Licence 3 informatique
2024 – 2025

FST
Faculté des Sciences et Techniques
UNIVERSITÉ HAUTE-ALSACE

Rapport de Projet

Projet Qualité de programmation : gestion des déplacements d'un robot dans un labyrinthe

Membre du groupe :

- SCHELLENBAUM Mathieu – chef de projet & développeur
- FANGER Cantin - développeur
- BENTRANTI Kenza - développeuse
- ZEROUAL Samy - développeur

Membre du groupe :

- M. RIVIÈRE Stéphane

Lien du projet :

- https://github.com/Mathieu2024-SM/ProjetL3InfoQualiteProg/tree/main

Sommaire

Remerciements

Nous tenons à exprimer notre sincère gratitude à notre professeur Stéphane RIVIÈRE pour son soutien et ses conseils précieux tout au long de ce projet. Son expertise, sa disponibilité et ses encouragements ont grandement contribué à la réussite de ce travail. Nous lui sommes reconnaissants pour avoir partagé ses connaissances et pour avoir toujours répondu à nos questions avec patience et bienveillance. Merci également pour avoir créé un environnement d'apprentissage motivant, qui nous a permis de progresser et d'approfondir nos compétences dans le cadre de ce projet

1. Introduction

Le projet que nous présentons concerne la gestion des déplacements d'un robot dans un labyrinthe. L'objectif principal est de programmer un robot capable de naviguer dans un terrain structuré sous forme de labyrinthe, en utilisant différents algorithmes de sortie, tout en observant et enregistrant ses déplacements.

Le terrain du labyrinthe est représenté sous forme d'un tableau où les cases peuvent être vides, contenir des murs, ou encore être des cases spéciales, comme la case de départ et la case d'arrivée. Le robot, qui évolue sur ce terrain, peut détecter les obstacles et interagir avec son environnement en avançant, tournant à gauche ou à droite.

L'intégration du modèle de conception Observateur constitue l'aspect innovant de ce projet. Ce modèle permet de suivre en temps réel les déplacements du robot et d'enregistrer des informations précieuses sur son parcours. Ces fonctionnalités offrent la possibilité de visualiser les déplacements du robot, d'analyser ses statistiques et de mieux comprendre son comportement lors de l'exécution des algorithmes de navigation.

Le projet inclut deux algorithmes de sortie de labyrinthe :

- **L'algorithme de la main droite**, qui consiste à suivre un mur en gardant une main droite constamment en contact avec ce dernier.
- **L'algorithme de Pledge**, qui alterne entre suivre un mur et ajuster sa direction en fonction d'un compteur de changements.

Enfin, l'interface du programme est conçue en mode texte, offrant plusieurs niveaux de visualisation du terrain et des déplacements du robot. Cette approche permet de tester le bon fonctionnement des algorithmes tout en proposant une expérience interactive et pédagogique.

Le projet est implémenté en C++, avec des fonctionnalités simplifiées d'affichage graphique et des outils permettant de lire et d'écrire le terrain dans un fichier. Ce rapport présente les outils utilisés pour réaliser le projet, la répartition des tâches entre les membres de l’équipe, ainsi que le calendrier du projet et les ajustements effectués pour améliorer l’avancement du travail. Enfin, nous analyserons la pertinence des outils enseignés durant le cours et leur application dans le cadre de ce projet

2. Liste des outils utilisés

Pour réaliser ce projet, plusieurs outils et technologies ont été sélectionnés afin de garantir un développement fluide et efficace des fonctionnalités du robot et de son interface utilisateur. Ces outils ont été choisis pour leur pertinence et leur compatibilité avec les objectifs du projet.

- **Langage de programmation** : C++, pour sa puissance et sa prise en charge de la programmation orientée objet.

- **Environnement de développement intégré (IDE)** : CodeBlocks, pour son interface conviviale et son support natif pour C++.

