

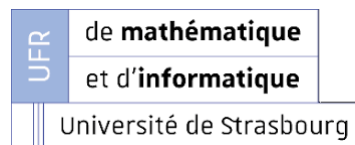
RENDU DE PROJET

« Pacman Namco (1980) »

Programmation Avancée

UFR Math-Info - Université de Strasbourg

1^{ère} année de Master SIL



Date de rendu : 07 mai 2023

Par

Matthieu FREITAG

<https://git.unistra.fr/chamaael/pec-men>

Index

1	Introduction	1
2	Fonctionnalités et choix	2
2.1	Les utilitaires	2
2.2	Les éléments de configuration	2
2.3	La logique de jeu	3
2.4	Le plateau	3
2.5	Les entités	4
2.6	L’affichage	5
3	Difficultés rencontrées	6

Introduction

Ce projet vise à réaliser un remake du jeu Pacman, sorti en 1980 et développé par Namco.

Approche proposée

La programmation se fera en C++ moderne. L'affichage se fera avec la librairie SDL2 et uniquement par copie de sprite à partir d'une ou plusieurs planches (images en format BMP). La librairie SDL2 est la seule autorisée.

Nous effectuerons ce projet en équipe de deux en nous basant sur les cours ainsi que les travaux pratiques de la matière Programmation avancée.

Voici les documents complémentaires à ce projet :

1. Le cahier des charges (non exhaustif) :
<https://www.gamedeveloper.com/design/the-pac-man-dossier>
2. Le coding-style :
<https://google.github.io/styleguide/cppguide.html>
3. Un exemple :
<https://www.youtube.com/watch?v=7O1OYQRqUag>

Ce rapport traitera uniquement des fonctionnalités que j'ai moi même implémenté. Pour toute information complémentaire, se référez au fichier README accessible depuis le repository.

Fonctionnalités et choix

Je ne pense pas pouvoir justifier convenablement en moins de quatre pages. De ce fait, je vais me concentrer sur les éléments qui peuvent paraître moins évident. Voici l'ensemble des fonctionnalités que j'ai implémenté.

2.1 Les utilitaires

- Direction :
une direction (haut, gauche, bas, droite).
- Counter :
compte le nombre de ticks jusqu'à atteindre une certaine valeur.
- Timer : (n'est plus utilisée)
démarré un chronomètre asynchrone (en millisecondes).
- Container : (n'est plus utilisée)
template pour une array.
- des fonctionnalités partagées dans le fichier `utils`.
- bonus : le CI/CD sur le repository GitLab.

2.2 Les éléments de configuration

Les éléments de configuration sont réparti en deux fichiers distinct : le premier gère les informations générales du jeu alors que le deuxième traite des informations visuelles. Ces fichiers représentent des namespace imbriqués. Ce n'est peut-être pas la méthode la plus optimale mais j'ai opté pour cette méthode car elle apporte beaucoup de simplicité d'utilisation, d'organisation et permet de facilement modifier les éléments voulus même si la personne ne connaît pas le code.

La configuration du plateau est chargée depuis le fichier `pacman_map.txt` où chaque chiffre indique le type d'une cellule.

2.3 La logique de jeu

Un tour de boucle est exécuté à interval de temps régulier. Techniquement, l'utilisateur a la possibilité d'indiquer une nouvelle direction lors de chaque tick. Un tick vérifie le statut actuel et le met éventuellement à jour. Si le statut autorise la reprise du jeu (i.e. le jeu n'est pas arrêté), alors l'ensemble des entités du jeu sont également traitées. Ensuite, on vérifie la présence d'éventuelles collisions, auquel cas les statuts associés et le jeu sont modifiés en conséquence. Une fois le tick achevé, l'affichage est également actualisé.

2.4 Le plateau

Une position est représentée par la classe `Position` qui indique une paire de coordonnées et possède de nombreuses méthodes utilitaires. Pour l'instant, cette paire est de type `int` mais mon binôme devrait la transformer en type dynamique. Ceci permettrait d'utiliser des positions `float` et de finalement avoir des vitesses dynamiques conforme au jeu original.

