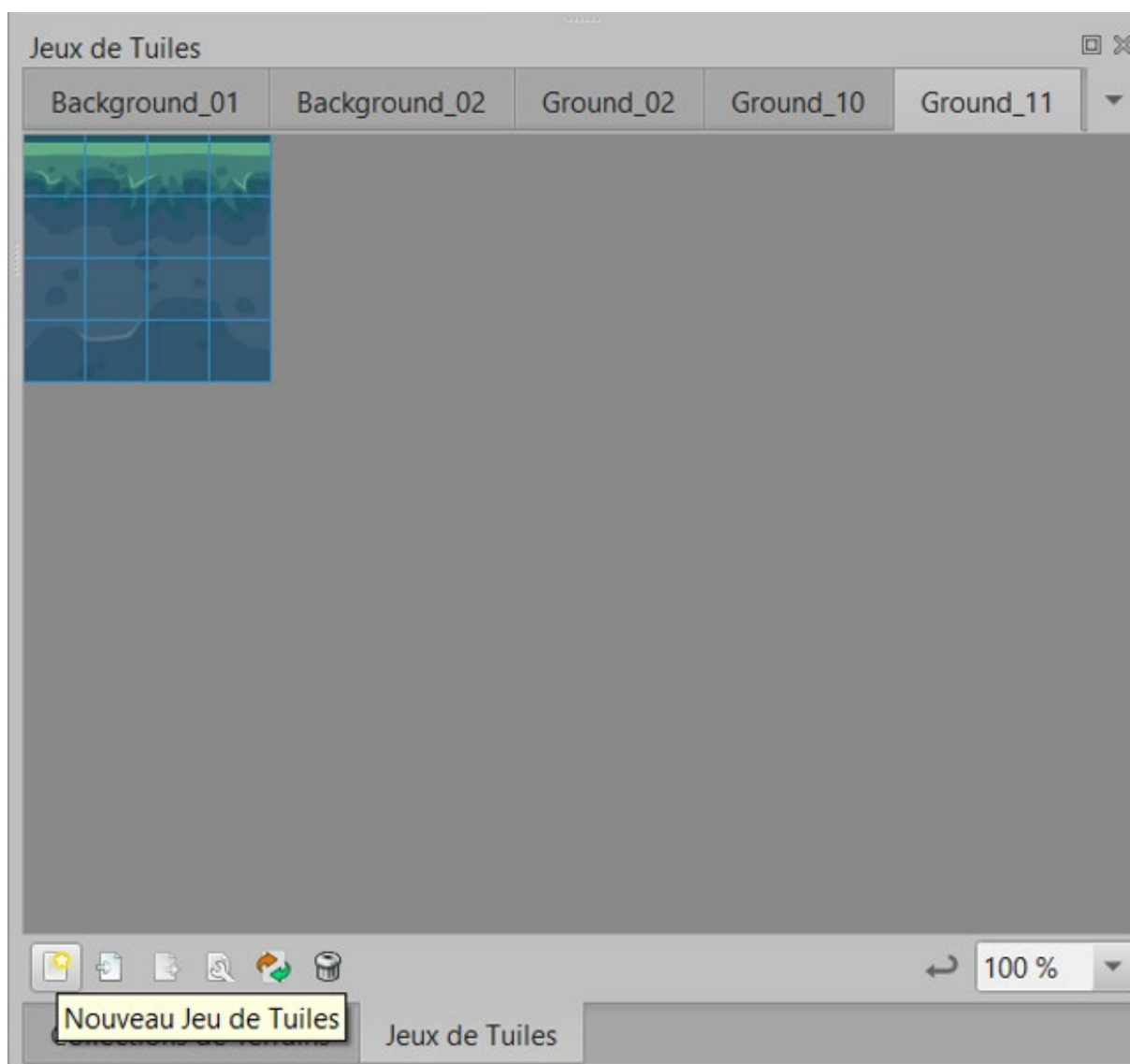
Création d'un jeu de tuiles



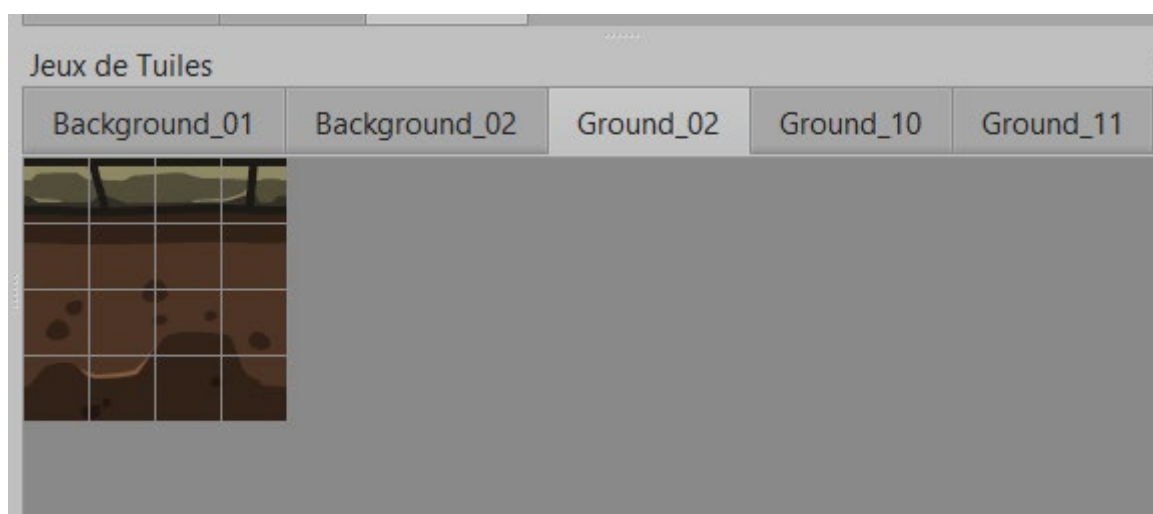
Choisis l'emplacement de l'image de ton jeu de Tuiles.

