TP C# : TinyPacman

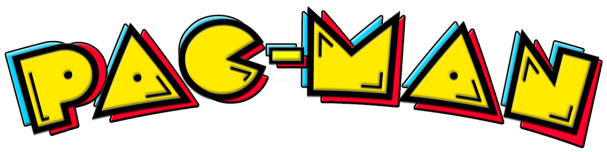


Table des matières

1 Introduction

Cet atelier s'adresse a des étudiants possédant déjà quelques bases en programmation. Ne vous inquiétez pas, nous vous guideront tout au long de cet atelier pour que vous ne soyez pas perdus.

L'objectif de cet atelier est de vous faire créer un petit jeu Pac-Man en console. Pour cela nous utiliseront le langage C# . Nous vous fournissons déjà les bases du projet pour que vous puissiez bien commencer. Pendant cet atelier vous êtes encadrés par plusieurs étudiants et enseignants de l'EPITA. N'hésitez pas à nous poser des questions ! Le but de cet atelier est de vous faire réaliser un petit Pac-Man en console. Vous trouverez en annexe des exemples de ce que vous serez capable de réaliser avec ce module et les connaissances acquises au cours de cet atelier.

2 Explications

2.1 Objets

Durant cet atelier nous utiliseront la programmation orientée objet en C# . Il s'agit d'un paradigme de programmation, une représentation des problèmes et des solutions associées qui va changer la manière d'exprimer les idées et la façon d'approcher un problème.

Les classes sont une manière de représenter un type de donnée via des propriétés. On peut créer des instances de classe que l'on appelle Objets.

Voyez une classe comme un moule et un objet comme un gâteau. Le moule permet de donner la forme générale mais le gâteau peut quand même avoir ses particularités pour le différencier des autres gâteaux créés à partir de ce moule. Les classes sont très utiles pour créer et traiter de la même manière des objets possédant des propriétés communes.

Il existe deux types de propriétés : les attributs et les méthodes. Les Attributs sont des données, des variables déclarées dans la classe. Les attributs peuvent être :

- **statiques**, ils existent dans la classe et ont la même valeur pour toutes les instances de la classe.
- **d'instance**, ils existent dans les objets et peuvent avoir des valeurs différentes pour chaque instance de la classe.

Les méthodes sont des fonctions définies dans la classe qui vont pouvoir donner un comportement aux objets.Les méthodes peuvent être :

- **statiques**, elles sont alors liées à la classe et peuvent être appelées sans avoir instancié d'objet.
- **d'instance**, elles ne peuvent être appelées que sur un objet instancié et dépendent des attributs de l'objet.

2.2 Les énumérations

Un type énumération (également appelé énumération ou enum) offre un moyen efficace de définir un ensemble de constantes intégrales nommées qui peuvent être assignées à une variable. Par exemple, supposez que vous devez définir une variable dont la valeur représente un jour de la semaine. Il peut devenir vite illisible d'utiliser 7 variables ou un tableau days 7 entrées pour les représenter. C'est pour nous faciliter la vie que l'énumération (enum) a été créée.

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
```

Voici un petit exemple pour vous éclairer :

