Labo 4 - Buffy

POA

Walid Slimani et Timothée Van Hove

16 juin 2024

Table des matières

Introduction

Dans ce projet, nous avons développé une simulation intitulée "Buffy vs. Vampires" où Buffy, la tueuse de vampires, doit protéger les humains d'une menace vampirique croissante. Le programme permet de simuler les interactions entre Buffy, les humains et les vampires sur une grille, en mode tour par tour. Les entités se déplacent, chassent et attaquent selon des comportements définis, avec Buffy cherchant à éliminer les vampires et les vampires chassant les humains. Cette simulation inclut également la possibilité d'effectuer des statistiques sur un grand nombre de simulations pour évaluer les performances et le succès de Buffy.

Compilateur et options de compilation

Compilateur : `g++ version 12.2.0`

Version c++ : `23`

Build systems : `cmake 3.23.2` et `ninja 1.10.1`

Options de compilation : `-Wall -Wextra -Wpedantic -Wconversion -Wsign-conversion -Wvla -Werror`

Compiler et lancer le programme

Note : Ninja et CMake doivent être installés sur votre machine

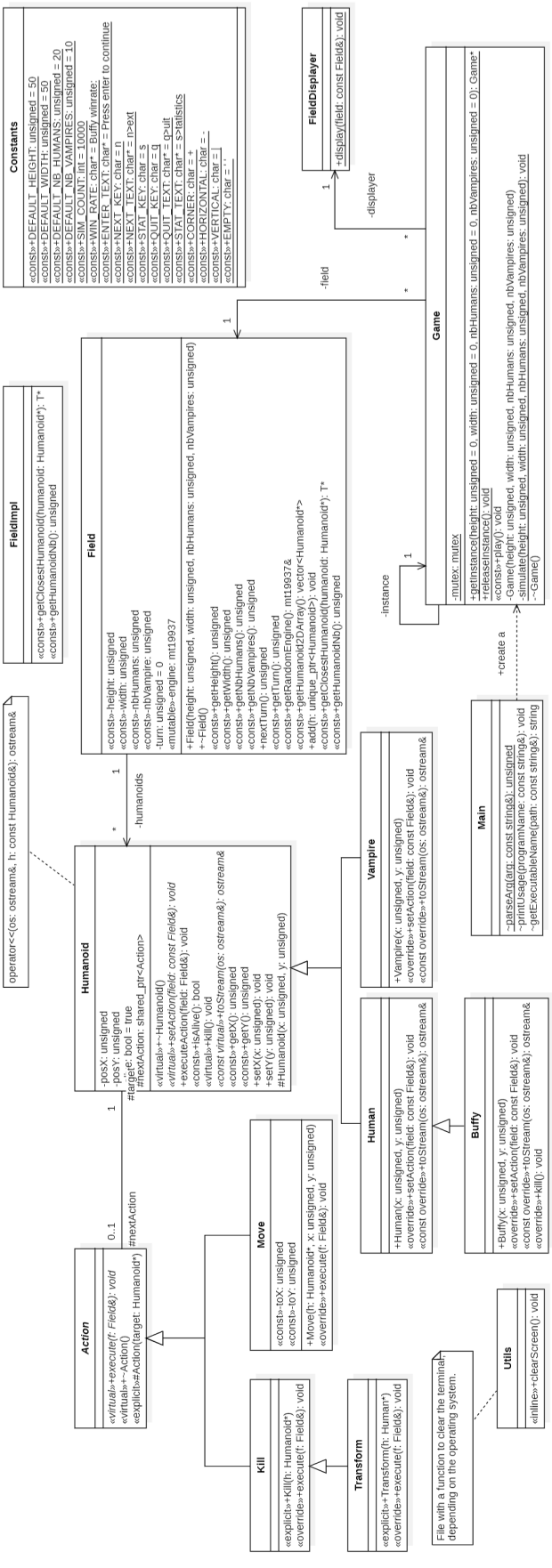
Windows

```
# Depuis la racine du projet
New-Item -Path build -ItemType Directory -Force; cd build; cmake .. -G "Ninja"; ninja; .\buffy.exe;
 ↪  cd ..
```

Linux

```
# Depuis la racine du projet
mkdir -p build && cd build && cmake .. -G "Ninja" && ninja && ./buffy && cd ..
```

Diagramme de classes



Choix d'implémentation

Notre programme "Buffy" est structuré de manière à séparer clairement les différentes responsabilités, ce qui facilite l'évolution du code. La structure générale du code se compose de plusieurs composants :

Le fichier `main.cpp` est le point d'entrée et est responsable de l'initialisation du jeu en fonction des paramètres fournis en ligne de commande ou de l'utilisation des valeurs par défaut. Le cœur du programme est la classe `Game`, implémentée en tant que singleton pour garantir qu'une seule instance du jeu est active à tout moment. La classe `Game` gère le déroulement des parties et l'initialisation et la libération des ressources. Le plateau de jeu est représenté par la classe `Field`, qui gère la création et le positionnement des entités (Buffy, humains, et vampires) ainsi que l'évolution de chaque tour de jeu.

