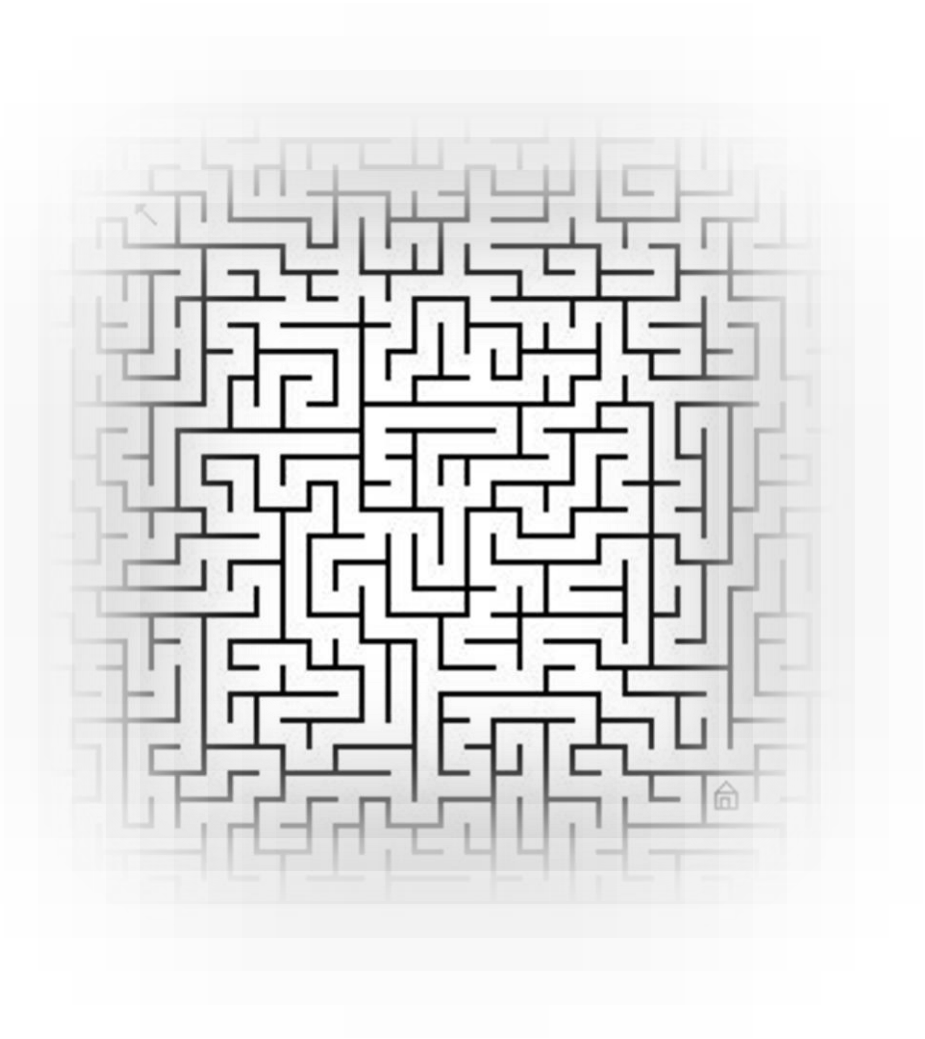
POLYTECH

Rapport du TP : Jeu du Labyrinthe



Joseph AKUETE, Léo BERNARDIN, 4A IMDS

Table des matières

1. Introduction

Dans le cadre d'un projet de Travaux Pratiques à réaliser en classe et le compléter à la maison, nous avons conçu et développé le jeu du labyrinthe en C++ en utilisant les principes de la programmation orientée objet. Ce projet visait à appliquer les notions apprises en cours, telles que l'héritage, la gestion des pointeurs, et l'utilisation des constructeurs/destructeurs. Le jeu, dans sa version de base, consiste à guider un personnage à travers un labyrinthe jusqu'à une sortie, tout en évitant les obstacles représentés par des murs.

Pour structurer ce rapport, nous commencerons par présenter les règles du jeu et les choix de conception qui nous ont guidé vers l'élaboration du code. Ensuite, nous détaillerons les aspects techniques de la structure du code, y compris les modifications apportées par rapport aux consignes initiales. Nous aborderons ensuite les principales difficultés rencontrées et les solutions mises en œuvre pour les surmonter. Enfin, nous conclurons en proposant des pistes d'amélioration pour enrichir ce projet et le rendre encore plus complet.