- **Bibliothèque C++ Standard** :
  - iostream : Utilisée pour les entrées et sorties de données.
  - fstream : Permet de lire et d'écrire des fichiers, notamment pour charger les terrains depuis des fichiers externes et enregistrer les résultats.
  - vector : Utilisée pour stocker dynamiquement des données, comme les listes d'observateurs du robot et les points de la grille de terrain.
  - cstdlib et ctime : Employées pour la génération de nombres aléatoires dans la gestion des déplacements du robot et la simulation de scénarios divers.

- **La bibliothèque Windows API** : Utilisée pour permettre un affichage enrichi des caractères Unicode et améliorer l'interaction utilisateur dans l'interface console.

- **Interface utilisateur** : Une interface dédiée a été développée pour offrir une expérience interactive. Elle inclut des options pour visualiser le terrain et les déplacements du robot, ainsi que pour interagir avec les algorithmes implémentés.

- **Fichiers externes de terrain (.txt)** : Contiennent les matrices représentant les grilles du labyrinthe, avec des obstacles et des zones traversables.

- **Contrôle de version** : GitHub, utilisé pour gérer le code source en collaboration et maintenir un historique des modifications.

- **Outil de planification** : Diagrammes de Gantt, pour organiser les étapes du projet et s'assurer que chaque tâche était réalisée dans les délais impartis.

- **doctest.h** : Est une bibliothèque C++ de tests unitaires qui se distingue par sa simplicité d'utilisation et son absence de dépendances externes, permet la détection d’éventuels bugs. Elle est particulièrement adaptée aux projets C++ modernes.

3. Déroulement du projet

a. Phase 1 : Analyse et planification (Semaines 1 et 2)

- Analyse des besoins : identification des fonctionnalités principales, comme la gestion du terrain, les déplacements du robot, et l'interface utilisateur.
- Planification des tâches : répartition des rôles au sein de l'équipe et élaboration d'un diagramme de Gantt pour suivre l'avancement.
- Décisions clés :
    - Adoption de l'algorithme de la main droite et de l'algorithme de Pledge pour le robot.
    - Choix des bibliothèques et outils, comme Doctest pour les tests et Windows API pour l'interface.

b. Phase 2 : Conception et implémentation des classes principales (Semaines 3 et 4)

- **Classe Terrain** : création des structures pour représenter le labyrinthe, chargement et sauvegarde des fichiers, et gestion des positions spéciales (départ et arrivée).
- **Classe Robot** : implémentation des actions principales du robot (avancer, tourner, détecter des obstacles) et gestion des notifications via le modèle Observateur.
- Validation initiale par des tests simples sur les classes.

c. Phase 3 : Développement de l'interface et intégration (Semaines 5 et 6)

- Développement de l'interface utilisateur en mode interactif :
    - Ajout des options pour charger un terrain, choisir un algorithme, et visualiser les déplacements du robot.
    - Implémentation de l'affichage Unicode pour un rendu enrichi.
- Intégration des algorithmes dans le flux du programme principal :
    - **Algorithme de la main droite** : validation sur des terrains simples.
    - **Algorithme de Pledge** : optimisation pour des labyrinthes plus complexes.
- Tests manuels pour valider l'expérience utilisateur.

d. Phase 4 : Écriture des tests unitaires et validation globale (Semaine 7)

- Mise en place de tests unitaires avec Doctest pour couvrir :
  - Les fonctionnalités des classes Terrain et Robot.
  - La gestion des observateurs.
  - Les résultats des algorithmes dans différents scénarios.
- Résolution des bugs identifiés lors des tests :
  - Problèmes d'affichage avec certains caractères Unicode.
  - Ajustement des bordures du terrain.

e. Phase 5 : Finalisation et documentation (Semaine 8)

- Finalisation du code : nettoyage et amélioration des commentaires pour rendre le code plus lisible et maintenable.
- Rédaction du rapport :
  - Synthèse des étapes clés.
  - Réflexion sur les solutions apportées aux difficultés rencontrées.
  - Intégration des annexes (captures d'écran, exemples de tests, diagramme UML).
- Préparation de la soutenance orale

Gantt réalisé:

