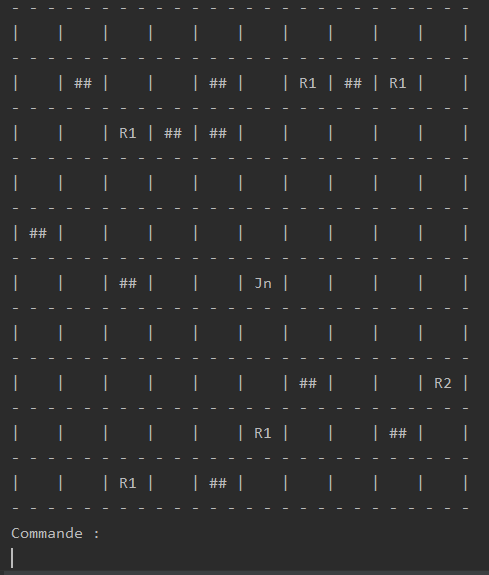


Rapport Qualité de programmation

Sujet : Robot Ecraseur



Groupe 9 : Yassine Goeller, Léandre Demuth, Victor Sauter, Victor Brungard

Sommaire

[I. Introduction 3](#_Toc29208177)

[II. Cahier des charges : 4](#_Toc29208178)

[III. Méthodes de travail 5](#_Toc29208179)

[a) Git 5](#_Toc29208180)

[b) Communication 5](#_Toc29208181)

[c) Répartition du travail 5](#_Toc29208182)

[IV. Réalisation 6](#_Toc29208183)

[a) UML 6](#_Toc29208184)

[b) Implémentation du jeu 7](#_Toc29208185)

[V. Conclusion 10](#_Toc29208186)

[a) Bilan humain 10](#_Toc29208187)

[b) Bilan pédagogique 10](#_Toc29208188)

[c) Le bilan technique 10](#_Toc29208189)

# Introduction

Ce projet a été réalisé dans la cadre de la matière qualité de programmation. Nous allons donc répondre aux attentes du sujet à l’aide de toutes nos connaissances.

Le sujet consiste à programmer un jeu de robot écraseur grâce au language C++. Le jeu se compose d’un joueur (expert ou normal) et de plusieurs robots (1ère ou 2e génération) sur un terrain, et le principe consiste à survivre le plus longtemps ou à détruire tous les robots en les faisant foncer les uns dans les autres ou dans des débris (disposés sur le terrain ou généré par la collision de deux robots). La partie est gagné si le joueur est en vie et que tous les robots est détruit. Et la partie est perdu si le joueur rentre en collision avec un robot ou un débris.

Le travail est à réaliser en groupe. Celui-ci est composé de Yassine Goeller, Léandre Demuth, Victor Sauter et Victor Brungard. Nous nous sommes réunis par affinité mais aussi car nous avons l’opportunité de pouvoir se voir facilement.

Il est donc indispensable de s’organiser. Pour cela, il a fallu une bonne communication et une bonne méthode de travail pour mener a bien ce projet.

# Cahier des charges

Le langage utilisé sera du C++.

Les fonctionnalités attendues sont :

* Le joueur peut se déplacer dans l’une des huit cases autour de lui
* Un robot cherche à écraser le joueur
* Deux types de robots : des robots première génération qui ne se déplacent qu’horizontalement et verticalement, et des robots deuxième génération qui se déplacent aussi en diagonales
* Les robots ne sont capables de détecter qu’un être humain
* Quand un robot se déplace sur la case du joueur, il écrase le joueur et celui-ci meurt
* Quand un robot se déplace sur la case d’un autre robot, les deux robots dont détruis et la case contient alors un débris
* Quand un robot se déplace sur la case d’un débris, il est aussi détruit
* La partie se termine soit quand tous les robots sont détruits, soit quand le joueur est écrasé
* Créer un terrain
* Sauvegarder un terrain
* Charger la configuration d’un terrain
* Organiser une partie de jeu avec ce terrain avec un affichage en mode texte
* Affichage graphique (wingbi)
* Ajouter d’autre types de robots
* Ajouter des débris dans le terrain
* Avoir un joueur expert qui ne se déplace qu’horizontalement et verticalement

# Méthodes de travail

## Git

Lors de la création du projet nous avons introduit directement le logiciel de version Git.

Ce logiciel permet de faciliter le partage des ﬁchiers et permettre à tout le monde d’avoir en permanence la dernière version du projet.

## Communication

La messagerie instantané Messenger fut l’outil le plus utilisé dans ce projet. Elle nous a permis de communiquer et surtout d’avoir un historique des messages pour les personnes n’étant pas disponible à ce moment-là.