2. Analyse et Conception

Version de base

Dans notre version de base, le jeu consiste à guider un personnage représenté par le symbole *P,* à travers le labyrinthe jusqu'à atteindre la sortie *S*. Le joueur peut se déplacer dans le labyrinthe grâce aux couloirs en évitant les murs symbolisés par *o*.

Déplacement du joueur :

Le Personnage peut effectuer des mouvements dans toutes les directions grâce aux touches suivantes :

- d : déplacement vers la droite
- g : déplacement vers la gauche
- h : déplacement vers le haut
- b : déplacement vers le bas

Les déplacements sont permis uniquement si la case visée n'est pas un mur (o).

Règles des mouvements :

Si le joueur tente de déplacer le personnage vers une case contenant un mur, un message d'erreur s'affiche pour signaler que le mouvement est impossible.

Tous les autres mouvements sont autorisés, sans restriction sur le nombre de coups ou déplacements.

Version améliorée

Dans cette version, nous avons ajouté quelques fonctionnalités supplémentaires pour améliorer l'expérience de jeu :

Ajout d'un ennemi

L'ennemi est représenté par la lettre E dans le labyrinthe.

Déplacement de l'ennemi :

L'ennemi se déplace de manière aléatoire à chaque tour, immédiatement après le déplacement du joueur.

Il ne se déplace que sur les couloirs (ne pas pas atteindre la sortie).

Le joueur doit éviter de croiser le chemin de l'ennemi. Si l'ennemi atteint le personnage(P), la partie se termine, et l'ennemi remporte la victoire.

Condition de victoire :

Le joueur gagne s'il parvient à atteindre la sortie (S) sans être "attrapé" par l'ennemi.

Labyrinthes personnalisés

L'utilisateur peut créer et jouer avec son propre labyrinthe à partir d'un fichier texte.

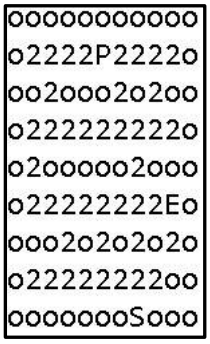
Nous vous avons remis trois fichiers : « labyrinthe_facile », « labyrinthe_difficile » et « labyrinthe_test » disponibles dans le fichier zip « labyrinthe_v2 » qui vous permettront de tester notre programme sans avoir à recréer vous-même un labyrinthe. Les deux premiers fichiers se différencient simplement par leur difficulté pour atteindre la sortie. Le troisième fichier texte est un labyrinthe qui ne respecte pas le bon format (cf consignes ci-dessous) que vous pouvez tester (bien mettre .txt à la fin du nom lorque vous entrez le nom du fichier dans le programme) pour vérifier le message d'erreur renvoyé par notre programme. Vous pouvez bien-sûr également tester le code avec votre propre labyrinthe s'il respecte les consignes inscrites ci-dessous :

Le fichier doit respecter une structure spécifique :

- o pour représenter les murs
- 2 pour représenter les couloirs ou le joueur et l'ennemi peuvent se déplacer.
- P pour le personnage
- E pour l'ennemi
- S pour la sortie

On utilise 2 pour les couloirs car l'espace n'était pas reconnu dans les fichiers.

Par convention que nous définissons, un labyrinthe doit contenir des murs sur tous ses bords, à part sur la case de la Sortie.



Exemple d'un labyrinthe conforme construit dans un fichier

Lorsque le jeu démarre, l'utilisateur a la possibilité de choisir entre deux options :

- Jouer avec le labyrinthe de base
- Importer un labyrinthe personnalisé à partir d'un fichier respectant le format défini.

Nous avons également donner un type à chaque objet graphique, pour pouvoir les différencier. Les différentes équivalences sont les suivantes :

- 1 : Mur
- 2 : Couloir
- 3 : Sortie
- 4 : Personnage
- 5 : Ennemi

3. Structures du code

Informations générales

Nous avons uniquement commenté le code de la version 2 (avec l'ennemi et l'intégration des fichiers) car la version 1 est incluse dans la version 2.

Nous avons appliqué le mot-clé const pour les méthodes qui ne modifient pas l'état interne de l'objet. Cela permet d'assurer une meilleure sécurité du code en indiquant que ces méthodes ne changent pas les attributs de la classe. Cette pratique a également facilité la compréhension du comportement attendu des méthodes et renforcé la robustesse de notre programme.

Nous avons utilisé le type Sortie dans les attributs de labyrinthe (différent du diagramme d'exemple montré en cours) car nous trouvions plus simple de manipuler la sortie comme un pointeur pour vérifier quand la partie s'arrête (quand les coordonnées de la sortie sont identiques à ceux du personnage).