Livret 1.4 Jeu VIDEO 2D PYTHON

Pour créer un nouveau jeu de tuiles :

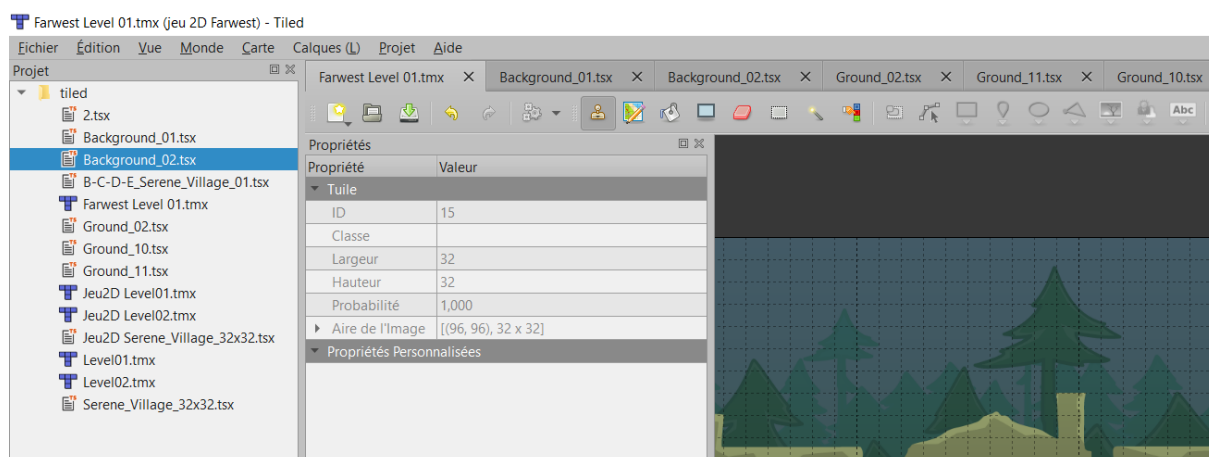


Choisis les onglets pour passer d'un jeu de tuiles à un autre



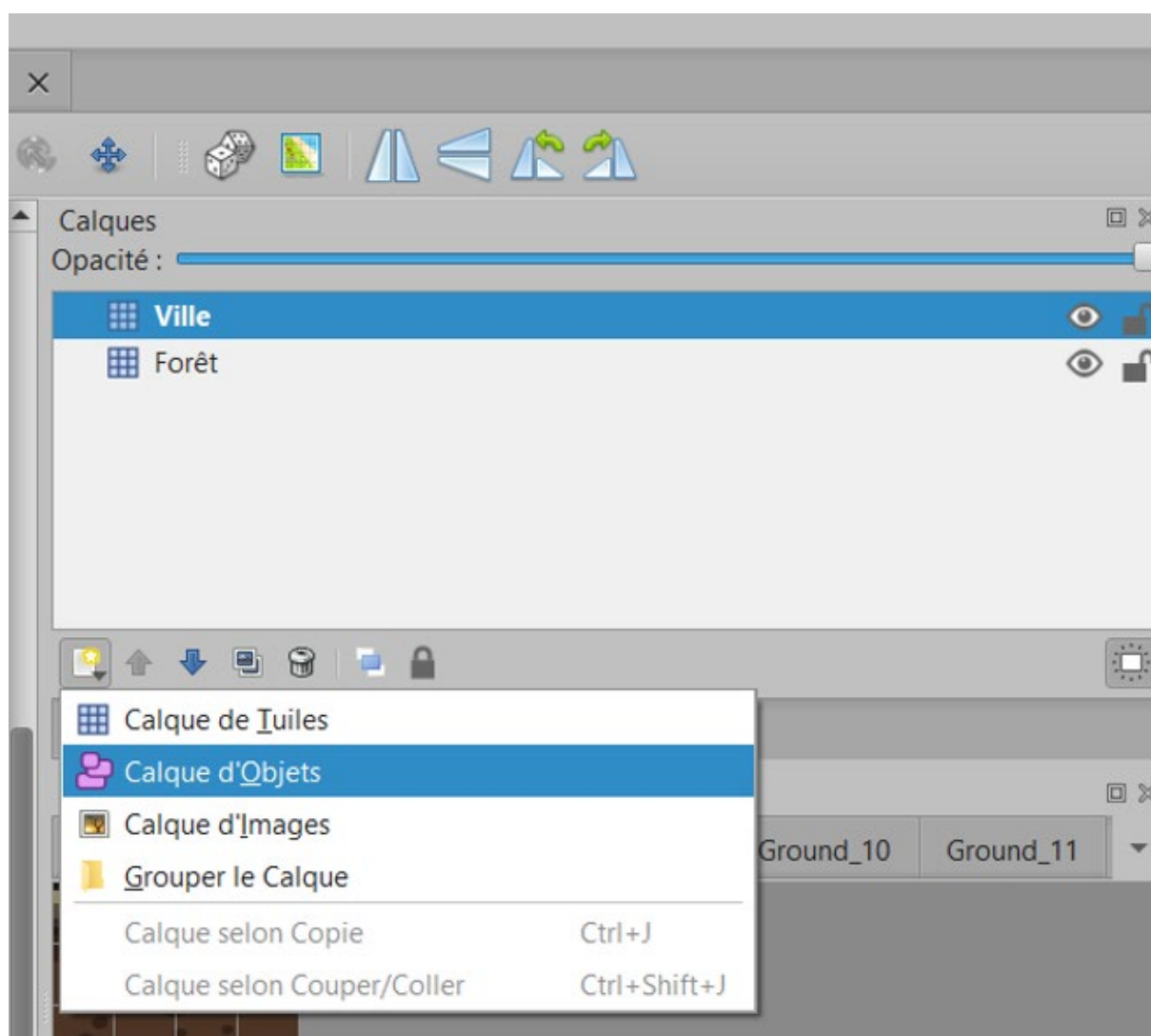
Livret 1.4 Jeu VIDEO 2D PYTHON

