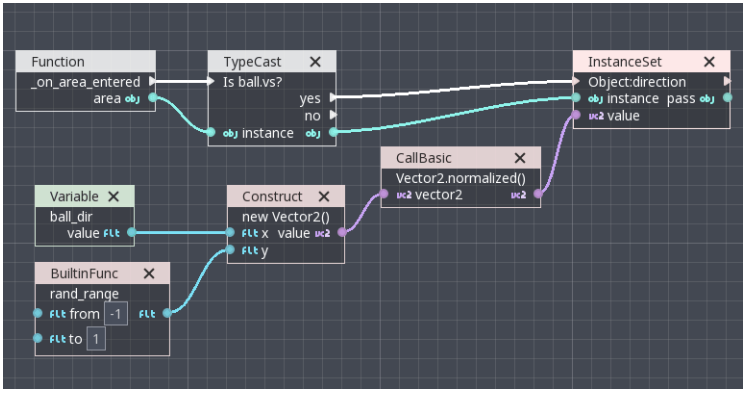
4

Initiation à la création de scripts avec Godot

Comme tous les moteurs de jeux, Godot permet la création de scripts. Les scripts permettent de mettre en place des interactions entre le joueur et le jeu. Nous pourrons récupérer les *inputs* (comme les entrées clavier/souris), gérer le comportement des objets (comme le déplacement du personnage ou des éléments à collecter) ou encore mettre en place les conditions de réussite/échec du jeu.

La création de scripts passe par de la programmation. Le langage de programmation historique de Godot est le GDScript, une sorte de Python. Ce langage est assez simple à prendre en main pour les débutants tout en offrant une grande puissance. À partir de la version 3 de Godot, d'autres langages de programmation ont été introduits, notamment le C# et un langage en *visual scripting* permettant de coder de façon visuelle par assemblage de blocs.

Figure 4.1 : *Le visual scripting sous Godot*



Depuis la version 4 de Godot, une nouvelle version de C#, plus performante, est proposée. C# a l'avantage d'être, dans la majorité des cas, plus puissant que GD Script.

En outre, c'est un langage très utilisé dans le monde du jeu vidéo (Unity, MonoGame, Stride, CryEngine, etc.). Il présente donc beaucoup d'atouts, et son emploi avec Godot progresse rapidement.

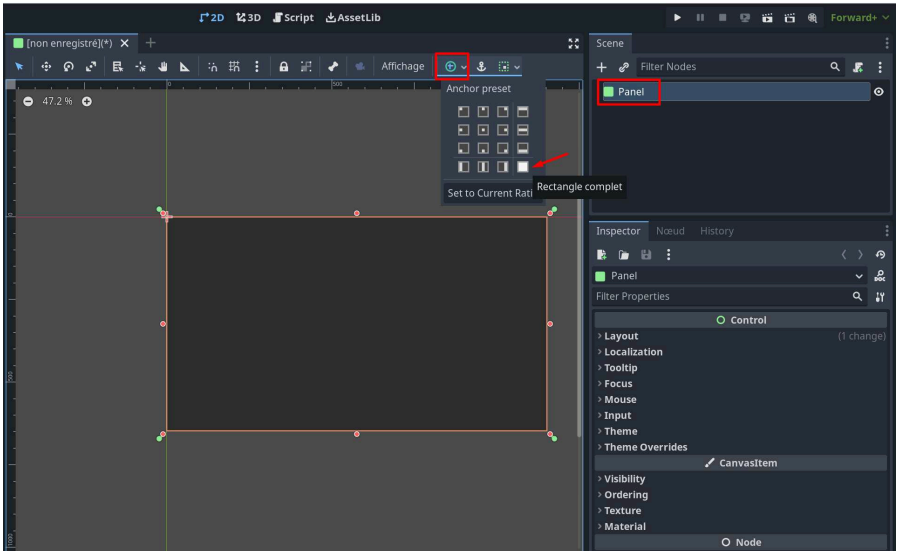
Bien que GDScript soit encore très populaire, nous n'utiliserons dans ce livre que le langage C#. Il s'agit du langage que je vous recommande car il va prendre beaucoup d'importance dans les années à venir. Il est aujourd'hui beaucoup plus documenté que par le passé et permet de faire autant que GDScript.