Les entités sont modélisées par la classe de base `Humanoid`, avec des classes dérivées pour Buffy, les humains et les vampires, chacune implémentant des comportements spécifiques. Les actions que ces entités peuvent effectuer, comme se déplacer ou attaquer, sont encapsulées dans des classes dérivées de `Action`. L'affichage du plateau est géré par la classe `FieldDisplayer`, qui se charge de la représentation visuelle du jeu dans la console. Les fonctions utilitaires et les constantes globales sont regroupées dans des fichiers séparés pour améliorer la modularité et la réutilisabilité du code.

Les choix de conception effectués dans ce programme ont été faits pour avoir une architecture extensible. La séparation des responsabilités entre différentes classes, chacune ayant une responsabilité bien définie, améliore la lisibilité du code et facilite la maintenance. Par exemple, la gestion des entités sur le plateau est confiée à la classe `Field`, tandis que la classe `FieldDisplayer` s'occupe uniquement de l'affichage, ce qui permet de modifier l'interface utilisateur sans impacter la logique de jeu. L'utilisation de `std::unique_ptr` pour la gestion des entités garantit une gestion appropriée de la mémoire, évitant les memory leaks et simplifiant le transfert de propriété des objets. Le pattern Command est utilisé pour implémenter les actions des entités, permettant d'encapsuler les requêtes d'action et de faciliter leur gestion et extension future. Cette approche permet de développer un programme évolutif, pouvant s'adapter aux futurs besoins et modifications.

Tests manuels

Voici la liste des test manuels que nous avons effectués :

Test #	**Description**	**Résultat Attendu**	Résultat
1	Initialisation par défaut	Le jeu se lance avec les valeurs par défaut : hauteur et largeur de 50, 20 humains, 10 vampires.	Ok
2	Initialisation avec paramètres	Le jeu se lance avec les paramètres fournis : hauteur et largeur de 30, 15 humains, 5 vampires.	Ok
3	Affichage initial du plateau	Le plateau est affiché avec une bordure, Buffy, humains et vampires positionnés aléatoirement.	Ok
4	Quitter le jeu	Le jeu se termine proprement sans erreurs.	Ok
5	Avancer au prochain tour	Le plateau est mis à jour avec les nouvelles positions des entités.	Ok
6	Afficher les statistiques	Les statistiques sont affichées avec le taux de victoire de Buffy après 10 000 simulations.	Ok
7	Déplacement d'un humain	L'humain se déplace aléatoirement vers une case adjacente libre.	Ok
8	Déplacement de Buffy vers un vampire	Buffy se déplace vers le vampire le plus proche.	Ok
9	Attaque de Buffy sur un vampire	Buffy attaque et tue le vampire adjacent.	Ok
10	Transformation d'un humain en vampire	Le vampire transforme l'humain adjacent en vampire.	Ok
11	Mort d'un humain par un vampire	Le vampire tue l'humain adjacent.	Ok
12	Cycle complet de tour	Le jeu progresse sans erreurs, avec les entités se déplaçant et interagissant correctement.	Ok

Test #	Description	Résultat Attendu	Résultat
13	Bordures du plateau	Les entités ne sortent pas des limites du plateau.	Ok
14	Fin de la partie lorsque tous les vampires sont tués	Le jeu indique la victoire de Buffy lorsque tous les vampires sont éliminés.	Ok
15	Fin de la partie lorsque tous les humains sont transformés ou tués	Le jeu continue jusqu'à ce que Buffy soit la seule entité vivante ou qu'elle soit entourée uniquement de vampires.	Ok
16	Affichage après plusieurs tours et résilience aux entrées utilisateurs invalides	Le jeu doit gérer les entrées invalides proprement et continuer de fonctionner correctement, avec un affichage cohérent à chaque tour.	Ok

Conclusion

Ce projet nous a permis de mettre en pratique plusieurs concepts avancés de la programmation en C++, notamment la gestion des ressources avec des smart pointers, l'utilisation des patrons de conception comme le singleton et la commande, ainsi que la manipulation des collections STL. Notre programme répond aux exigences spécifiées, permettant de simuler les interactions entre les entités et d'évaluer les performances de Buffy à travers des simulations statistiques.