Choisis le premier onglet tmx pour dessiner tes niveaux

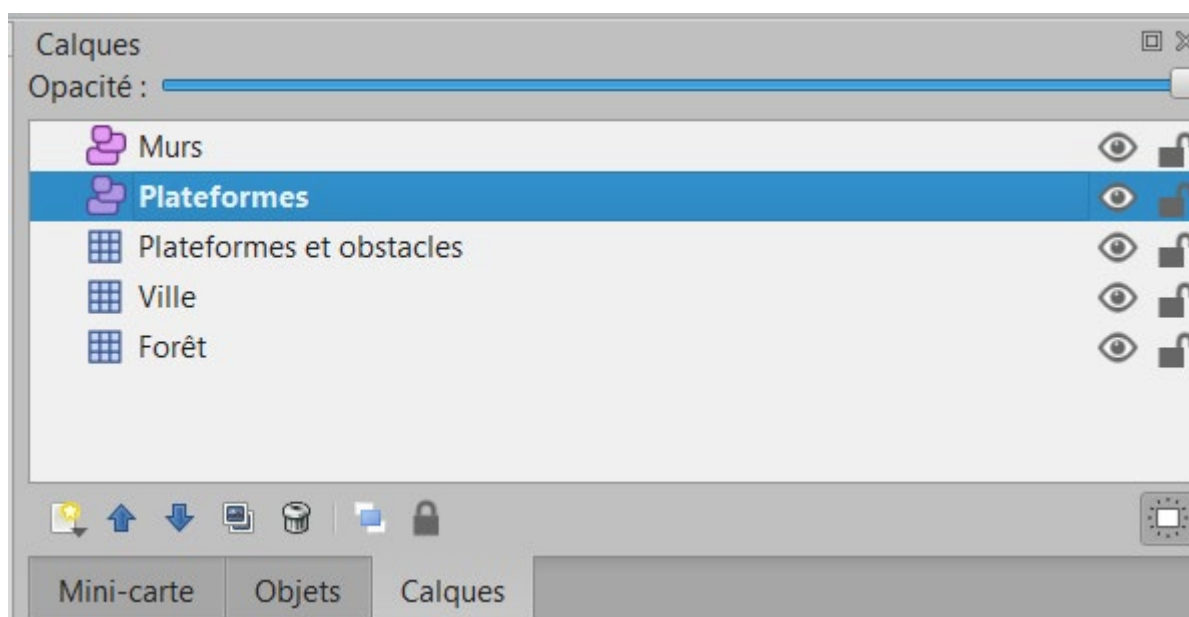


Gérer les collisions

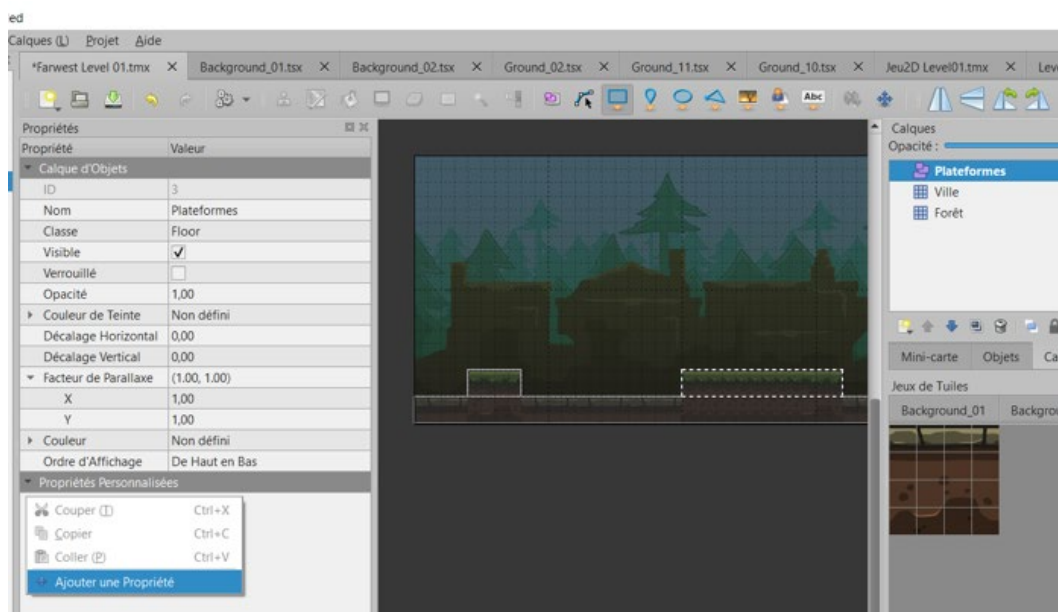
Choisis calque d'objets pour définir les collisions avec les plateformes