```
public enumDay {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
public static voidPrintDay(Day day)
{
    Console.WriteLine("Let's take a look at " + day.ToString());
    if(day <= Day.Fri)
        Console.WriteLine("It's a work day");
    else
        Console.WriteLine("Week end !");
}
```

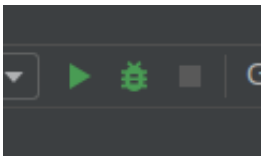
Vous l'aurez compris l'avantage pricipal des enums par rapport à un type numérique est que vous spécifiez clairement pour le code client les valeurs qui sont valides pour la variable.

3 Rider

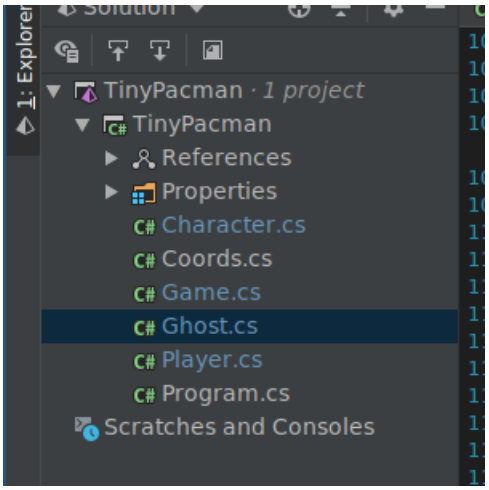
Rider est le nom de l'environnement de travail que vous allez utiliser. Ceux qui connaissent Visual Studio ne seront pas perdus.

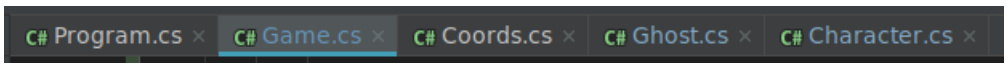
Un projet de C# est représenté par une *solution*. Pour commencer à coder, ouvrez la solution que nous vous avons fourni.

Si vous cliquez sur Run puis sur Run 'Default', votre programme est *compilé* puis exécuté dans une console interne. Un raccourci est accessible en haut à droite, la petite flèche verte. Le raccourci clavier associé est Ctrl + F5 (vous pouvez aussi utiliser F5, qui lance le mode Debug).



Vous pouvez vous deplacer entre les different fichier du projet dans l'explorateur. En double cliquant sur un fichier un nouvel onglet s'ouvre avec le fichier en question :





Tip

Pour les exercices qui vont suivre, si vous bloquez n'hésitez surtout pas à nous appeler. Certains concepts peuvent être difficile a assimiler. Une correction est également disponible ici si vous voulez vérifier vos résultats ou vous débloquer : ici

4 Character et Player

Le rôle de la classe `Player` est de contrôler tous les comportement de notre Pac-Man. C'est une extension de la classe `Character`, elle utilise donc une icône, une direction et un système de coordonnées [1].

1. On dit que la classe `Player` *hérite* de la classe `Character`

4.1 Player : Print

Tip

pos est un objet de type Coords. Qui représente... des coordonnées. Pour accéder à la position en x de player on utilisera donc `player.pos.x` de même pour y.

Commencez par faire la fonction `Print()` de la classe `Character` qui doit afficher l'icône du joueur à la position de la console qui correspond à ses coordonnées à l'aide de la fonction `Console.SetCursorPosition()`.

4.2 Player : GetInput

Votre travail consiste maintenant à compléter la classe `Player` pour permettre de contrôler et afficher notre ami tout rond tout jaune.

Pour cela, vous devrez écrire la fonction `GetInput()`, qui détermine la direction dans laquelle le joueur veut aller en fonction de la touche du clavier enfoncée. On récupère la touche avec `Console.ReadKey()`. Voici un exemple d'utilisation :

```
ConsoleKeyInfo key = Console.ReadKey();
if (key.Key == ConsoleKey.Space)
    Console.WriteLine("<Space> Key pressed");
```

Si vous essayez ce code, vous remarquerez que le programme s'arrête et attend lorsqu'il atteint `Console.ReadKey()`. Ça nous laisse le temps d'appuyer sur la touche pour voir ce qu'il se passe, mais pour un jeu, on ne peut pas se permettre d'attendre ! On rajoute donc l'argument `true` pour que l'appel ne soit pas *bloquant*.

Ah, et plutôt que d'enchaîner les `if (key == ...)`, voici comment utiliser un `switch` :

```
ConsoleKeyInfo key = Console.ReadKey(true);
switch (key.Key)
{
    // Cas où key.Key vaut W ou UpArrow
    case ConsoleKey.W:
    case ConsoleKey.UpArrow:
        dir = Direction.Up;
        break;
    ...
}
```

5 Game

Cette classe représente le plateau de jeu dans lequel le joueur et les ennemis se déplaceront.

5.1 Game : PrintMap

`PrintMap()` affiche la carte du jeu a l'écran. C'est a vous de décider par quel caractère vous voulez représenter les murs, pacgommes et espaces vides. Le type d'une case est stocké dans une énumération appelée `CellType`. C'est une variable spéciale qui peut valoir `Wall`, `Pacgum` ou `Empty`. Pour chaque case, vous allez donc déterminer son type, de la même façon que dans `GetInput()` :

```
// map[0, 0] est de type CellType
switch (map[0, 0])
{
    // Si c'est vide
    case CellType.Empty:
        Console.Write(' ');
        break;

    // Si c'est un mur
    case CellType.Wall:
        ...

