Interface de la liste des projets

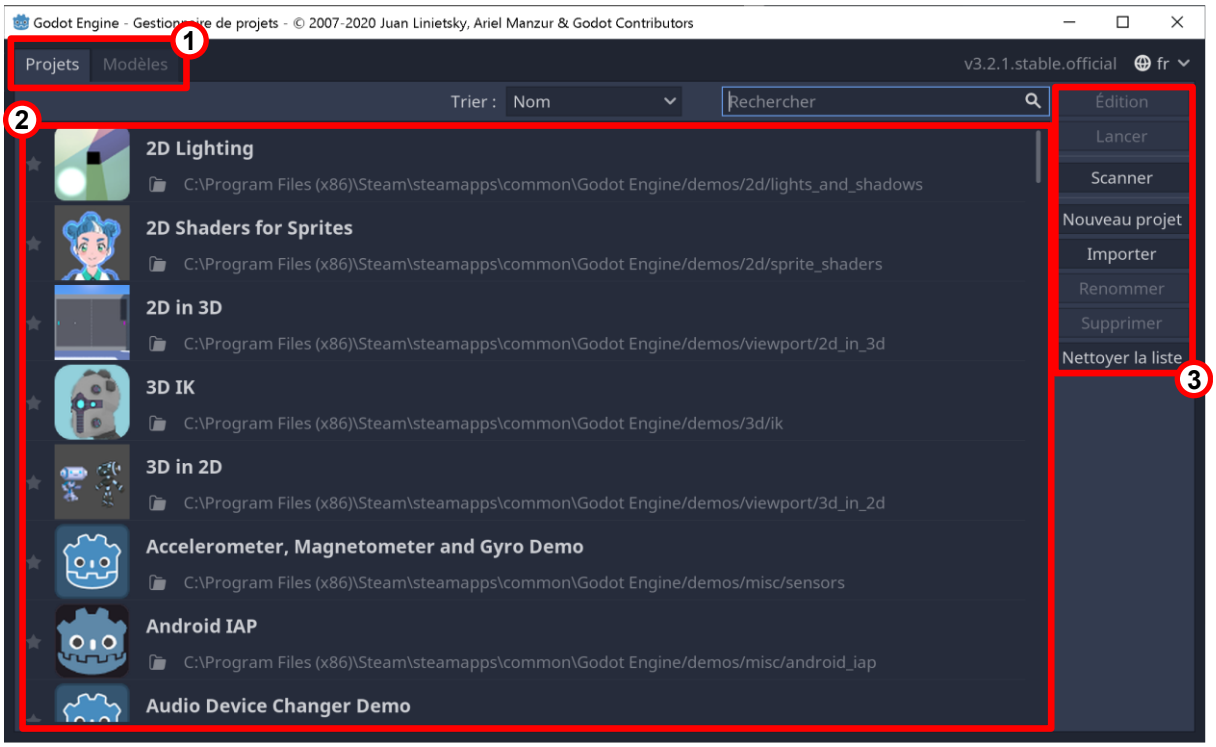

Figure 1: Interface de Godot pour le lancement d'un projet

Lorsque Godot est lancé, la première fenêtre qui s'affiche est la fenêtre pour le lancement d'un projet. Avec cette fenêtre, nous pouvons accéder soit à la liste des projets (Sur l'image : Onglet **Projet** dans l'encadré rouge n°1), soit à des projets qui sont proposés par d'autres personnes que l'on peut télécharger (Sur l'image : Onglet **Modèles** dans le même encadré rouge).

Lorsque le projet est choisi dans **la liste de projets** (Sur l'image : Encadré rouge n°2), nous pouvons le lancer de différentes manières. Dans l'encadré rouge n°3, le bouton **Edition** signifie que le projet va être modifié et va donc lancer l'éditeur de Godot (voir la Figure 2 plus loin dans le cours). Le bouton **Lancer** permet de jouer directement au projet sans l'éditer. Le boutons **Scanner** permet de choisir un dossier contenant plusieurs projets Godot et de les ajouter automatiquement à la liste. Le bouton **Importer** fait la même chose mais pour ne mettre qu'un projet dans la liste. Le bouton **Nettoyer la liste** permet de nettoyer la liste mais ne supprime pas les projets stockés sur son ordinateur.