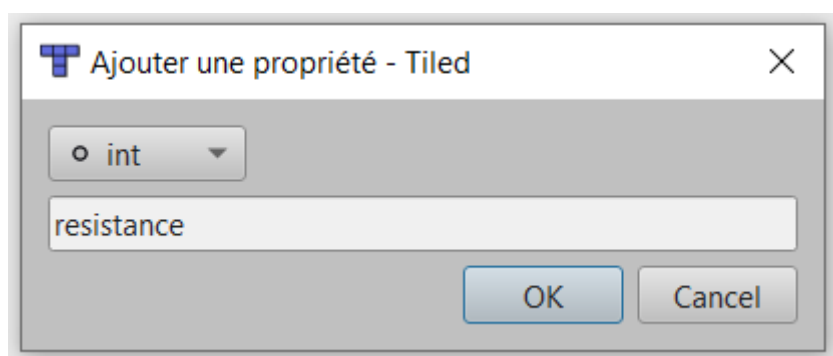
Livret 1.4 Jeu VIDEO 2D PYTHON



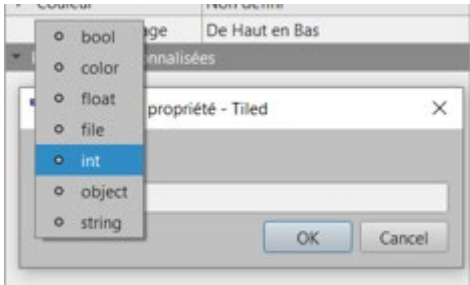
Dans « Classe » indique : Floor



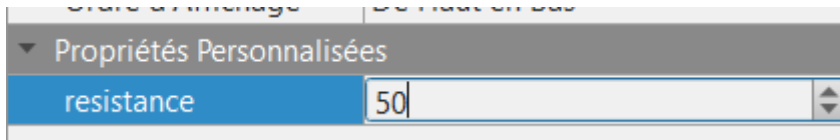
Puis crée une nouvelle propriété pour définir la résistance du sol :



