BORDEAUX INP Enseirb-Matmeca

Rapport

Projet de Programmation Pac-Man

Rédigé par :
Tribak Noussayba
Ben Amor Eya

Encadrant :
Swartvagher Philippe

23 avril 2025

1 Introduction

Ce rapport documente notre projet de programmation ayant pour objectif de recréer le jeu Pac-Man en utilisant la bibliothèque SDL. Dans ce rapport, nous décrirons notre approche de développement, les défis que nous avons rencontrés, les solutions que nous avons adoptées .

Présentation du travail réalisé

Le développement du jeu Pac-Man a été encadré par une série de tâches progressives, allant de l'implémentation des déplacements de base à des fonctionnalités avancées comme la génération procédurale de labyrinthes ou l'intelligence des fantômes.

Dans ce rapport, nous nous concentrerons principalement sur trois aspects spécifiques du projet : la mise en œuvre de l'algorithme qui permet aux fantômes de suivre Pac-Man, la génération aléatoire de labyrinthes en utilisant une méthode de fusion, ainsi que la création d'un éditeur de cartes interactif permettant de concevoir manuellement des niveaux personnalisés.

2 Diriger les fantômes vers Pac-Man

Afin de déterminer la direction optimale que doivent prendre les fantômes pour poursuivre efficacement Pac-Man, nous avons implémenté et comparé trois approches : la **distance euclidienne**, la **distance de Manhattan**, et la **recherche en largeur (BFS)**.

2.1 Distance Euclidienne

La distance euclidienne correspond à la distance "en ligne droite" entre deux points dans un espace continu :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Avantages :
— Représente fidèlement la distance réelle entre deux positions dans un espace ouvert.
— Simple à implémenter et rapide à calculer.
Inconvénients :
— Ne prend pas en compte les murs ou les obstacles du labyrinthe.
— Peut suggérer des directions inaccessibles.

2.2 Distance de Manhattan

La distance de Manhattan additionne les déplacements verticaux et horizontaux :

$$d = |x_2 - x_1| + |y_2 - y_1|$$

Avantages :
— Parfaitement adaptée aux déplacements sur grille.
— Moins coûteuse en calcul que la distance euclidienne (pas d'opérations flottantes).
Inconvénients :
— Ne tient pas compte non plus des obstacles.
— Moins représentative dans un espace réel.

2.3 Recherche en largeur (BFS)

L'algorithme de *Breadth-First Search*[1] explore la carte à partir de la position du fantôme, jusqu'à atteindre celle de Pac-Man, tout en respectant les contraintes du labyrinthe.

Avantages :

— Prend en compte les murs et les obstacles.

— Garantit le chemin le plus court (en nombre de mouvements).

Inconvénients :

— Plus gourmand en mémoire et en temps de calcul.

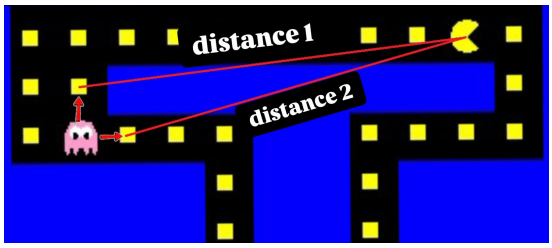
— Moins adapté aux grands environnements sans optimisation.

2.4 Comparaisons

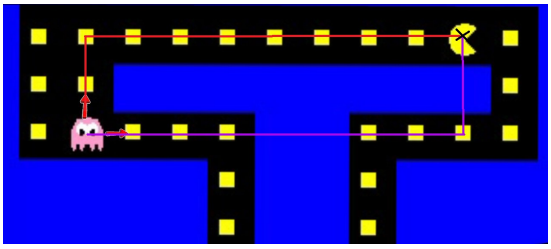
Comparaison entre la distance Euclidienne et la distance de Manhattan

Dans un labyrinthe comme celui de Pac-Man, les déplacements sont restreints aux axes verticaux et horizontaux. La distance de Manhattan est donc plus pertinente car elle reflète précisément le nombre de mouvements possibles, en respectant les contraintes de déplacement.