Interface d'édition de Godot

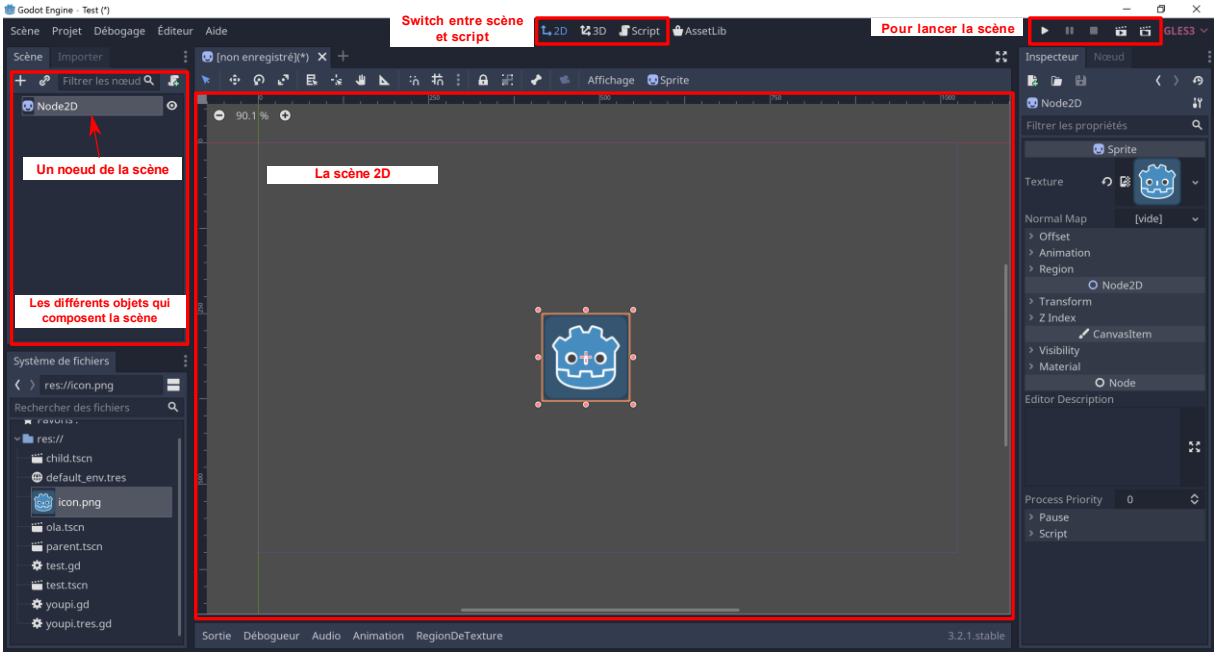

Figure 2: L'interface de Godot

L'interface d'édition de Godot est riche en information, et nous allons voir quelques zones qui nous intéressent.

La première zone en haut à gauche est la zone qui correspond à **une scène**. Une scène est tout simplement un endroit où l'on va mettre tous les objets pour constituer une partie d'un jeu. Un objet peut être n'importe quoi comme une image, une animation, une ligne, un timer, un bouton d'interface, une autre scène, … Par exemple, lorsque l'on fait le menu principal d'un jeu, cela peut être fait dans une scène ou encore lorsque l'on veut créer un niveau d'un Tetris, çà peut constituer une autre scène. Nous verrons plus tard que plusieurs scènes peuvent être combinées ensemble pour faire des scènes plus complexes. **Ce qu'il faut retenir maintenant**, c'est que l'on peut **ajouter de nouveaux objets** dans cette partie que l'on appelle **l'arbre de la scène**. Dans l'exemple de l'image, nous avons ajouté un premier objet qui est une image représentant la mascotte de Godot.

La zone principale au centre **représente la scène 2D avec tous les objets** qui sont dans l'arbre de la scène. C'est ici que l'on pourra voir et manipuler les objets, leur position, leur rotation, etc…

La zone en haut à droite permet **de lancer une scène** pour voir la scène en action (les éléments qui bougent, etc…). Ici ce qui nous intéresse c'est la **petite flèche « Play »** pour lancer la scène (son raccourci associé est *F5)* et le **petit carré « Stop »** qui permet de quitter la scène lancée. Nous verrons plus tard pour les autres boutons.

La zone en haut permet de **changer de mode d'édition**. Actuellement sur l'image, nous sommes en mode 2D, c'est-à-dire l'édition de la scène dans un espace 2D. Un second onglet correspond au mode 3D, mais que nous n'utiliserons pas pour le moment. Enfin le troisième onglet qui s'appelle Script va nous être très utile, car il permet d'ajouter ou de modifier des scripts aux objets de la scène. Il faut voir le script comme un bloc de code qui va permettre de programmer le comportement de l'objet auquel est attaché le script.