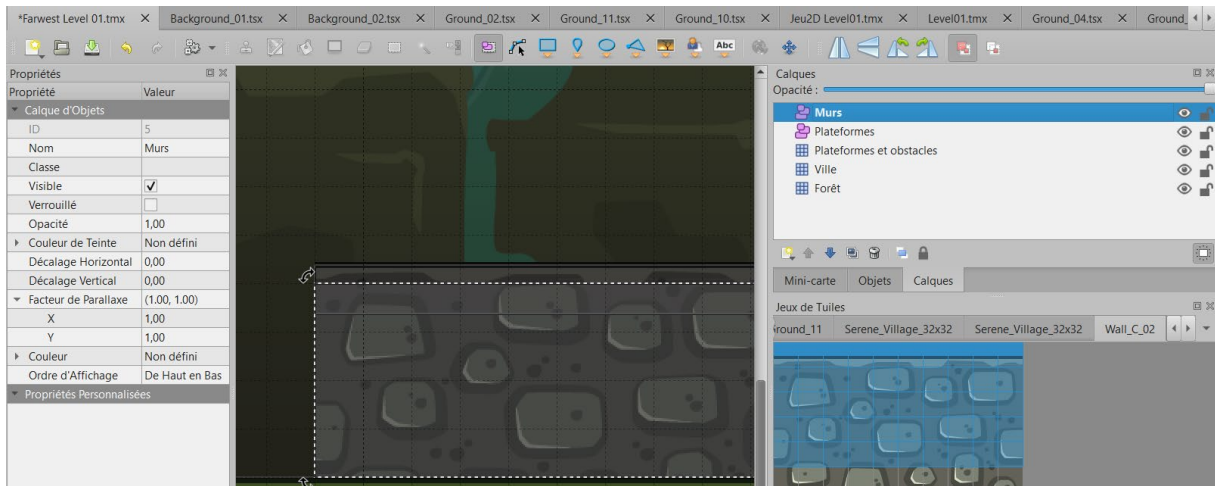
Livret 1.4 Jeu VIDEO 2D PYTHON



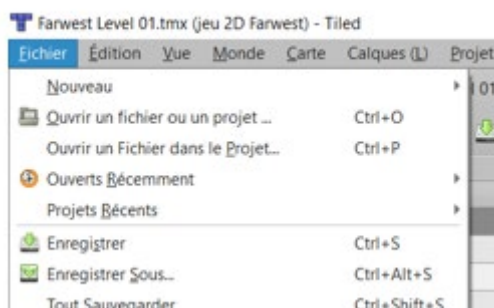
Avec la valeur 50 :



Crée un objet pour gérer la collision contre un mur



Puis enregistre le projet :



Rajoute dans ton projet dans la fonction « myInitialization » :

L'appel au chargement de ton projet TILED

self.loadTiled("Farwest Level 01.tmx", "tiled\\")

Livret 1.4 Jeu VIDEO 2D PYTHON

Re factoring pour la génération du player

En programmation, l'opération d'optimiser son code et le rendre plus lisible est souvent appelée **refactoring**. Le refactoring consiste à modifier la structure interne du code sans en changer le comportement externe. L'objectif est d'améliorer la lisibilité, la maintenabilité, la performance et la compréhension du code.

Le refactoring peut inclure des actions telles que :

- Renommer des variables et des fonctions pour qu'elles soient plus descriptives.
- Réorganiser le code en fonctions ou modules plus petits et plus spécialisés.
- Supprimer le code mort ou redondant.
- Simplifier les structures de contrôle.
- Améliorer la documentation et les commentaires.

Afin de rendre ton code plus simple à comprendre, tu vas regrouper toutes les instructions pour la création du « player » dans une seule fonction :

```
def generatePlayer(self):  
  
    # Création d'un joueur  
    player = Player(300, 200, 10, 20, Color.GREEN)  
    # Définition des animations du joueur  
    ANIMATIONS_PLAYER = {"DOWN" : ["devant01.png",  
                                     "devant02.png",  
                                     "devant03.png"],  
                          "UP" : ["dos01.png",  
                                   "dos02.png",  
                                   "dos03.png"],  
                          "LEFT" : ["cotegauche01.png",  
                                    "cotegauche02.png",  
                                    "cotegauche03.png"],  
                          "RIGHT" : ["cotedroit01.png",  
                                      "cotedroit02.png",  
                                      "cotedroit03.png"],
```

Livret 1.4 Jeu VIDEO 2D PYTHON

```
"IDLE" : ["attendredevant.png"],  
"IDLEDOWN" : ["attendredevant.png"],  
"IDLEUP" : ["attendredos.png"],  
"IDLELEFT" : ["attendregauche.png"],  
"IDLERIGHT" : ["attendredroit.png"]}]
```

```
animation = Animation(ANIMATIONS_PLAYER, "images/player",  
delay=5)
```

```
# Affectation de l'animation au joueur
```

```
player.loadAnimations(animation, "IDLE", 15, 40)
```

```
return player
```

puis dans le programme principale tu remplaces toutes les instructions pour la création du « player » par :

```
# Création d'un joueur
```

```
player = self.generatePlayer()
```


