Introduction à la programmation : variables, types de données, conditions, boucles

Ce cours vous introduira aux concepts fondamentaux de la programmation en C# dans le contexte du moteur de jeu Unity. Vous apprendrez à manipuler des **variables**, à utiliser des **types de données**, à écrire des **conditions** et à appliquer des **boucles** pour créer des logiques de jeu simples et interactives.

---

1. Introduction à la Programmation en C#

C# (prononcé "C-sharp") est un langage de programmation moderne, utilisé dans Unity pour écrire des scripts qui contrôlent la logique du jeu. Le langage C# est puissant et flexible, et il est principalement utilisé dans le développement de jeux et d'applications.

Dans Unity, vous créez des scripts C# pour interagir avec la scène, les objets, et gérer les comportements du jeu (comme les mouvements, les animations, et les interactions).

---

2. Variables et Types de Données

Qu'est-ce qu'une Variable ?

Une **variable** est un espace mémoire où vous pouvez stocker des informations. Vous pouvez modifier la valeur d'une variable pendant l'exécution du jeu. Chaque variable a un **type de donnée** qui détermine quel genre d'information elle peut stocker.

Déclaration d'une Variable

En C#, une variable est déclarée en indiquant le type de donnée, suivi du nom de la variable.

Exemple :

```
int score; // Déclare une variable 'score' de type entier
(int)
```

Types de Données Communs en C#

1. **int** (entier) : Utilisé pour les nombres entiers (ex : -5, 0, 42).

   ```
   int score = 100;
   ```

2. **float** (nombre à virgule flottante) : Utilisé pour les nombres décimaux (ex : 3.14, -1.0).

   ```
   float speed = 3.5f; // Notez le suffixe 'f' pour indique
   r un float
   ```

3. **bool** (booléen) : Utilisé pour représenter une valeur vraie ou fausse.

   ```
   bool isGameOver = false;
   ```

4. **string** (chaîne de caractères) : Utilisé pour stocker des textes.

   ```
   string playerName = "Alex";
   ```

5. **Vector3** : Type spécifique dans Unity pour stocker des coordonnées dans l'espace 3D (x, y, z).

   ```
   Vector3 playerPosition = new Vector3(0, 1, 0);
   ```

---

3. Conditions (if, else, else if)

Les **conditions** sont utilisées pour exécuter un code spécifique si une condition donnée est vraie ou fausse. Les conditions les plus courantes en C# sont les instructions `if`, `else if`, et `else`.

Structure de base d'une condition

```
if (condition)
{
    // Code exécuté si la condition est vraie
}
else
{
    // Code exécuté si la condition est fausse
}
```

Exemple de condition simple

```
int score = 100;

if (score >= 50)
{
    Debug.Log("Vous avez gagné !");
}
else
{
    Debug.Log("Vous avez perdu !");
}
```

Instruction `else if`

Cela permet de tester plusieurs conditions successivement.

```
int score = 75;

if (score >= 90)
{
    Debug.Log("Excellent !");
}
else if (score >= 50)
{
    Debug.Log("Bon travail !");
}
else
```

```
{
    Debug.Log("Essayez encore !");
}
```

---

4. Boucles (for, while, foreach)

Les **boucles** permettent de répéter des actions plusieurs fois en fonction d'une condition.

Boucle `for`

La boucle `for` est utilisée lorsque vous savez à l'avance combien de fois vous devez répéter une action.

```
for (int i = 0; i < 5; i++) // Répète 5 fois
{
    Debug.Log("Iteration " + i);
}
```

Dans cet exemple, la boucle commence avec `i = 0` et continue tant que `i` est inférieur à 5. À chaque itération, `i` est incrémenté de 1.

Boucle `while`

La boucle `while` répète une action tant qu'une condition est vraie.

```
int i = 0;
while (i < 5)
{
    Debug.Log("Iteration " + i);
    i++; // Incrémenter i
}
```

Ici, la boucle continue tant que `i` est inférieur à 5. Il est important de ne pas oublier de modifier la variable conditionnelle (`i++`) pour éviter une boucle infinie.

Boucle `foreach`

La boucle `foreach` est utilisée pour itérer sur des collections comme des tableaux ou des listes.

```
string[] fruits = { "Pomme", "Banane", "Orange" };
foreach (string fruit in fruits)
{
    Debug.Log(fruit);
}
```

Cela affiche chaque fruit de la liste un par un.

---

5. Exemple Pratique : Créer un Compteur de Score

Voici un exemple simple où nous utilisons des variables, des conditions et des boucles pour créer un système de score.

Script C# : Compteur de Score

1. Créez un nouveau script en C# dans Unity et nommez-le **ScoreManager**.
2. Collez le code suivant dans votre script :

```
using UnityEngine;

public class ScoreManager : MonoBehaviour
{
    int score = 0;

    void Start()
    {
        Debug.Log("Début du jeu");
    }

    void Update()
    {
        if (score >= 100)
        {
            Debug.Log("Vous avez gagné !");
        }
        else if (score >= 50)
```

```
        {
            Debug.Log("Bon travail !");
        }
        else
        {
            Debug.Log("Score actuel : " + score);
        }

        // Simuler l'augmentation du score à chaque seconde
        score += 1;  // Augmente le score de 1 à chaque fra
me
    }
}
```

Explication du code :

- **Variable** `score` : Un entier qui représente le score du joueur.
- **Méthode** `Start` : C'est ici que le jeu commence, vous pouvez initialiser des valeurs.
- **Méthode** `Update` : Cette méthode est appelée à chaque frame du jeu. Nous vérifions si le score atteint certaines valeurs pour afficher des messages correspondants.
- **Condition** `if` : Si le score est supérieur ou égal à 100, un message de victoire est affiché. Si le score est supérieur ou égal à 50, un message de bon travail est affiché.
- **Boucle de score** : À chaque frame, le score est augmenté de 1 (simulation d'une augmentation de score dans un jeu).

---

6. Conclusion

En C# pour Unity, vous avez appris les bases de la programmation, telles que les variables, les types de données, les conditions, et les boucles. Ces éléments sont fondamentaux pour créer des comportements dans un jeu et contrôler la logique de votre application. Vous pouvez maintenant manipuler les variables pour ajuster des aspects du jeu (comme le score) et utiliser des conditions pour modifier l'état du jeu en fonction de certaines règles. Les

boucles vous permettent de répéter des actions de manière efficace, ce qui est crucial pour le développement de jeux.

Exercices Complémentaires :

1. Créez un script où le joueur a un **compteur de vies** qui diminue à chaque fois qu'il touche un obstacle.
2. Utilisez une **boucle** `for` pour créer une série d'ennemis dans votre scène et appliquez-leur des comportements spécifiques.
3. Créez un **système de niveaux** où un **score minimum** est requis pour passer au niveau suivant.