Nous avons vu durant la première séance, qu'un script était attaché à l'image d'un ballon de foot et nous avons vu qu'à partir de ce script, nous pouvions modifier la position du ballon. Nous verrons qu'il y a plein d'autres propriétés qu'il est possible de modifier dans les objets.

L'interface d'édition de script dans Godot

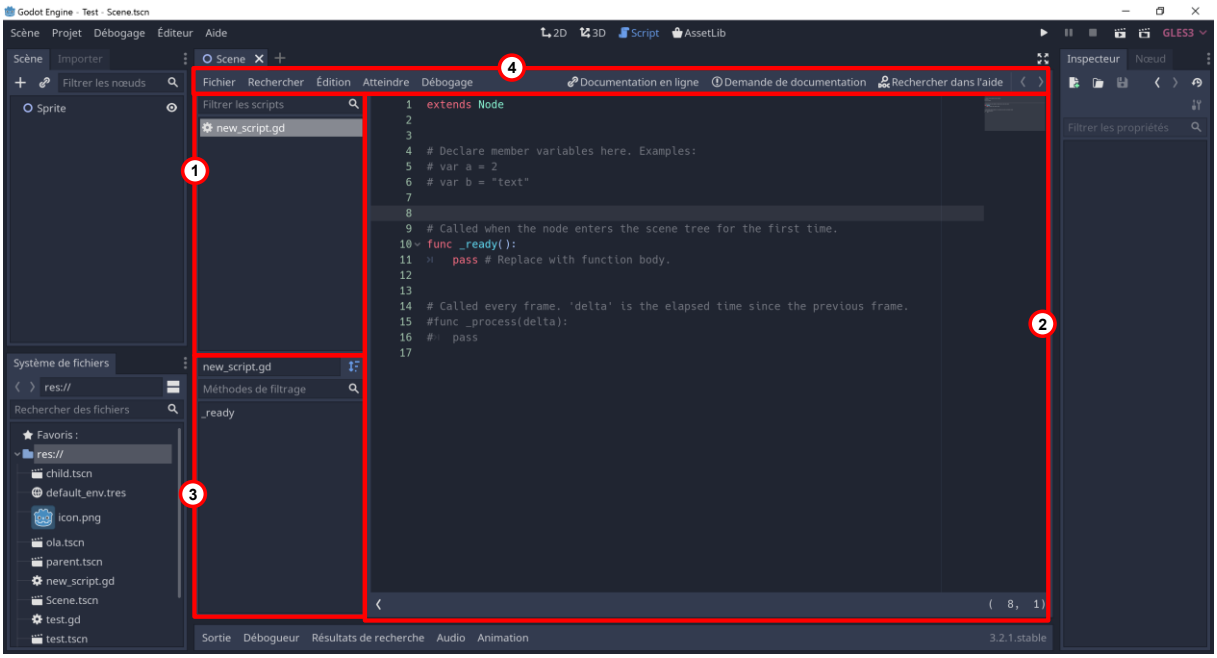

Figure 3 : Interface pour l'édition des scripts

Lorsque l'on switch dans l'interface d'édition des scripts, nous tombons sur cette fenêtre (image ci-dessus) avec un petit éditeur pour écrire du code.

La zone n°1 correspond à la **liste de scripts**, si l'on veut modifier un autre script dans la liste, il suffit d'y cliquer dessus et il apparaitra dans la zone n°2.

La zone n°2 correspond à **l'édition du script**, nous parlerons du code à l'intérieur dans la partie suivante.

La zone n°3 correspond à la liste des **fonctions** dans le script actuellement ouvert dans la zone n°2. On peut voir une fonction comme un petit morceau de code qui peut être exécuter plusieurs fois. La fonction _ready dans le code apparaît donc dans la liste.

La zone n°4 est une zone pour réaliser des actions plutôt classiques. C'est ici que l'on peut faire beaucoup de choses avec du traitement de texte, comme en créer un nouveau, faire des copier-coller, rechercher un mot dans un script, etc…

Le script GDScript

Introduction

Un script est un **programme** qui exécute une **série d'actions** à partir d'un **langage de programmation** (beaucoup de termes complexes dans une seule phrase 😉). Essayons d'expliquer chacun des termes.

Programme : cela peut être n'importe quoi, allons des gros logiciels comme Word, Photoshop ou encore ceux créés avec Livecode.