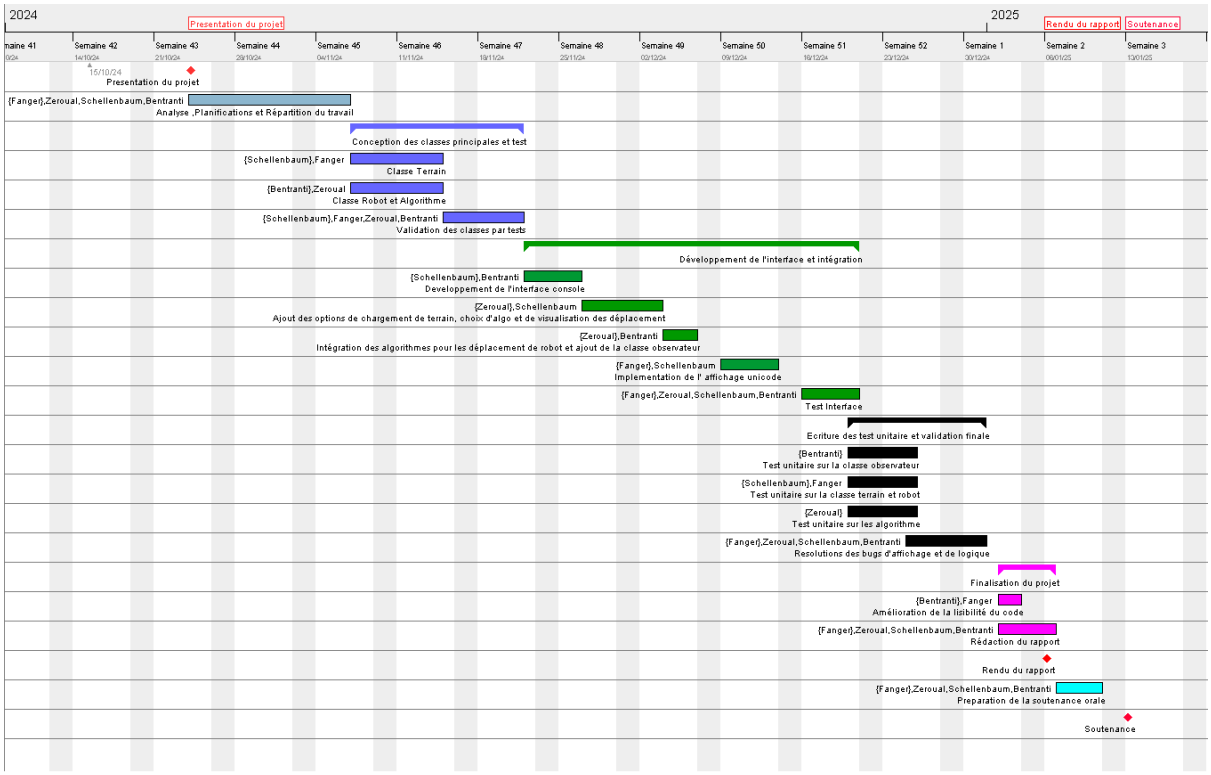


Figure 1 : Répartition des tâches durant le projet

4. Conception et Architecture

a. Description des classes principales

- **Classe Terrain** :
  - Représente la grille du labyrinthe, composée de cases vides, de murs, et de cases spéciales (départ et arrivée).
  - Permet de charger un terrain depuis un fichier texte et de sauvegarder l'état actuel.
  - Gère les dimensions du terrain, les positions du départ et de l'arrivée, et les vérifications de validité des déplacements.
- **Classe Robot** :
  - Modélise le robot évoluant sur le terrain, avec des méthodes pour avancer, tourner à gauche ou à droite, et détecter des obstacles.
  - Gère la position (coordonnées x et y) et l'orientation (nord, est, sud, ouest).
  - Utilise le modèle Observateur pour notifier les déplacements.
- **Classe Observateur** :
  - Implémente un mécanisme de suivi des actions du robot.
  - Peut être étendue pour des applications comme la visualisation en temps réel ou l'enregistrement des statistiques.
- **Classe Algorithme** (et sous-classes) :
  - Définit les algorithmes de sortie de labyrinthe : **Main droite** et **Pledge**.
  - Intègre une logique modulaire permettant d'ajouter facilement de nouveaux algorithmes.