À l'inverse, la distance euclidienne, bien qu'elle fournisse une estimation plus fidèle dans un espace continu, peut sous-estimer la distance réelle sur une grille. Toutefois, dans notre cas, cette limite est partiellement contournée puisque seules les directions valides (c'est-à-dire sans obstacle) sont considérées comme options lors du calcul — ce qui évite les choix de direction non réalisables, bien que cette approche puisse parfois rester moins efficace que d'autres méthodes plus adaptées à une structure en grille.



(a) Distance euclidienne
Direction choisie : droite. Cette méthode estime que ce chemin est plus court car $distance_1 < distance_2$.



(b) Distance de Manhattan
Les deux directions ont la même distance → pas de préférence claire → choix arbitraire selon l'ordre d'évaluation.

FIGURE 1 – Comparaison visuelle entre la distance euclidienne et la distance de Manhattan sur une même grille.

Résumé comparatif Ces constats peuvent être résumés de manière synthétique dans le tableau suivant :

Méthode	Gestion des obstacles	Précision	Complexité	Vitesse
Euclidienne	Non	Moyenne	Faible	Rapide
Manhattan	Non	Moyenne	Très faible	Très rapide
BFS	Oui	Très élevée	Moyenne	Plus lente

TABLE 1 – Comparaison des différentes méthodes de calcul de direction pour les fantômes.

2.5 Conclusion

Dans un environnement structuré comme le labyrinthe de Pac-Man, l'algorithme de **BFS** est le plus pertinent pour un comportement intelligent, car il tient compte de la topologie du terrain. Cependant, pour des comportements secondaires, ou lorsqu'une exécution rapide est nécessaire, les distances *euclidienne* et *Manhattan* offrent un bon compromis entre performance et simplicité.

3 Éditeur graphique de labyrinthe

Pour simplifier la création des cartes du jeu, un éditeur graphique a été développé avec la bibliothèque SDL. Il permet de construire visuellement un labyrinthe en cliquant directement sur les tuiles de la grille. Le type d'élément à placer (mur, chemin, départ de Pac-Man, fantôme, Mur secret) .

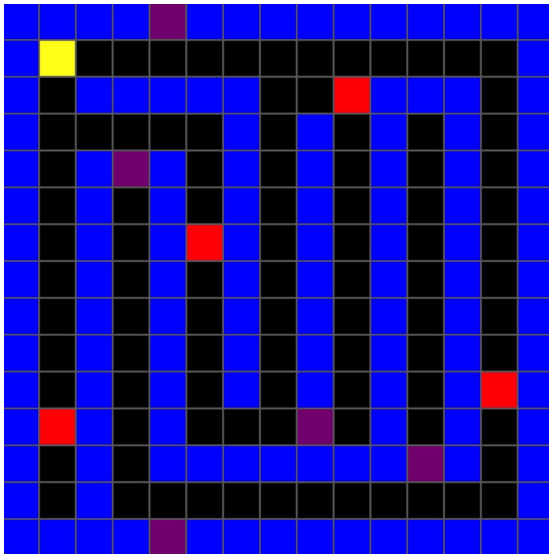
Lors du lancement, l'utilisateur indique le nom du fichier de sauvegarde, ainsi que les dimensions de la carte (nombre de colonnes et de lignes). Chaque tuile est représentée dans un tableau 2D, qui est ensuite exporté dans un fichier texte .

Voici les correspondances entre les touches et les types de tuiles :

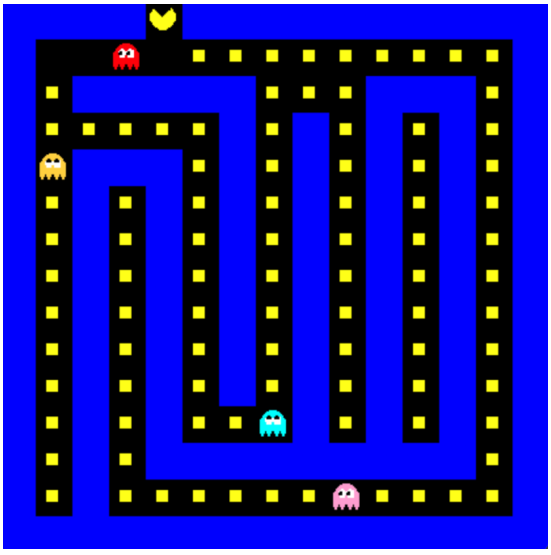
— `1` : Mur (bleu)
— `0` : Chemin (noir)
— `2` : Départ de Pac-Man (jaune)
— `3` : Départ de fantôme (rouge)
— `4` : Mur secret (violet)
— `s` : Sauvegarde de la carte

Grâce à l'interface interactive, il est possible de construire rapidement un labyrinthe personnalisé en visualisant directement les couleurs à l'écran.