Livret 1.4 Jeu VIDEO 2D PYTHON

Sonorisation du joueur

Modifie la fonction generatePlayer pour rajouter des éléments sonores aux animations du joueur notamment lors des déplacements et du saut :

```
def generatePlayer(self):
    # Création d'un joueur
    player = Player(300, 200, 10, 20, Color.GREEN)
    # Définition des animations du joueur
    ANIMATIONS_PLAYER = {"DOWN" : ["devant01.png",
                                     "devant02.png",
                                     "devant03.png"],
                          "UP" : ["dos01.png",
                                   "dos02.png",
                                   "dos03.png"],
                          "LEFT" : ["cote gauche01.png",
                                    "cote gauche02.png",
                                    "cote gauche03.png"],
                          "RIGHT" : ["cote droite01.png",
                                     "cote droite02.png",
                                     "cote droite03.png"],
                          "IDLE" : ["attendre devant.png"],
                          "IDLE DOWN" : ["attendre devant.png"],
                          "IDLE UP" : ["attendre dos.png"],
                          "IDLE LEFT" : ["attendre gauche.png"],
                          "IDLE RIGHT" : ["attendre droite.png"]}

    SOUNDS_PLAYER = { "UP" : ["jump_01.wav"],
                      "LEFT" : ["SF-course_beton-2.mp3"],
                      "RIGHT" : ["SF-course_beton-2.mp3"]}

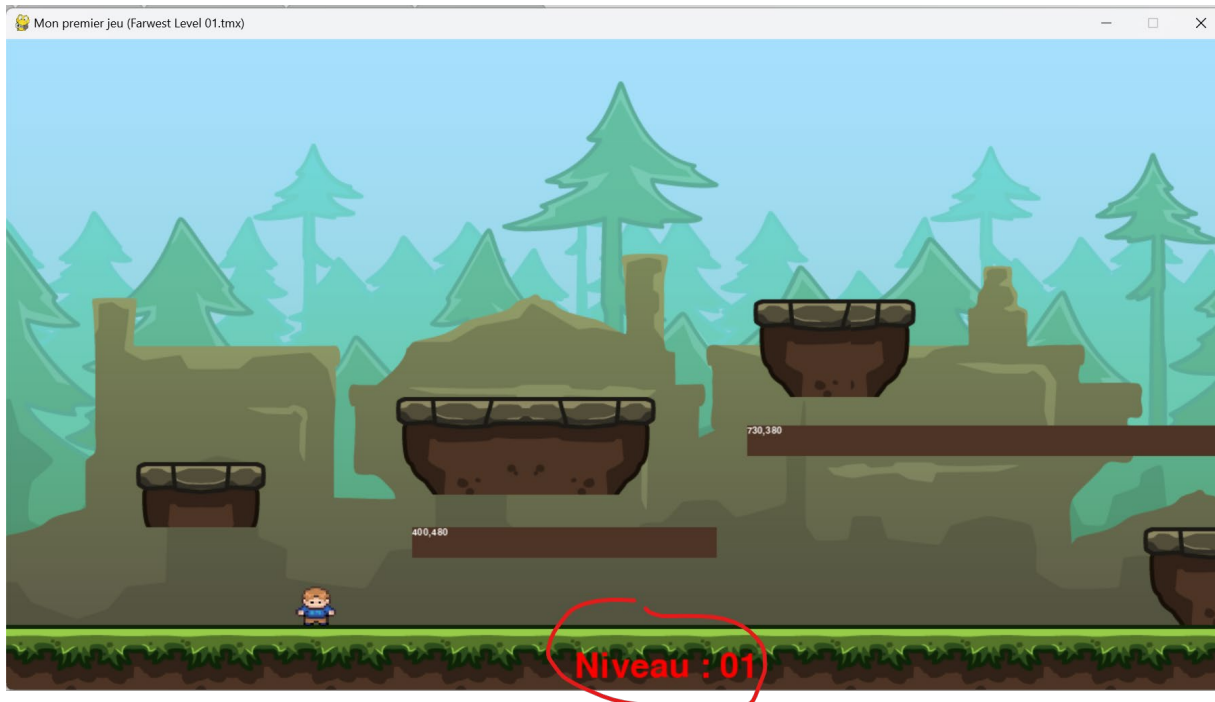
    animation = Animation(ANIMATIONS_PLAYER, "images/player",
                           delay=5, soundAnimation = SOUNDS_PLAYER, soundPath = "Sounds")

    # Affexation de l'animation au joueur
    player.loadAnimations(animation, "IDLE", 15, 40)

    return player
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Affichage d'un texte à l'écran



Ajoute dans l'import des bibliothèques l'objet **Text**

```
from classes.common import Color, Shape, Circle, Text
```

Puis dans la fonction **myInitialization** la création de l'objet du texte :

```
def myInitialization(self):
```

```
    global numberLevel
```

```
    numberLevel = Text("Niveau : 01", (Game2D.SCREEN_SIZE[0]/2)-40 , Game2D.SCREEN_SIZE[1]-40, Color.RED)
```

Les options pour l'objet Text :

- Le texte à afficher
- Abscisse en pixel du positionnement du texte
- Ordonnée en pixel du positionnement du texte
- La couleur du texte
- La police du texte
- La taille du texte

N'oublie pas d'indiquer de dessiner le texte :

```
def myDisplayInFront(self, camera, screen):  
    numberLevel.draw(screen)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Affichage d'une barre de vie



Rajoute l'objet **LifeBar** dans l'import :

```
from classes.Game2D import Game2D, Player, Event, Animation,  
LifeBar
```

Puis demande la création de ta barre de vie dans la fonction **myInitialization** :

```
def myInitialization(self):  
    global hearts  
    # génération de la barre des vies  
    hearts = LifeBar(5, "heart.png", "images/token", 10,  
Game2D.SCREEN_SIZE[1]-40)
```

Les options pour l'objet Text :

- L'image de chaque vie
- Le chemin d'accès de l'image
- Abscisse en pixel du positionnement de la barre de vie
- Ordonnée en pixel du positionnement de la barre de vie

```
def myDisplayInFront(self, camera, screen):  
    hearts.draw(screen)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Génération de tokens



Tu vas générer des tokens que le joueur pourra attraper. Les tokens ont deux états : un en attente de se faire attraper, un autre lorsqu'ils sont touchés par le joueur. Un bruit est émis lorsque le joueur touche un jeton.