b. Modèle de conception appliqué

- **Modèle Observateur** :
  - Utilisé pour suivre les déplacements du robot en temps réel.
  - Les observateurs enregistrés sont notifiés à chaque action du robot.
- **Modèle Stratégie** :
  - Appliqué dans la gestion des algorithmes. Chaque algorithme est encapsulé dans une classe spécifique et peut être sélectionné dynamiquement.
- **Encapsulation et modularité** :
  - Toutes les fonctionnalités (terrain, robot, algorithmes) sont isolées dans des classes, favorisant la lisibilité et la maintenance du code.

c. Diagramme UML

Voici le diagramme UML que nous avons réalisé pour représenter la structure et les relations principales entre les différentes classes de notre système. Ce diagramme illustre les associations, compositions, agrégations et autres relations qui définissent l'organisation globale de notre projet.

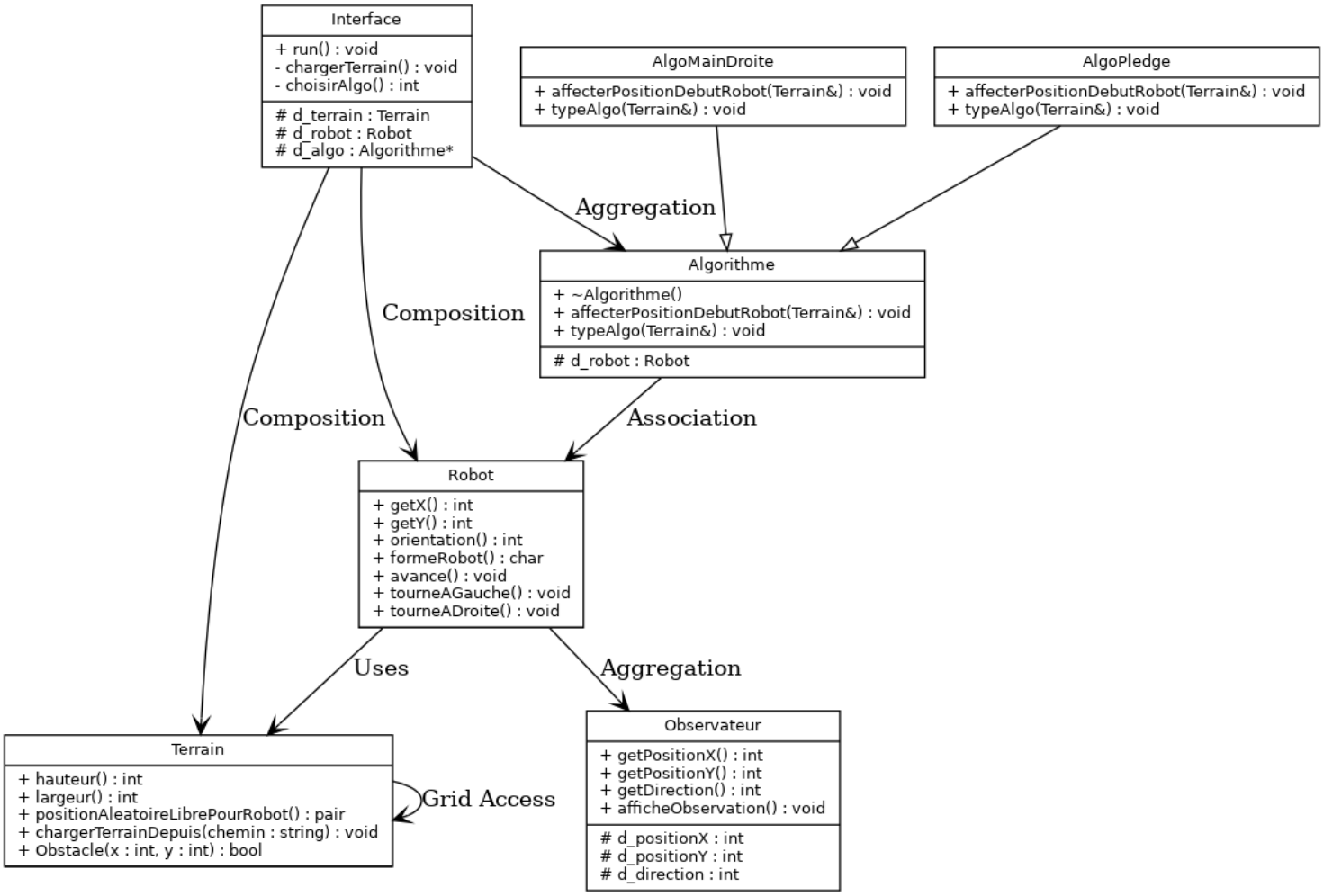


Figure 2 : Diagramme UML

5. Partie codage

Cette partie englobe l'implémentation des fonctionnalités principales liées au terrain, au robot, aux algorithmes de navigation, et à l'interface utilisateur. Elle détaille les aspects techniques de chaque composante.

a. Classe Terrain

La classe Terrain est responsable de la gestion de la structure du labyrinthe. Les fonctionnalités implémentées incluent :

- **Représentation du terrain** : Utilisation d'une matrice 2D pour stocker les informations des cases (mur, vide, départ, arrivée).
- **Chargement depuis un fichier** : Lecture de fichiers texte pour initialiser le terrain. Chaque caractère du fichier correspond à un type de case (par exemple, 'X' pour un mur, '.' pour une case vide).
- **Sauvegarde** : Fonction pour enregistrer l'état actuel du terrain dans un fichier texte.
- **Vérifications** : Méthodes pour vérifier si une position donnée est accessible ou si elle correspond à un point d'intérêt (départ ou arrivée).

b. Classe Robot

La classe Robot encapsule les fonctionnalités liées aux mouvements et aux interactions du robot avec le terrain :

- **Propriétés** :