Les figures suivantes illustrent le processus de création : l'éditeur graphique en cours d'utilisation, suivi du labyrinthe final construit manuellement.



(a) Interface de l'éditeur pendant la création d'un labyrinthe



(b) Labyrinthe final généré avec l'éditeur

FIGURE 2 – Création manuelle d'un labyrinthe à l'aide de l'éditeur graphique

4 Génération automatique de labyrinthes

Pour générer un **labyrinthe parfait**[2], on utilise l'algorithme de **fusion aléatoire de chemins**. Il repose sur l'idée que chaque cellule doit être reliée à toutes les autres par un chemin unique, sans boucle.

Le labyrinthe est représenté sous forme d'une grille. Seules les cellules situées à des coordonnées impaires (comme (1,1), (1,3), etc.) sont considérées comme des cases de passage. Les autres représentent les murs, qui séparent les cellules.

Chaque case de passage reçoit un **identifiant unique**, ces identifiants peuvent être définis librement. Par exemple, on peut utiliser la formule `id[y][x] = y * N_X_TILES + x;` . Les cases murales, elles, reçoivent toutes le même identifiant, soit 0.

Ces identifiants servent à savoir si deux cases sont déjà reliées ou non. Si deux cases ont des identifiants différents, cela signifie qu'elles ne sont pas encore connectées.

Deux cases accessibles ayant chacune un identifiant unique sont considérées comme voisines si elles sont alignées verticalement ou horizontalement, avec une seule case murale entre elles .

Relier deux cases signifie donc que la case située entre elles, qui représentait un mur, devient une case de passage.

L'algorithme fonctionne ainsi : une cellule est choisie au hasard, ainsi qu'une de ses voisines. Si leurs identifiants sont différents, les deux cases seront reliées. L'identifiant de l'une des deux cellules est alors attribué à l'autre, et à toutes les autres cases déjà reliées avec elle.

On répète cette opération exactement $n-1$ fois, où $n$est le nombre total de cellules accessibles. Cela signifie que toutes les cellules accessibles partagent le même identifiant à la fin.

C'est une propriété fondamentale de la **théorie des graphes** : un arbre couvrant sur n sommets possède toujours $n-1$ arêtes, ce qui garantit une connexion complète sans boucle.

Les schémas ci-dessous illustre, étape par étape, la construction d'un labyrinthe de taille 7×7 à l'aide de la méthode de fusion d'ensembles.