```
def generateToken(self):  
    # Animations graphiques du jeton  
    ANIMATIONS_TOKEN = {  
        "WAIT": ["token01.png",  
                 "token02.png",  
                 "token03.png",  
                 "token04.png"],  
        "DEAD": ["token_dead.png"],  
    }  
    # son lorsque le jeton est touché par le joueur  
    SOUNDS_TOKEN = { "DEAD" : ["hit_12.wav"]}   
    # création de l'animation du jeton  
    animationToken = Animation(ANIMATIONS_TOKEN,  
                                "images/token", (64, 64), delay=1, soundAnimation =  
                                SOUNDS_TOKEN, soundPath = "Sounds")
```

Livret 1.4 Jeu VIDEO 2D PYTHON

création d'un token et affectation de son animation

```
token = Circle(50, 50, 30, (255, 255, 0))  
token.loadAnimations(animationToken, "WAIT")
```

return token

Maintenant, que la déclaration de la génération des tokens est faite, tu vas générer les tokens et les afficher.

Génère les tokens dans la fonction **myInitialization** :

```
def myInitialization(self):  
    global tokens  
  
    # Génération du jeton  
    token = self.generateTokens()
```

Tu peux utiliser deux méthodes pour générer plusieurs tokens.
Méthode 1, tu declares tous les emplacements de tes tokens :

```
tokens = EntitiesManager(token,  
    ((120, 300), (400, 200), (800, 100), (1100, 210)))
```

Méthode 2, tu demandes à générer un nombre de tokens aléatoirement dans un espace défini :

```
tokens = EntitiesManager(token, 10, 100, 150,  
    Game2D.SIZE[0], Game2D.SIZE[1]-300)
```

Puis dessine les tokens dans les fonctions **myUpdate** et **myDisplayInFront** :

```
def myUpdate(self):  
    tokens.move()  
  
def myDisplayInFront(self, camera, screen):  
    tokens.draw(camera)
```

Fait en sorte de dessiner les tokens avant le player.

Livret 1.4 Jeu VIDEO 2D PYTHON

Attraper les tokens

Modifie-la génération de la barre des vies pour la paramétrer avec une variable qui comptera le nombre de vie.

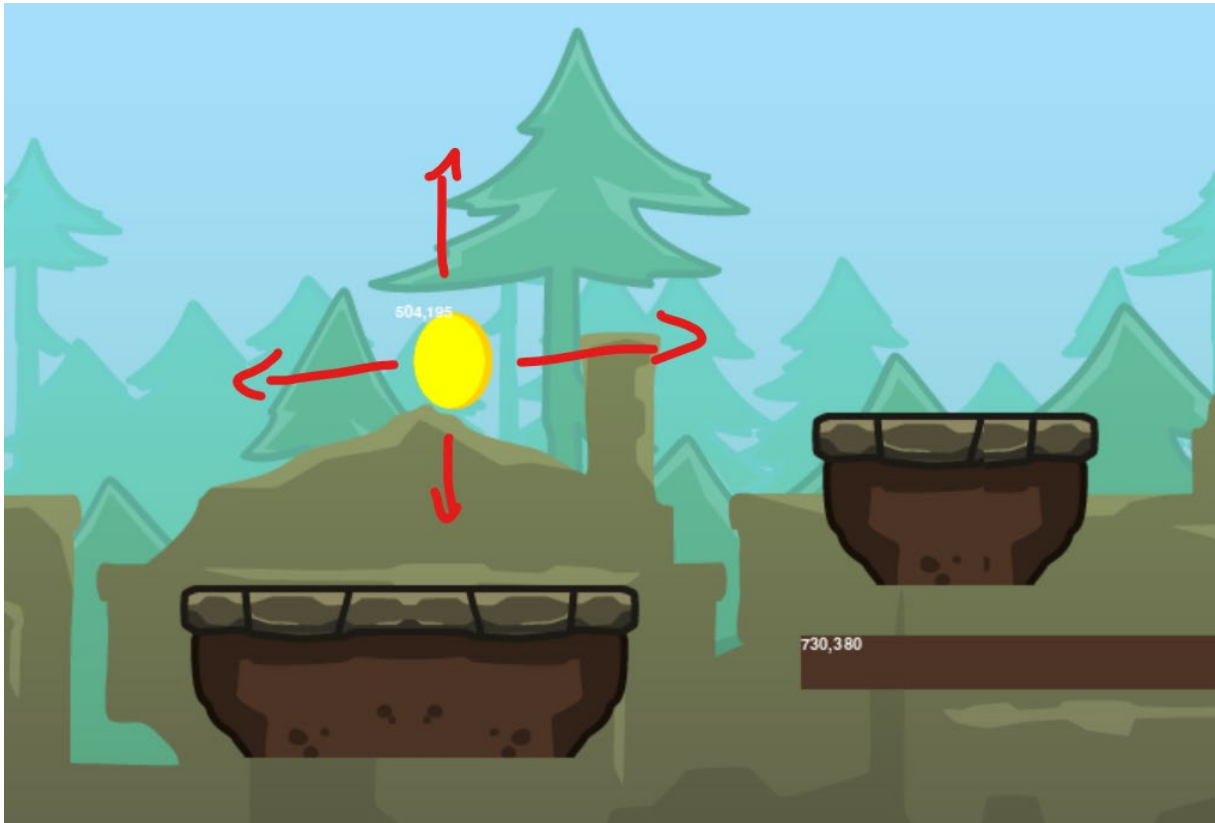
```
def myInitialization(self):  
    global nbrLives  
  