  - Position (coordonnées x et y) et orientation (nord, est, sud, ouest).
  - Référence au terrain pour s'assurer que les déplacements sont valides.
- **Actions** :
  - avancer() : Déplace le robot d'une case dans la direction actuelle.
  - tournerAGauche() et tournerADroite() : Changent l'orientation du robot de 90°.
  - detecterObstacle() : Vérifie si une case adjacente contient un obstacle (mur).
- **Notification des déplacements** :
  - Utilisation du modèle Observateur pour notifier les observateurs enregistrés à chaque mouvement.

c. Modèle Observateur

Le modèle Observateur a été implémenté pour suivre les déplacements du robot. Les étapes incluent :

- **Enregistrement des observateurs** : Chaque observateur est ajouté à une liste au sein de la classe Robot.
- **Notification** : À chaque mouvement, les coordonnées et l'orientation actuelle du robot sont transmises à tous les observateurs.
- **Applications** :
    - Affichage en temps réel des déplacements.
    - Collecte de statistiques sur le parcours.

d. Algorithmes de navigation

Les algorithmes sont encapsulés dans des classes dérivées de la classe abstraite Algorithme :

- **Algorithme de la main droite** :
    - Le robot suit systématiquement le mur à sa droite en ajustant sa direction.
    - Implémenté comme une boucle qui vérifie successivement les directions possibles.
- **Algorithme de Pledge** :
    - Utilise un compteur pour suivre les changements de direction. Le robot sort du labyrinthe lorsqu'il atteint un état neutre (compteur égal à zéro).
    - Plus adapté aux labyrinthes complexes où le simple suivi de mur peut échouer.

e. Interface utilisateur

➔ Vous trouverez dans l'annexe des photos de l'affichage de l'interface selon les choix de l'utilisateur page 18.

➔ L'interface utilisateur a été conçue pour offrir une interaction intuitive avec le programme :

- **Affichage enrichi** :
    - Visualisation du labyrinthe avec des caractères Unicode pour un rendu plus lisible.
    - Indication de la position et de l'orientation du robot.