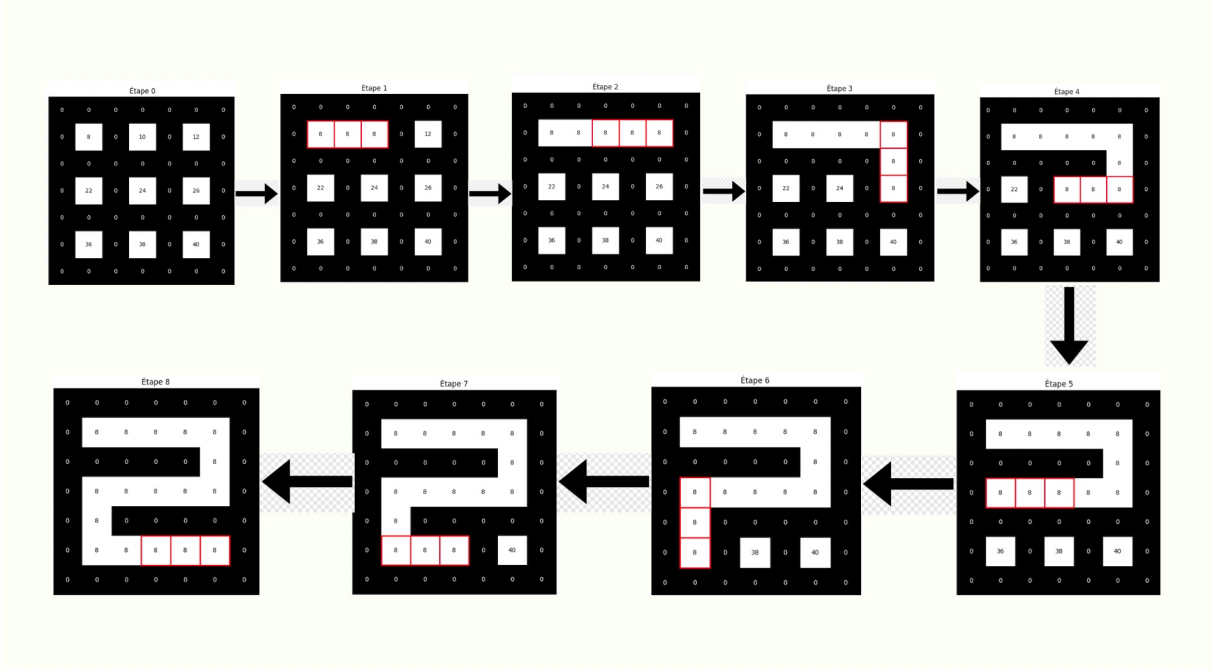


FIGURE 3 – Construction étape par étape d'un labyrinthe via la méthode de fusion.

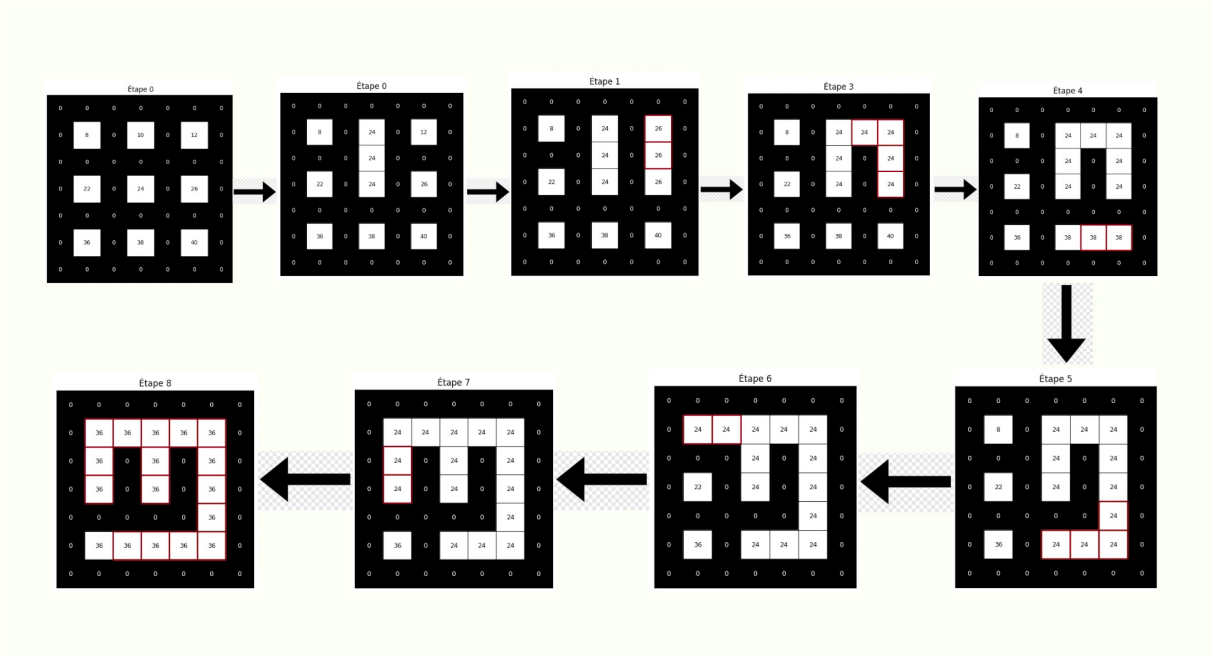


FIGURE 4 – Construction étape par étape d'un labyrinthe via la méthode de fusion.

Autre méthode : l'exploration exhaustive

Une autre méthode possible pour générer un labyrinthe est **l'exploration exhaustive**.