Une cellule est représentée par la classe `Cell` et indique une position. Elle peut éventuellement posséder une entité statique qui est un `std::shared_ptr` ce qui permet de déclarer ou non une entité mais aussi de modifier l'entité directement depuis les méthodes de gestion du jeu.

Le plateau est représenté par la classe `Map` et implémente des méthodes permettant de vérifier la validité des mouvements que peuvent tenter les entités. Cette classe présente une liste de cellules et des membres utiles à l'affichage ou au paramétrage global du plateau. En réalité, cette classe possède deux listes de cellules : la première représente l'ensemble des cellules tandis que la seconde représente uniquement les cellules possédant des entités. Cette séparation permet de gagner en performance lors des boucles d'affichage, de reset et de gestion des entités. En plus de cela, les listes sont au format `vector<shared_ptr>` ce qui permet d'appliquer directement la modification d'une cellule à l'ensemble des listes.

2.5 Les entités

Une entité est représentée par la classe `Entity`. Cette dernière comprend un statut (visible, cachée, morte, figée,...), une position, un sprite, une valeur etc. Cette entité est statique (i.e. ne peut pas se déplacer) et est directement utilisée par les éléments `Pellet` et `Energizer` du jeu. Elle possède également une méthode qui permet de gérer son statut et paramètres à chaque tick.

Un fruit est représenté par la classe `FruitObject` qui indique une valeur, une liste des niveaux auxquels il apparaît ainsi qu'une animation. L'ensemble des fruits est géré par la classe `Fruit` qui étend la classe `Entity`. Elle inclue une liste de fruits (i.e. `FruitObject`) et permet de gérer l'activation et la désactivation des fruits.

Une entité qui peut se déplacer est représentée par la classe `MovingEntity` et étend la classe `Entity`. Elle comprend une vitesse, des animations (une par direction) et des paramètres relatifs à ses déplacements. Elle gère sa direction et formule des requêtes de mouvement au plateau. Ses paramètres sont mis à jour d'après le résultat de ces requêtes.

L'entité principale du jeu est représentée par la classe `Pacman`. C'est une classe simple qui tire la quasi-totalité de ses fonctionnalités depuis `MovingEntity`. En effet, elle permet simplement de gérer l'animation de sa mort.

Un fantôme est représenté par la classe `Ghost` qui étend `MovingEntity`. Un fantôme possède un statut supplémentaire qui lui est propre ainsi que d'autres animations relatives à ce statut. Ces paramètres sont mis à jour à chaque exécution de tick.

Il existe également des fantômes dont les comportements varient selon leur type. C'est la classe `GhostSpecial` (étendant `Ghost`) qui les représentent. Cette classe possède un type dynamique (selon le type du fantôme) qui permet d'avoir plusieurs méthodes ayant le même intitulé mais des comportements qui diffèrent, très utile pour le calcul des cibles en mode Chase.

L'ensemble des fantômes est représenté et géré au sein de la classe `Ghosts`. Cette dernière permet de synchroniser les changements de statut et/ou réinitialisation par exemple.

2.6 L'affichage

Une image affichée à l'écran est représentée par la classe `Sprite`. Elle indique une position sur la bitmap (non modifiable) ainsi qu'une position sur la fenêtre (modifiable à chaque tick). Cette dernière n'est pas forcément la position finale sur l'écran puisqu'il est possible d'y ajouter un offset permettant notamment de placer l'image au centre d'une cellule.

Une animation est représentée par la classe `Animation`, composée d'une liste de sprites et d'autres paramètres permettant de personnaliser chaque animation avec notamment un nombre de ticks entre chaque swap.

Difficultés rencontrées

J'ai rencontré quelques difficultés lors de l'implémentation des mouvements et de la logique permettant d'autoriser ou non un mouvement. Ce fut une phase très complexe qui m'a demandé beaucoup de temps, de réflexion et de tests, notamment car la logique globale employée par Pacman et les Ghost n'est pas totalement identique. Ainsi, j'ai du reprendre et revoir la logique des mouvements après l'implémentation des fantômes.

Hormis cet élément, je ne pense pas avoir rencontré de difficulté particulière si ce n'est le temps car j'ai trouvé ce projet très chronophage.