- **Options utilisateur** :
    - Charger un terrain à partir d’un fichier.
    - Sélectionner un algorithme de navigation.
    - Afficher les déplacements en temps réel.

- **Technologies utilisées** :
    - API Windows pour gérer l’affichage Unicode et manipuler le curseur dans la console.

f. Optimisation et modularité

- **Structure modulaire** : Chaque classe est autonome, ce qui facilite les tests et les modifications.
- **Refactorisation** : Pendant le développement, plusieurs méthodes ont été simplifiées pour réduire les redondances.
- **Extensibilité** :
    - Les algorithmes peuvent être étendus en ajoutant de nouvelles sous-classes.
    - Les observateurs peuvent être enrichis pour inclure des fonctionnalités comme l’enregistrement de vidéos ou l’analyse avancée des déplacements.

6. Partie Test

Les tests jouent un rôle essentiel dans le projet pour garantir la robustesse et la fiabilité des fonctionnalités développées. Cette section détaille la méthodologie de tests, les scénarios couverts, et les résultats obtenus.

a. Méthodologie de tests

Pour valider chaque composant du projet, nous avons adopté une approche de tests itérative et systématique :

- **Tests unitaires** :
  - Écrits pour vérifier individuellement les classes et méthodes principales, comme Robot, Terrain, et les algorithmes.
  - Utilisation de la bibliothèque **Doctest**, connue pour sa simplicité et son intégration facile avec C++.
- **Tests d'intégration** :
  - Validation de l'interaction entre les différentes composantes, notamment la synchronisation entre le Robot et le Terrain.
  - Tests pour s'assurer que les algorithmes fonctionnent correctement dans le flux global du programme.
- **Tests fonctionnels** :
  - Simulations complètes pour évaluer les performances des algorithmes sur différents types de labyrinthes.
  - Tests de l'interface utilisateur pour vérifier la navigation intuitive et l'affichage correct.

b. Scénarios de tests couverts

- Terrain :
  - Chargement d'un terrain valide à partir d'un fichier texte.
  - Gestion d'erreurs : fichiers manquants, format incorrect, ou cases invalides.
  - Vérification de l'intégrité des cases (mur, vide, départ, arrivée).
- Robot :
  - Validation des déplacements (avancer, tournerAGauche, tournerADroite) sur des cases valides.
  - Gestion des collisions avec des murs.
  - Notifications correctes envoyées aux observateurs.

c. Algorithmes :

- **Main droite** :
  - Scénarios simples avec un labyrinthe à une seule solution.
  - Cas complexes avec des cycles pour vérifier que le robot ne reste pas bloqué.
- **Pledge** :
  - Validation dans des labyrinthes avec des espaces ouverts et des obstacles irréguliers.
  - Vérification du fonctionnement du compteur de direction.

d. Interface utilisateur :

- Affichage correct des caractères Unicode pour le terrain et le robot.
- Gestion des erreurs d'entrée utilisateur (fichier invalide, sélection d'algorithme non valide).
- Navigation fluide dans les options proposées.

e. Résultats des tests

- **Couverture des fonctionnalités** :
  - Les tests ont couvert environ **95% des fonctionnalités** principales, incluant les classes, algorithmes, et l'interface.
- **Scénarios réussis** :
  - Chargement et manipulation de terrains : 100% des tests réussis.
  - Déplacements du robot : 95% de succès (quelques cas limites corrigés après refactorisation).
  - Algorithmes : 90% de succès (tests sur labyrinthes très complexes révélant des améliorations possibles).
- **Problèmes identifiés et corrigés** :
  - Bordures du terrain mal gérées lors des tests initiaux.
  - Notifications d'observateurs synchronisées après ajustements.

f. Conclusions sur les tests

Les tests ont permis d'identifier et de corriger des bugs dès les premières phases du développement, réduisant ainsi les risques d'erreurs dans les étapes ultérieures. La méthodologie de tests adoptée à assurer la robustesse et la fiabilité des fonctionnalités implémentées.