Des destructeurs ont été implémentés uniquement pour les classes manipulant des pointeurs. Cela nous a permis de libérer correctement la mémoire allouée avec new et d'éviter tout risque de fuite de mémoire. Les classes sans pointeurs n'ont pas nécessité de destructeurs crées explicitement.

Modifications du code apportées par rapport au diagramme donné en cours

Nous avons ajouté la classe Ennemi et également ajouter des méthodes supplémentaires au code. Au début, nous avions ajouté la méthode EstDansPlateau() dans la classe Labyrinthe. Elle permettait de vérifier qu'un objet graphique ne sortait pas du plateau. On a ajouté cette méthode pour que les conditions de déplacements soient plus lisibles et simples d'utilisaiton dans notre code. Cependant, nous avons choisi d'ôter cette méthode et d'imposer la convention qu'un labyrinthe devait être composé de murs sur tous ses bords (excepté la case de la sortie). En effet, nous trouvons plus logique qu'un labyrinthe n'ait qu'une sortie possible.

Nous avons également ajouté une surcharge de l'opérateur « = » dans Labyrinthe. Cela nous a permis d'allouer un labyrinthe différent en fonction du choix de l'utilisateur (paramétré ou grâce à un fichier) dans le programme principal.

Puisque dans notre deuxième version nous permettons à l'utilisateur d'utiliser son propre labyrinthe à partir d'un fichier, nous avons également dû créer de nouvelles méthodes. Nous avons fait l'ajout de taillePlateauFichier() qui permet d'obtenir le nombre de lignes et de colonnes d'un labyrinthe crée dans un fichier. Cette méthode nous a été particulièrement utile pour créer un nouveau constructeur qui prend en argument un fichier dans Labyrinthe et qui appelle un constructeur de Plateau que nous avons également crée pour gérer les fichiers. Ce dernier prend en argument le nom du fichier, le nombre de lignes, colonnes, les coordonnées du personnage, de l'ennemi et et de la sortie.

Nous avons mis tous ces arguments dans le constructeur de Plateau afin que le constructeur de labyrinthe n'ait en argument que le nom du fichier et que la taille, la position du personnage, de l'ennemi et de la sortie en soient directement déduits, sans que l'utilisateur ait à entrer d'informations supplémentaires.

Exceptions

Comme mentionné précédemment : **Par convention que nous définissons, un labyrinthe doit contenir des murs sur tous ses bords, à part sur la case de la Sortie.**

Nous avons créé une classe ExceptionMouvement qui permet de retourner un message d'erreur à l'utilisateur si le déplacement du personnage qu'il souhaite faire est invalide. Les deux conditions qui peuvent entraîner une erreur sont :

- Un déplacement du personnage dans un mur.
- Un déplacement du personnage vers l'ennemi.

L'exception ExceptionMouvement n'arrête pas le programme et est simplement présente à but informatif pour le joueur.

De plus, nous avons créé une classe nommée ExceptionFormat qui renvoie un message d'erreur et qui arrête le programme si le format du labyrinthe représenté dans le fichier texte n'est pas correct. Dans le terme « format correct », nous entendons un labyrinthe qui ne contient que des murs sur ses bords à part une sortie (unique). Pour ce faire, nous avons ajouté la méthode VérificationPlateauFichier() (dans la classe Plateau) qui compte le nombre de murs sur les bords et qui vérifie qu'il est bien égal au nombre de cases sur les bords – 1, c'est-à-dire 2*nombre de lignes(pour avoir les bords gauches et

droits) + 2*nombre de colonnes (pour avoir les bords du haut et du bas) – 4 (pour enlever les cases comptées deux fois) – 1 (pour enlever la case de la sortie).

Enfin, nous avons crée une classe nommée ExceptionOuvertureFichier qui renvoie un message d'erreur et qui arrête le programme si le fichier a un problème d'ouverture (ou s'il n'existe pas).

4. Difficultés rencontrées

Constructeurs en cascade

Nous avons rencontré des difficultés pour gérer les constructeurs en cascade, notamment lorsqu'il fallait initialiser les objets dérivés dans une classe héritée. La logique derrière l'appel des constructeurs des classes parentes et la manière d'organiser leur initialisation n'étaient pas évidentes au départ. Nous avons surtout eu des problèmes concernant les liens des constructeurs entre les objets graphiques et les objets graphiques mobiles/fixes.