Dans cette méthode, on travaille uniquement avec les cases de coordonnées impaires en ligne et en colonne, représentent les cellules accessibles du labyrinthe. Toutes ces cellules sont initialement marquées comme non visitées.

On choisit une cellule de départ au hasard, qu'on marque comme visitée, puis on cherche aléatoirement une voisine non visitée.

Si une voisine est disponible, on transforme la case murale située entre les deux cellules en case de passage, puis on avance vers la nouvelle cellule, qu'on marque également comme visitée. Si aucune voisine n'est disponible, on revient en arrière (backtracking) jusqu'à trouver une cellule avec des voisines encore non visitées.

Ce processus se poursuit jusqu'à ce que toutes les cellules accessibles aient été visitées.

Comparaison des méthodes

Les deux méthodes présentées **la fusion aléatoire de chemins** et **l'exploration exhaustive** — permettent de générer des labyrinthes aléatoires valides, mais leurs résultats diffèrent sur plusieurs aspects.

La **fusion aléatoire de chemins** tend à produire des labyrinthes mieux équilibrés, avec davantage de bifurcations et de ramifications. Chaque mur a approximativement autant de chances d'être ouvert, ce qui permet une meilleure répartition des chemins. Cette propriété est particulièrement intéressante dans un jeu comme Pac-Man, car elle rend les déplacements plus variés et renforce l'intérêt stratégique.

À l'inverse, l'**exploration exhaustive** génère souvent des arbres de chemins déséquilibrés : les premiers chemins créés sont longs et dominants, tandis que ceux générés plus tard sont courts et peu développés. Le labyrinthe final est donc constitué de quelques axes principaux avec peu de ramifications secondaires, ce qui peut réduire l'intérêt du jeu.

Sur le plan technique, la **fusion aléatoire de chemins** nécessite une gestion plus rigoureuse (identifiants, ensembles, fusions), mais elle garantit une construction homogène et totalement

connexe. L'**exploration exhaustive**, plus simple à coder, est adaptée aux petites tailles, mais moins efficace pour produire des structures équilibrées sur des grandes grilles.

Pour ce projet, la méthode de**fusion aléatoire de chemins** a été retenue car elle permet de générer des labyrinthes logiques, uniformément répartis et parfaitement connectés, ce qui convient mieux à l'expérience de jeu souhaitée dans Pac-Man.

5 Tâches supplémentaires réalisées

Dans un souci d'amélioration de l'expérience utilisateur et de la robustesse du jeu, plusieurs fonctionnalités ont été ajoutées :

- **Écrans de fin de partie** : affichage d'une image de victoire (toutes les pacgommes mangées ou tous les fantômes éliminés) ou de défaite (Pac-Man attrapé).
- **Lecture robuste de la carte** : la fonction de lecture gère les erreurs de format (espaces superflus, lignes inégales) pour garantir une carte fidèle au fichier texte.
- **Relance automatique de la partie** : à la fin du jeu, une nouvelle partie peut être lancée sans redémarrer l'application, avec réinitialisation propre de la fenêtre SDL.

6 Défis rencontrés

Plusieurs défis techniques ont été relevés durant le développement :

- **Dynamisation du programme** : Passage d'un labyrinthe à taille fixe à un système adaptable aux dimensions données à l'exécution.
- **Correction des erreurs mémoire** : Utilisation de Valgrind pour identifier et corriger fuites et accès invalides liés aux allocations dynamiques.
- **Lisibilité et structure du code** : Refactorisation continue pour maintenir un code clair, modulaire et facilement extensible.

7 Conclusion

La mise en œuvre des tâches a abouti à la création d'une version de Pac-Man satisfaisante, intégrant les fonctionnalités de base du jeu. Chaque tâche a présenté ses propres défis, mais ceux-ci ont été surmontés grâce à une analyse méticuleuse et à un travail d'équipe efficace. En conclusion, le jeu offre une expérience fidèle à l'original, avec Pac-Man se déplaçant à travers la carte pour manger des pac-gommes tout en évitant les fantômes.

Références

[1] Wikipedia, *Algorithme de parcours en largeur*. `https://fr.m.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur`,

[2] Wikipedia, *Modélisation mathématique d'un labyrinthe*. `https://fr.wikipedia.org/wiki/Mod%C3%A9lisation_math%C3%A9matique_d%27un_labyrinthe`,