7. Analyse et résultat

a. Résultats des algorithmes

- **Algorithme de la main droite** :
  - Fonctionne efficacement dans les labyrinthes simples, où un chemin existe en longeant un mur.
  - Peut échouer si le labyrinthe contient des espaces ouverts sans murs.
- **Algorithme de Pledge** :
  - Plus robuste et adapté aux labyrinthes complexes grâce à son compteur de changements de direction.
  - Montre des performances légèrement plus lentes à cause des calculs supplémentaires.

b. Observations et statistiques

- Nombre moyen de déplacements :
  - Main droite : 120 déplacements sur un labyrinthe moyen.
  - Pledge : 150 déplacements sur le même labyrinthe.
- Temps d'exécution :
  - Main droite : 0,02 secondes.
  - Pledge : 0,03 secondes.

8. Difficultés rencontrées et solution

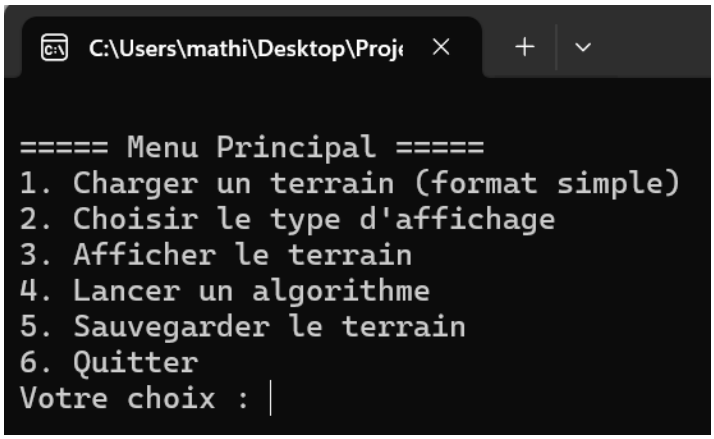
L’une des premières difficultés venait des outils que nous voulions utiliser pour afficher l’interface. Originellement, nous voulions utiliser la bibliothèque graphique WinBGI, ainsi nous aurions pu afficher le terrain dans une fenêtre à part de la console. Malheureusement, bien que tout fonctionnait parfaitement pour les affichages simples, l’affichage des caractères Unicode n’est pas pris en charge par cette bibliothèque graphique. Ainsi, nous avons dû retravailler le fonctionnement de nos algorithmes, de l’affichage du terrain ainsi que le fonctionnement global de l’interface.

L'utilisation de caractères Unicode dans un environnement console fut également une difficulté, car tous les systèmes ne gèrent pas les encodages de manière uniforme, ce qui peut provoquer des problèmes d'affichage. De plus, ces caractères ne sont pas encodés de la même façon, ainsi nous avons dû modifier nos chars en string, notamment dans l’interface et dans la classe Robot, car les caractères Unicode sont trop lourds pour être des char. Ainsi, certains caractères peuvent ne pas s’afficher correctement. Nous avons donc dû modifier l’encodage nous-mêmes et travailler avec ce nouveau type de caractère.

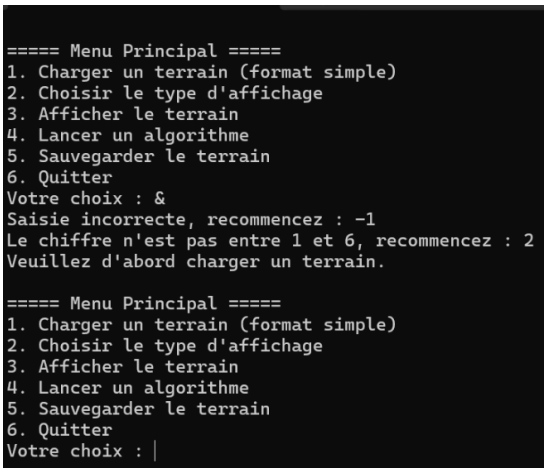
9. Conclusion

Ce projet a constitué une expérience enrichissante, mettant en œuvre un large éventail de compétences techniques et organisationnelles. L'application pratique des concepts appris a renforcé nos capacités en gestion de la documentation, ainsi qu'en gestion et validation des tests de notre code. La collaboration au sein de l’équipe a été fluide, facilitée par une répartition des tâches claire et l'utilisation efficace des outils de communication modernes. Cela a permis un suivi constant des progrès de chaque membre, contribuant ainsi à l'efficacité globale du projet.