    // Si aucun des cas précédents n'a été identifié
    default:
        Console.Write('?');
        break;
}
```

Les deux fonctions les plus importantes sont sûrement `Launch` et `Update`.

5.2 Game : Update

`Update()` mets à jour l'état du jeu. Premièrement, elle doit vérifier si une condition d'arrêt est rencontrée : la variable `gameIsRunning` doit passer à `false` si `timer` atteint 0 ou si `GetPacgum()` renvoit 0 (quand on a tout mangé). Ensuite, si le jeu continue, elle doit appeler la fonction `MovePlayer()` et décrémenter `timer`.

5.3 Game : Launch

`Launch()` s'occupe de préparer et faire tourner la partie. C'est la fonction de plus haut niveau du jeu. D'abord, on appelle `PrintMap()`. On le fait avant la boucle principale parce que les objets statiques ne vont pas bouger, donc pas besoin de les actualiser. Ensuite, tant que la variable `gameIsRunning` est à `true`, on exécute les instructions récurrentes qui constituent la *boucle de jeu* principale :

— On récupère les entrées de l'utilisateur en appelant `Player.GetInput()`
— On met les données à jour avec `Update()`
— On affiche la nouvelle position du joueur avec `player.PrintPlayer()` et le score avec `PrintScore()`

— Pour ralentir le jeu et éviter les bugs d'affichage liés au temps de rafraîchissement de l'écran, on rajoute un petit `System.Threading.Thread.Sleep(150)`

Enfin pour faire joli on affiche un `"Game Over"` en dehors de la boucle : si on sort de la boucle, c'est que le jeu est terminé.

Vous pouvez maintenant tester votre jeu !

6 Ghosts

Des ennemis ! Du danger ! De l'action ! Du frisson ! Que serait Pac-Man sans ses iconiques fantômes ?

Pour l'instant, ils attendent sagement dans leur chambre sans rien faire. C'est normal, la fonction qui leur sert de cerveau est vide !

6.1 Ghosts : Move

À vous de la recoder pour leur permettre de bouger. Pour commencer simplement, vous pouvez leur faire choisir une direction aléatoire chaque fois que la fonction est appelée. Voici un exemple de génération de nombres aléatoire :

```
Random rng = new Random();
Console.WriteLine(rng.Next());
Console.WriteLine(rng.Next(5));
Console.WriteLine((Character.Direction)rng.Next(5));
```

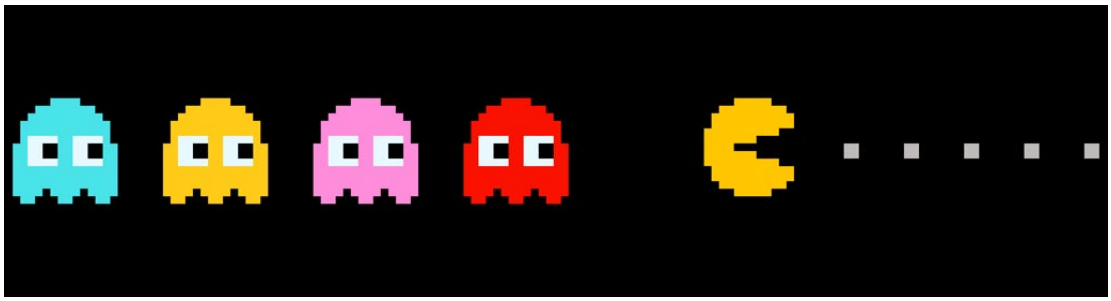
Mais si vous voulez avoir de vrais fantômes un minimum menaçant, il faudrait qu'ils essaient de se rapprocher de Pac-Man à chaque instant... C'est a vous de jouer, Implémentez une intelligence aux fantômes.

7 Les bonus

Maintenant que vous avez terminé le TP vous pouvez améliorer votre Pac-Man en ajoutant par exemple :

— Des fruits : Des pacgommes qui rapportent plus de points ou qui vous permettent de "dégommer" les fantômes.
— Des sons : Rajoutez des effet sonores lorsque votre Pac-Man mange des pacgommes. Om-Gnom-Gnom.
— Des animations : Changez l'apparence de Pac-Man quand il change de direction ou bien quand il mange un pacgomme.
— Des vies
— ...

Vous pouvez retrouver a tout moment ce TP a cette adresse : https://github.com/Lyx09/Immersion_pacman



I see no point in coding if it can't be beautiful.