Une série d'actions peut être par exemple :

- Fais avancer le ballon vers la droite de 3 mètres
- Fais rebondir le ballon sur 2 mètres de hauteurs
- Change le ballon avec un autre ballon
- …

Cette série d'actions que l'on va appeler des instructions sont codés différemment en fonction du langage de programmation, parce que le langage de programmation est une façon pour quelqu'un de parler à la machine comme nous avons différentes façons de parler entre êtres humains (langage des signes, la langue, le texte écrit, le dessin, etc…). Par exemple, nous avons vu que programmer avec du LiveCode et du script dans Godot cette série d'instructions sont deux façons très différentes pour un même résultat. Nous verrons plus tard que les langages de programmation ne sont qu'une façon d'orthographier ce que l'on souhaite dire à la machine, car le plus important c'est de savoir ce que l'on veut dire à la machine, c'est ce qu'on appelle **l'algorithmique**.

Nous débutons avec le langage de programmation propre à Godot appelé le **GDScript.**

Le GDScript

La première chose à connaître en GDScript est la notion de **variable**. Une variable est comme une sorte de boîte pouvons contenir n'importe quoi, que ce soit un nombre, un vecteur, un point, … La variable comme son nom l'indique, peut évoluer tout au long de l'exécution du programme. Elle peut donc avoir une certaine valeur au départ et se retrouver avec une valeur différente à la fin, cela ne pose aucun problème à partir du moment que la personne qui écrit le script sait ce qu'elle fait.

La déclaration d'une variable est l'action de créer une nouvelle variable (une boîte vide). Voici l'exemple en GDScript :

```
1
2
3  var nombre;
4  var point;
5
```

Figure 4 : On vient de créer en GDScript deux nouvelles variables appelées nombre et point.

Le choix du nom de la variable n'a pas d'impact sur comment va être exécuter le code. On peut tout à fait déclarer une variable avec un nom comme : jesuisunjoueurdeviolonetjaimejouertouslesjours

Evidemment il y a des noms qui ont plus de sens que d'autres 😊. On ne va pas appeler une variable *point* si cette dernière va contenir du texte. Les seules contraintes sur les noms de variables sont les suivantes :

- Pas d'accents, ni de caractères spéciaux, ni d'espaces
- N'utiliser que des caractères alpha en minuscules (a, b, c, …, z) et des nombres entiers (1, 2, 3, …)
- Si la variable contient plusieurs mots, les séparer avec le caractère '_' comme par exemple : nombre_de_points
- Eviter des noms trop génériques pour les variables comme 'a', 'b', 'abc' ; excepté à certains moments où c'est acceptable, mais nous en reparlerons.

La langue utilisée que ce soit l'anglais, le français ou l'islandais n'a pas d'importance. Généralement si l'anglais est utilisé, c'est seulement pour une question de collaboration internationale entre développeurs si l'on veut que son code soit compris par tout le monde.

Pour affecter une valeur à une variable, c'est-à-dire remplir la boîte vide. Il suffit de le faire de la façon suivante :

```
5
6       var nombre
7       var point
8
9       nombre = 5
10      point = Vector2(1, 3)
11
```

Figure 5: On affecte la valeur 5 à la variable nombre et la valeur Vector2(1, 3) à la variable point.

Il faut bien comprendre ici que tout ce qui est à gauche et la variable qui va recevoir une valeur et tout ce qui est à droite correspond à la valeur. Une valeur peut être la valeur elle-même ou le résultat d'une opération ou encore bien d'autres choses (exemple sur l'image ci-dessous).

```
6       var nombre
7       var point
8
9       nombre = 5 + 3
10      point = Vector2(1, 3) + Vector2(3, 7)
```

Figure 6 : la variable nombre contient le résultat de l'opération 5+3, c'est à dire 8. La variable point contient le résultat de l'addition des 2 Vector2D dont le résultat est un Vector2D et la valeur -> (4, 10)

Une variable peut être déclarée et être affectée d'un valeur directement :

```
6       var nombre = 5 + 3
7       var point = Vector2(1, 3) + Vector2(3, 7)
```

Figure 7: Les variables nombre et point sont déclarées et sont affectées d'une valeur sur une seule même ligne

En ce qui concerne les opérations entre variables, on peut faire beaucoup de choses, dont les opérations que l'on va couramment utiliser sont les opérations mathématiques connues comme l'addition, la soustraction, la multiplication ou encore la division.

```
6       var nombre = 5
7       var nombre2 = 10
8
9       nombre = nombre2 # nombre vaut désormais 10
10      nombre = nombre + 8 # nombre vaut désormais 10 + 8, donc 18
11      nombre2 = -nombre # nombre2 vaut -18
12
13      nombre = (nombre2 + nombre) * 2 / 10 # que vaut nombre ici???
```

Figure 8: Exemple d'opérations possibles

On remarquera sur la figure ci-dessus des commentaires ajoutés en fin de ligne. Ces commentaires ne sont pas pris en compte lors de l'exécution du programme, ils ne sont là que pour aider la personne qui programme de ce qu'elle a fait. Il n'y a pas de contraintes pour l'écriture d'un commentaire tant que celui-ci est précédé d'un #.