Dans ce chapitre, nous allons découvrir comment écrire nos scripts en C#.

4.1. Premier script

Commencez par créer une nouvelle scène vide afin de partir sur une base simple. Créez ensuite un nœud Panel (Ctrl+A) que vous agrandirez pour couvrir la taille de l'écran de jeu. Pour aller vite, vous pouvez utiliser les ancres prédéfinies.

Figure 4.2 : *Création d'un Panel*



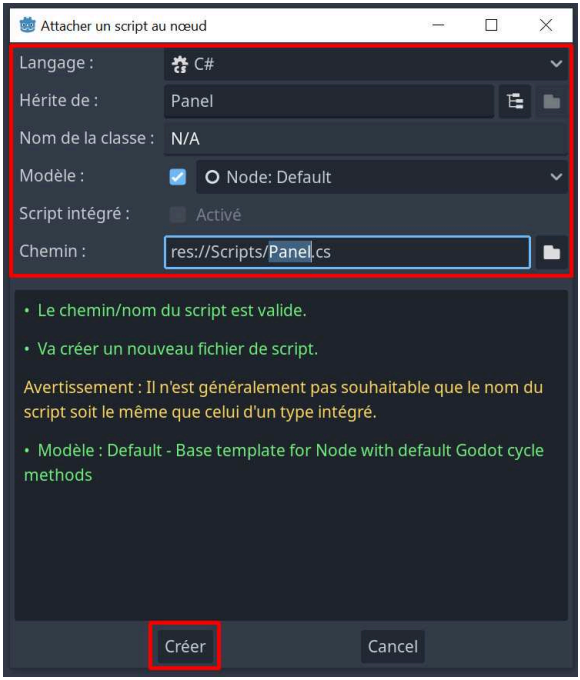
Note *› Le Panel est le composant qui est en général utilisé pour contenir des éléments d'interface comme du texte, des boutons et des champs textes. Un menu principal ou un menu pause sera par exemple conçu dans un panel.*

Nous avons créé un Panel car nous avons besoin d'un nœud principal pour pouvoir créer un script. Nous aurions pu créer n'importe quel autre type de nœud. Nous avons choisi un Panel car nous allons créer plus tard une UI (User Interface).

Lorsque vous avez un nœud dans votre scène, vous avez la possibilité de lui ajouter un script. Pour cela, avec le nœud sélectionné, cliquez sur l'icône Script se trouvant en haut à droite du panneau Scène.

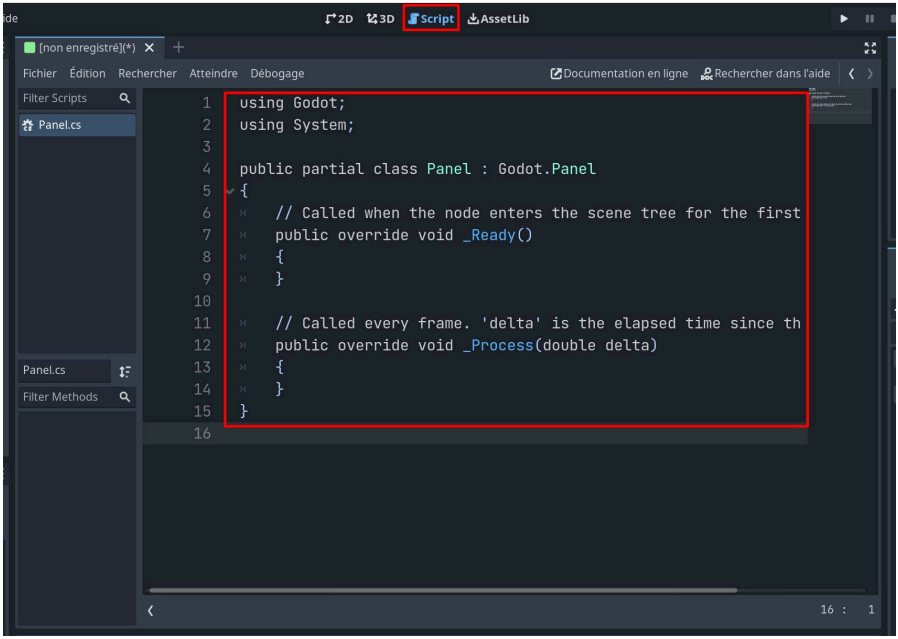
Lors de la création d'un script, vous devez sélectionner le langage ; nous allons choisir C#. Vous devez également préciser le modèle, gardez celui par défaut. Enfin, il vous faut spécifier un chemin : cliquez sur l'icône en forme de dossier afin de choisir le chemin d'enregistrement (je vous recommande de sauvegarder votre script dans un sous-dossier nommé `Scripts`). Une fois que tout est paramétré, cliquez sur CRÉER.