    ...  
  
    nbrLives = 0  
    # génération de la barre des vies  
    hearts = LifeBar(nbrLives, "heart.png", "images/token", 10,  
                    Game2D.SCREEN_SIZE[1]-40)
```

Puis tu vas contrôler quand le joueur touche un jeton dans la fonction myUpdate :

```
def myUpdate(self):  
    global nbrLives  
    tokens.move()  
    # si collision entre le joueur et un des tokens  
    token = tokens.isCollide(player)  
    # si un token a été touché alors animer et  
    programmer la disparition du token et rajouter une vie  
    if token :  
        self.destructionManager.addEntity(token, 1, "DEAD")  
        tokens.remove_entity(token)  
        nbrLives+=1  
        hearts.refresh(nbrLives)
```

Livret 1.4 Jeu VIDEO 2D PYTHON

Faire bouger les tokens



Rajoute dans la fonction **myInitialization**

```
def myInitialization(self):
```

une destination à tes tokens avec la méthode suivante :

tokens.goToTarget([0,100],10) pour un déplacement de 100 pixels sur l'axe y soit de haut en bas et de bas en haut

tokens.goToTarget([100,0],10) pour un déplacement de 100 pixels sur l'axe x soit de droite vers la gauche puis de la droite vers la gauche

Le deuxième paramètre donne la vitesse du déplacement en nombre de pixels à chaque affichage.

Tu peux également proposer un déplacement sur les deux axes :

tokens.goToTarget([100,100],10)

Livret 1.4 Jeu VIDEO 2D PYTHON

Faire bouger des plateformes



Rajoute dans la fonction **myInitialization** une méthode pour le déplacement de ta plateforme :

```
def myInitialization(self):  
    platform01.goToTarget([0,100],5)
```

N'oublie pas de rajouter dans la fonction **myUpdate** :

```
def myUpdate(self):  
  
    # Faire bouger les plateformes  
    platform01.move()
```

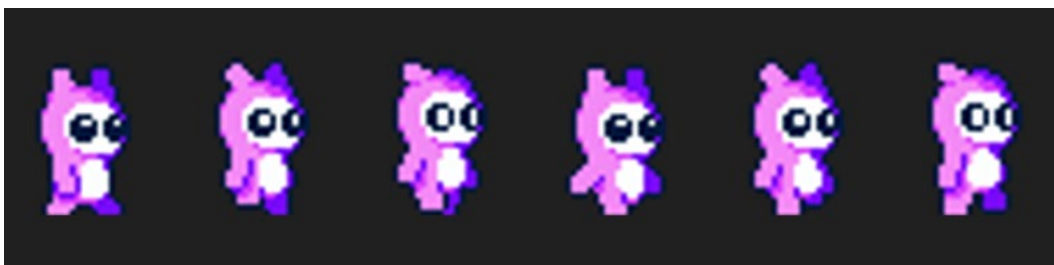

Livret 1.4 Jeu VIDEO 2D PYTHON

Création d'un monstre



Crée une fonction pour générer un monstre :

```
def generateMonster(self):  
  
    ANIMATIONS_MONSTER = {  
        "DEAD": ["Pink_Monster_Idle_4.png:4:H"],  
        "WALK": ["Pink_Monster_Walk_6.png:6:H"],  
    }  
    animationMonster = Animation(ANIMATIONS_MONSTER,  
"images/Monsters", (32, 32), delay=1)  
    monster = Shape(300, 750, 32, 32, Color.RED)  
    monster.loadAnimations(animationMonster, "WALK", 0, 0)  
    return monster
```



"Pink_Monster_Walk_6.png:6:H"

L'image regroupe toutes les images d'animation de marche du monstre. Il faut déclarer le nombre d'image et la façon dont les découper H horizontalement ou V verticalement.

Livret 1.4 Jeu VIDEO 2D PYTHON

Génération de plusieurs monstres



Rajoute dans la fonction **myInitialization** la génération aléatoire des monstres puis demander que les monstres se dirigent vers le joueur :

```
def myInitialization(self):  
    global monsters  
  
    monster = self.generateMonster()  
    monsters = EntitiesManager(monster, 10, 100, 150,  
                                Game2D.SIZE[0], Game2D.SIZE[1]-300)  
  
    monsters.goToTarget(player)
```