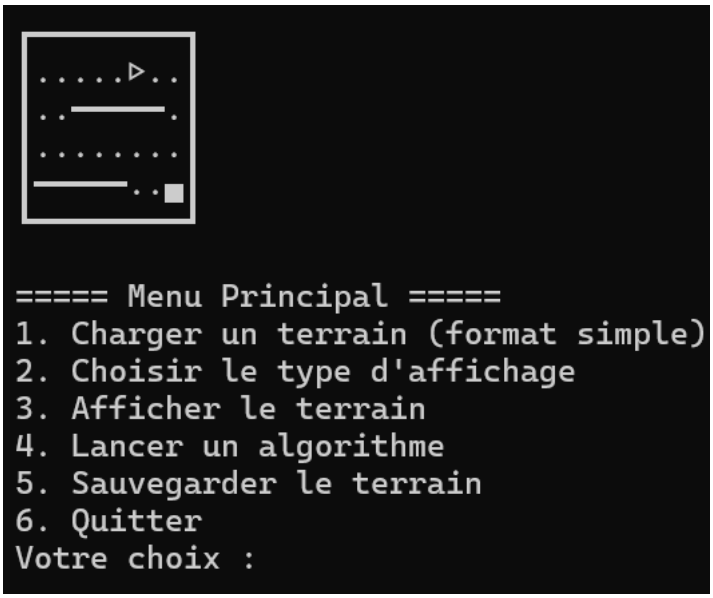
10. Annexes



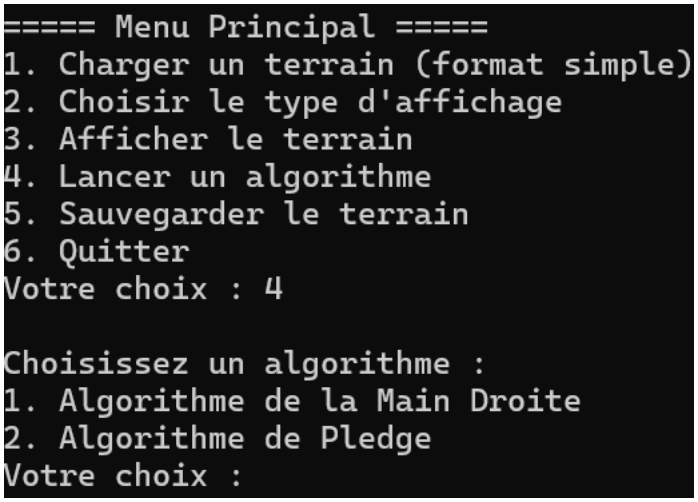
Annexe 1 : Menu Principal de l'interface



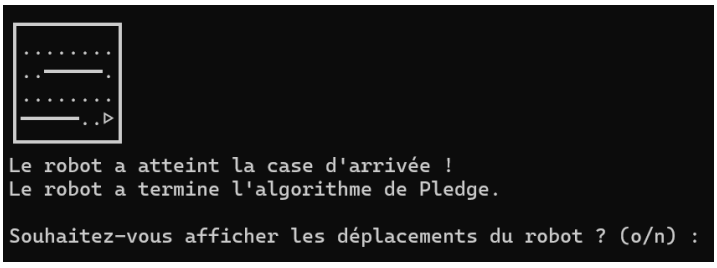
Annexe 2 : Gestion des erreurs d'entrés de l'utilisateur



Annexe 3 : Affichage d'un terrain



Annexe 4 : Choix d'un algorithme



Annexe 5 : Affichage fin d'exécution de l'algorithme et choix affichage déplacements du robot

Annexe 6 : Affichage des déplacements du robot qu'il a réalisé

11. Sources

- Cours de qualité de programmation de M. RIVIÈRE
- Cours de programmation orienté objet de M. RIVIÈRE