Des réunions hebdomadaires ont été mises en place. Nous parlions des problèmes rencontrés, de l’avancement des tâches qui avaient été distribués et des prochaines tâches pour la semaine à venir. Elles permettent donc un suivi du projet, et surtout de faire avancer le projet en résolvant le problème ensemble.

## Répartition du travail

Les premières réflexions du projet ont été faite ensemble, ainsi que le diagramme UML.

Pour la suite tout s’est fait de manière égale.

Victor Sauter s’est occupé d’une partie de la classe terrain et game, ainsi que de la classe robot, robot première génération et deuxième génération, et la mise en page du rapport.

Léandre Demuth a réalisé la classe persistance, entiteMouvante et une partie de la classe game.

Yassine Goeller quant à lui a fait la classe joueur, joueurExpert, joueurNormal et les tests.

Victor Brungard a fait la classe position, une partie de la classe game et terrain, ainsi que le rapport et manuel utilisateur.

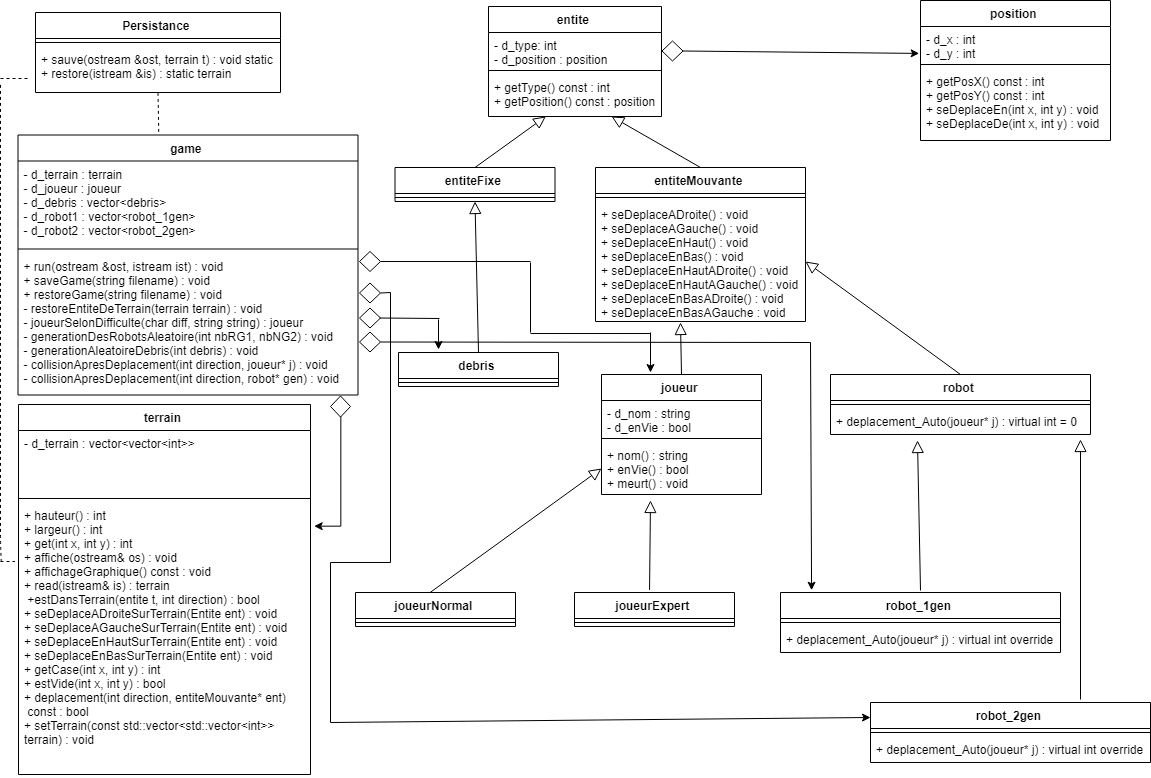
Sachant que nous nous sommes aidés tout au long du projet sur les différentes classes.

# Réalisation

## UML

Avant de se lancer tête baissée nous avons réfléchis aux différentes classes que nous auront besoin pour réaliser le programme.

Nous avons réalisé un UML :



## Implémentation du jeu

La classe persistance va permettre de sauvegarder un terrain ainsi que de le restaurer.

La classe position va nous permettre de sauvegarder les coordonner x et y et avec des méthodes mettre déplacer ces positions à partir d’un vecteur ou de déplacer la position à une autre position.

La classe robot première génération va permettre de créer un robot avec une position x et y et avec une méthode on va pouvoir déplacer le robot sur le terrain et une autre méthode on pourra trouver le meilleur déplacement à faire pour que le robot suive le joueur.