Passons maintenant à ce qui nous intéresse. Les opérations possibles avec les vecteurs et les points que nous avons vu dans la partie géométrie. Tout d'abord, ce qu'il faut absolument savoir, c'est que Godot ne fait pas la différence entre un vecteur et un point, il n'utilise qu'un seul type qui est appelé ***Vector2.*** Le Vector2 pour Godot n'est qu'une boîte contenant deux nombres et c'est tout.

C'est donc à la personne qui programme de faire la différence et donc à toi de donner les bons noms de variables afin de reconnaitre ce qui est un vecteur d'un point.
Par exemple, donné le nom de variable ***position*** pour exprimer que c'est un point dans l'espace et le nom de ***deplacement*** pour exprimer un vecteur dans l'espace. La responsabilité est de ton côté jeune padawan 😊

Les opérations entre vecteurs, on les résume ici :

- Addition/Soustraction d'un vecteur avec un autre vecteur donne un autre vecteur
- Addition/Soustraction d'un vecteur avec un point donne un autre point

Essayons de faire ces opérations en GDScript :

```
6   var point_a = Vector2(0,0)
7   var point_b = Vector2(5, 5)
8   var deplacement_u = Vector2(2, 0)
9   var deplacement_v = Vector2(-4, 2)
10
11  # Addition de deux vecteurs:
12  var deplacement_w = deplacement_u + deplacement_v #resultat : Vector2(-2, 2)
13  # Soustraction de deux vecteurs
14  deplacement_w = deplacement_u - deplacement_v #resultat : Vector2(6, -2)
15  # Addition d'un point avec un vecteur
16  var point_c = point_a + deplacement_u #resultat : Vector2(2, 0)
17  # Soustraction d'un point avec un vecteur
18  point_c = point_b - deplacement_v #resultat : Vector2(9, 3)
19
20  # Soustraction de deux points
21  point_c = point_a + point_b # Aucun problème pour Godot, mais est-ce que çà a du sens?
22
23  # Multiplication de deux vecteurs
24  deplacement_w = deplacement_u * deplacement_v # Aucun problème pour Godot, mais est-ce que çà a du sens
```

Figure 9 : Différentes opérations entre points et vecteurs

On voit bien ici que le GDScript est très flexible. A toi de t'amuser à réaliser différentes opérations pour bien comprendre tout çà.

Pour finir cette fiche de cours, nous allons voir une **propriété** de l'objet sur lequel on a attaché un script. D'abord une propriété est une variable de l'objet, mais que l'on ne voit pas directement dans le script, par contre on peut y accéder sans problème. C'est comme si ces propriétés étaient stockées dans un autre script pour éviter de nous encombrer dans un nouveau script. Cependant, la propriété contrairement aux variables que l'on peut créer nous-même, influencent directement l'état de l'objet dans la scène.

La **position** de l'objet fait partie de ces propriétés et c'est un **Vector2** car, c'est un point dans l'espace 2D. Dans l'exemple ci-dessous, nous pouvons voir que nous accédons à la propriété de l'objet via le mot clé *self*. Ce mot clé désigne l'objet sur lequel on lui a attaché le script que l'on est en train d'éditer.

```
7   self.position = Vector2(4,3) # la nouvelle position est (4,3)
8
9   self.position = self.position + Vector2(10, 50) # la nouvelle position est (14, 53)
10  self.position += Vector2(10, 50) # même opération que la ligne précédente mais écrite différement
```

*Figure 10 : On modifie la propriété **position** de l'objet en y accédant via self.position. Les lignes 7 affecte une nouvelle valeur à position. La ligne 9 utilise la valeur de position pour lui ajouter en + un nouveau Vector2, l'ancienne valeur n'est donc pas perdue puisque réutilisée. Enfin la dernière ligne est l'équivalent de la ligne 9 grâce au signe += qui signifie que l'on affecte à la variable à gauche du signe sa valeur + la valeur à droite du signe.*