Figure 4.3 : *Configuration du script*



Lorsque vous validez, le script se crée et s'ouvre comme sur la Figure 4.4.

Figure 4.4 : *Script par défaut*



L'espace de travail SCRIPT s'est automatiquement ouvert. Quelques lignes de code sont déjà écrites par défaut. Voici quelques explications pour comprendre :

`using` est un mot clé qui permet d'inclure des fonctionnalités à notre script. Ici, nous incluons les fonctionnalités de Godot et du système.

`class` est le mot clé permettant de créer une classe (le code de notre objet) et `:Godot.Panel` d'indiquer que cette classe hérite des propriétés de `Panel` (car notre nœud est un Panel). L'héritage permet à l'enfant (le script) d'accéder aux propriétés du parent (ici Panel). Cela nous permettra de déclencher des événements propres au Panel si besoin.

Dans le code présent par défaut, vous remarquerez la présence de deux fonctions (`_Ready` et `_Process`).

FONCTION

Une fonction est une partie du programme qui exécute des opérations bien précises. Par exemple, nous pourrions créer une fonction `Soustraire` qui prendrait en paramètres `nombre1` et `nombre2` et qui retournerait la soustraction de ces deux nombres. Cette fonction pourrait ressembler à cela :

```
public int Soustraire(int nb1, int nb2)
{
  return nb1 - nb2;
}
```

Le mot clé `public` indique que la fonction est publique et qu'elle peut être appelée de n'importe où et par n'importe qui. L'inverse serait l'utilisation du mot clé `private` qui lui indique que la fonction ne peut être appelée que par la classe elle-même.

Le mot clé `return` permet de renvoyer une valeur. Une fonction peut être appelée par son nom, dans notre cas ce serait : `Soustraire(10,5);`. Le résultat serait alors 5.

`_Ready()` est une fonction particulière car elle s'exécute au lancement du programme. C'est la première fonction qui est lancée à l'ouverture de la scène.

`_Process(double delta)` est également une fonction très importante, car elle tourne en boucle. Cela signifie que si votre jeu tourne en 60 images par seconde, elle s'exécutera 60 fois par seconde. Son paramètre par défaut `delta` correspond au temps écoulé entre deux frames (entre deux images). Il permettra par exemple de calibrer vos animations pour que leur vitesse soit la même indépendamment de la puissance de l'ordinateur qui exécute le programme. Ce sera très utile par exemple pour les animations, afin de faire avancer le personnage de façon fluide.

Enfin, vous verrez dans les différents programmes, lors de la déclaration des fonctions, les mots clés `void` ou encore `override`. `void` signifie vide. Une fonction déclarée comme étant `void` ne retourne rien. Elle s'exécute mais ne retourne aucune valeur. Quand on parle de valeur de retour, il s'agit d'une valeur traitée par une fonction et renvoyée pour être utilisée ailleurs dans le code. Par exemple, une fonction `Additionner` devrait retourner une valeur qui serait le résultat de l'addition. `override`, quant à lui, est un mot clé qui permet de réécrire le comportement d'une fonction déjà existante. Dans le cas de `_Ready` par exemple il s'agit d'une fonction présente de base avec Godot. Pour la réécrire ou la redéclarer, on utilise `override`.