La classe robot va permettre de créer une entité mouvante avec une position x et y.

La classe terrain va créer un terrain avec une largeur et une hauteur et on pourra avec une méthode afficher le terrain avec une autre méthode regarder si les entités sont dans le terrain enfin on pourra déplacer les entités mouvantes dans le terrain avec tous les méthode déplacement.

La classe entite représente la classe mère de nombreuses sous-classes. Elle a une position noté d\_pos et a un entier noté d\_type.

La classe entite possède :

* Deux constructeurs, un qui prend en paramètre x et y qui sont deux entiers permettant de mettre à jour la position de l’entité. L’autre qui prend un paramètre une position.
* Un destructeur
* Une méthode getPosition() qui renvoie la position de l’entite
* Une méthode getType() qui renvoie un entier c’est-à-dire le type de l’entite

La classe entiteFixe est une classe dérivée de la classe entite. Elle permet de structurer les différentes classes qui vont suivent. L’entite ne pourra donc pas bouger. C’est pour cela que la classe débris dérive de cette classe.

Ensuite, il existe la classe entiteMouvante. A l’inverse d’une entiteFixe, cette classe possède des méthodes permettant de se déplacer. Les 4 méthodes sont seDeplaceADroite(), seDeplaceEnHaut(), seDeplaceAGauche(), seDeplaceEnBas().

Un robot est une entiteMouvante, tout comme un joueur. La classe joueur a un nom de type string et une méthode permettant de connaître son nom, getNom().

2 classes vont être hérités de la classe joueur, joueurExpert et joueurNormal. La classe joueurNormal va créer un joueur avec nom, un position x et y. Ce joueur peut se déplacer sur n’importe qu’elle case autour de lui. La classe joueurExpert est comme joueurNormal mais ne peut se déplacer que horizontalement ou verticalement.

La classe game correspond au déroulement d’une partie. Elle a un terrain noté d\_terrain, un joueur d\_joueur, une liste de débris d\_debris, une liste de robots première génération d\_robot1, une liste de robots deuxième génération robot2.

Elle possède les méthodes suivantes :

* game() c’est-à-dire le constructeur. Il crée un terrain et l’initialise
* run(std ::ostream& ost, std ::istream& ist) qui démarre une partie. Crée le joueur, les robots et les placent sur le terrain. Puis demande à l’utilisateur de jouer pour faire défiler les tours de jeux jusqu’à la fin de la partie
* saveGame(std ::string fileName) qui permet de sauvegarder la partie dans un fichier donné en paramètre
* restoreGame(std ::string fileName) qui permet de restaurer une partie à partir d’un fichier donné en paramètre

## Les fonctionnalités réalisées

Voici la liste des fonctionnalités du jeu :

* Sauvegarder une partie
* Reprendre une partie précédemment enregistrer
* Charger un terrain avec un fichier
* Fin de partie (gagner ou perdu) avec score
* Affichage graphique et texte
* Déplacement automatique des robots selon leur génération
* Interface par ligne de commandes pour contrôler le jeu

# Conclusion

## Bilan humain

Sur le plan humain, ce projet nous a permis de mieux communiquer avec les membres de notre équipe. En effet, il nous arrivait de rencontrer des problèmes aussi bien que des bugs que des erreurs bloquant l’exécution du code, mais l’esprit d’équipe et la bonne communication nous ont permis de surpasser ces problèmes. De plus, cette communication entre les membres de notre équipe s’est améliorée, ce qui a permis de comprendre et de résoudre plus rapidement les problèmes.

## Bilan pédagogique

Au point de vue pédagogique, nous avons eu une nette amélioration entre le début du projet et la fin. En effet, nous avons dû nous adapter aux langages qui étaient totalement nouveaux pour nous. De plus, nous avions des réunions hebdomadaires ce qui nous forçait à développer et à apprendre rapidement pour arriver à un résultat nécessaire à la réunion.

Durant cette période, nous avons amélioré nos compétences en C++. À force d’utiliser GIT, nos connaissances, concernant cet outil de gestion de version, se sont accrues et par conséquent nous avons développé des automatismes comme faire des commits plus précis et plus logiques comparé à nos premiers commits qui étaient très chargés et qui rentraient en confrontation avec les autres codes des membres de notre équipe.

## Le bilan technique

Sur le plan technique, nous sommes satisfaits du travail effectué. Le jeu est fonctionnel, cependant l’aspect graphique pourrait être améliorer. Il reste également quelques fonctionnalités qui seraient bon d’implémenter comme par exemple la création de terrain de façon dynamique. C’est-à-dire que l’utilisateur choisis où positionner les différentes entités.