Compréhension des inclusions de fichiers .hpp

La gestion des inclusions de fichiers d'en-tête a été également complexe. Nous avons dû apprendre à éviter les inclusions circulaires, qui causent des erreurs de compilation, tout en respectant les relations d'héritage entre les classes. Nous avons travaillé sur papier pour bien noter les héritages entre les classes.

Création du labyrinthe à partir d'un fichier

Une autre difficulté que nous avons rencontrée était dans la lecture du fichier représentant le labyrinthe. Nous avons décidé d'obtenir les positions du personnage, de la sortie et de l'ennemi directement dans le fichier représentant le labyrinthe. En effet, cela permet une expérience plus ergonomique à l'utilisateur. C'était complexe car nous avons dû modifier la classe Plateau et Labyrinthe pour qu'elles communiquent ces informations entre elles. Nous avons fait face à cette difficulté en utilisant un passage par référence, ce qui nous a permis d'extraire les positions directement depuis le fichier et de les affecter dans les constructeurs nécessaires.

5. Pistes d'amélioration

Durant l'élaboration de notre code, nous nous sommes créé une liste d'idées d'améliorations à effectuer pour notre programme. Parmi les pistes d'amélioration, figurait l'ajout d'un ennemi, que nous avons pu implémenter. Par manque de temps, la grande majorité des idées que nous avions n'ont malheureusement pas pu être ajoutées. Voici la liste des idées auxquelles nous avions pensé :

- Ajout d'un mode de jeu avec deux personnages qui doivent chercher la sortie en premier et qui joue chacun à leur tour.
- Ajout d'une case « Trésor » de type ObjetGraphiqueFixe qui enlève un mur (le transforme en couloir) aléatoirement sur le plateau si le personnage va dessus. Le mur sélectionné au hasard est parmi les murs qui ne sont pas sur les bords du labyrinthe car cela n'aiderait pas le personnage à trouver la sortie. Cet ajout serait limité au mode Solo sans Ennemi car le but de la case est de profiter à un seul joueur. On pourrait penser à une case « Trésor » où le joueur peut choisir le mur qu'il enlève dans un mode Joueur vs Joueur ou Joueur vs Ennemi également.
- Ajout d'une case « Paralysie » de type ObjetGraphiqueFixe qui empêche l'ennemi ou le joueur adverse de se déplacer pendant un tour.
- Amélioration de l'ennemi qui se déplace de manière intelligente
- Ajout d'une interface Qt pour pouvoir jouer avec les flèches du clavier.
- Ajout d'un mode « Hardcore » qui lance un chronomètre qui indique le temps restant avant que la partie soit perdu, avec un timer choisi par l'utilisateur.
- Ajout d'une aide pour expliquer le principe de chacun des différents modes de jeu.
- Génération d'un labyrinthe de façon aléatoire.
- Ajout d'identifiants de connexion.
- Reprendre une partie sauvegardée et donner un nom à sa sauvegarde grâce à des fichiers qui enregistrent l'état de la partie.
- Intégrer un score max (meilleur temps en mode « Hardcore ») par identifiant.

6. Conclusion

Ce projet de jeu de labyrinthe a été une expérience enrichissante qui nous a permis de mettre en pratique les concepts de programmation orientée objet tout en développant nos compétences en conception et en résolution de problèmes. À travers la création de fonctionnalités telles que le déplacement du personnage, l'ajout d'un ennemi et la personnalisation des labyrinthes, nous avons pu intégrer des notions complexes comme l'héritage, la gestion de mémoire dynamique et la manipulation de fichiers.

Bien que nous ayons rencontré des défis, notamment dans la gestion des constructeurs en cascade, l'organisation des inclusions et la lecture des fichiers, ces difficultés nous ont permis d'apprendre à structurer un projet complexe. Les solutions mises en œuvre, telles que l'utilisation de passages par référence et l'ajout de méthodes spécifiques, ont renforcé la robustesse et la flexibilité de notre code.

Nous avons également réfléchi à plusieurs pistes d'amélioration qui pourraient enrichir l'expérience de jeu, comme l'ajout de nouveaux modes de jeu, une interface graphique, ou encore des fonctionnalités de sauvegarde et de score. Ces idées montrent le potentiel évolutif de ce projet et les nombreuses opportunités qu'il offre pour approfondir nos compétences techniques.

En conclusion, ce projet a été non seulement un défi académique, mais aussi une source de satisfaction personnelle, car il nous a permis de concevoir un jeu fonctionnel tout en consolidant nos bases en programmation orientée objet.