Buffy la tueuse de vampires

LABO 4

Forestier Quentin & Herzig Melvyn

POO2 – 01.06.2021

Table des matières

Introduction

Ce laboratoire porte sur l'implémentation de la simulation de Buffy la tueuse de vampires. L'application permet de simuler les comportements des différentes entités, en mode tour par tour. Il est aussi possible d'effectuer des statistiques permettant d'obtenir le taux de réussite de Buffy.

Les 3 entités de la simulation sont :

- Buffy
- Les vampires
- Les humains

Les humains se promènent simplement sur le terrain. Les vampires doivent chasser les humains, afin de les tuer ou de les transformer en vampires. Buffy doit chasser les vampires, afin de tous les tuer.

Buffy réussit à sauver l'humanité si aucun vampire n'est sur le terrain, et qu'au moins un humain est en vie.

Implémentation

Field

Un Field n'est finalement qu'un contrôleur d'humanoïde. Nous avons choisi de stocker les humanoïdes dans une liste de pointeur, et non dans un tableau 2 dimensions. En effet, la plupart des interactions entre entités se font au moyen de la recherche du plus proche. Un tableau 2 dimensions ne nous aurait pas aidés, et aurait rajouté une multitude de cases vides.

La classe Field contient la méthode nextTurn(), qui fait interagir les entités entre elles pour passer au tour suivant. Elle possède également des méthodes génériques, qui permettent de créer un humanoïde d'un certain type (Buffy, Human, Vampire) ou encore une méthode permettant de récupérer l'humanoïde du type choisi le plus proche d'une position.

Humanoid

Toutes les entités héritent de la classe abstraite Humanoid. Elles implémentent de ce fait, des méthodes permettant de s'afficher, de se déplacer ou d'obtenir des informations sur l'humanoïde.

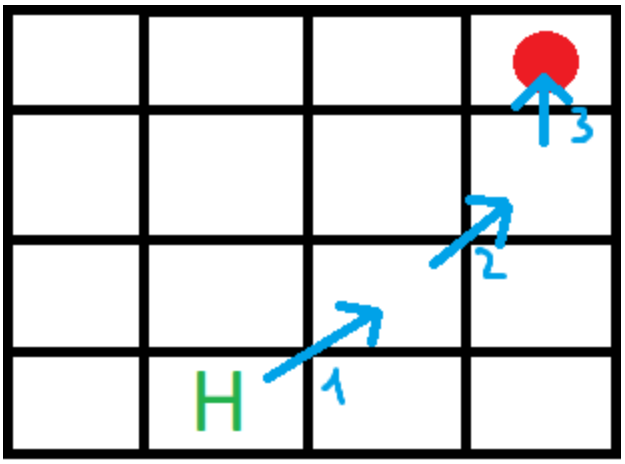
Les humanoïdes implémentent un système d'actions. Ce système d'actions se base sur le design pattern Command, qui dans ce contexte, permet qu'à chaque tour, tous les humanoïdes définissent leur prochaine Action, puis les exécutent dans un deuxième temps. Ceci permet de ne pas prioriser en fonction de l'ordre de la liste d'humanoïdes du Field.

Cependant, il est possible que 2 vampires essayent de tuer le même humain au même tour, et donc qu'un des 2 vampires « gâchent » un tour.

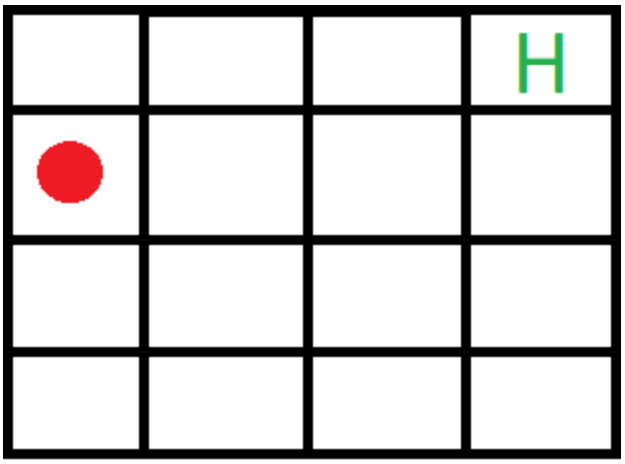
Les humains et les vampires héritent aussi de la classe abstraite IKillable, permettant de tuer l'entité grâce à la méthode kill().

Il faut aussi noter que pour le déplacement des humains, ils devront arriver à leur destination avant de changer de direction.

Voici le déroulement :



Ici, on voit les déplacements de l'humanoïde vers sa destination pendant 3 tours. En arrivant à sa destination, une nouvelle sera générée aléatoirement.



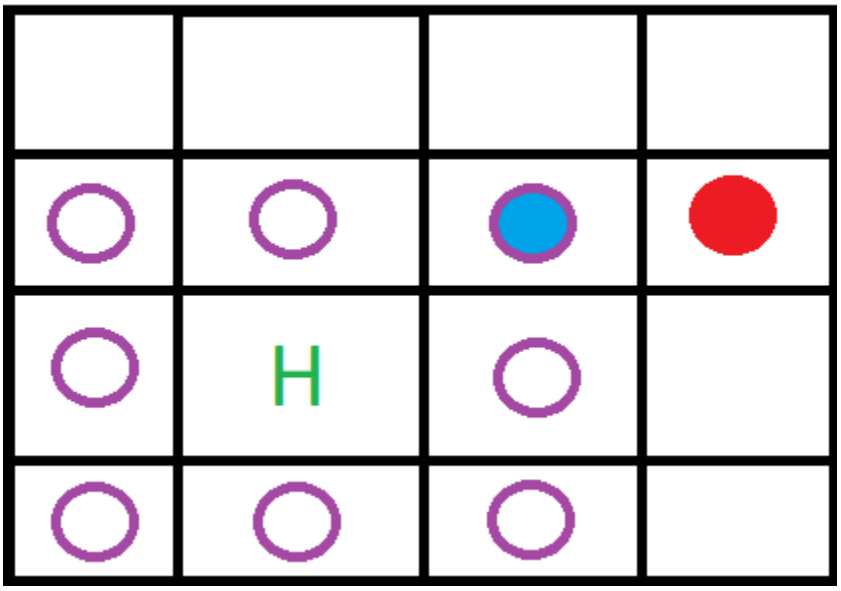
Action

Pour les actions des humanoïdes, nous avons choisi de suivre le design pattern Command. Cela permet à l'humanoïde de stocker et d'exécuter une action, quelle qu'en soit la nature.

Chaque action va simplement appeler une méthode de l'humanoïde qui l'utilise afin d'effectuer l'action.

Move

L'action Move est un peu plus compliquée, car elle doit calculer la nouvelle position à partir d'une direction. La direction peut être une position aléatoire ou bien l'emplacement d'une cible, cela ne fait aucune différence pour Move.



Sur le schéma, l'humanoïde doit se déplacer dans la direction donnée. Il doit donc se rapprocher au maximum de cette dernière. On voit les déplacements possibles de l'humanoïde ainsi que sa nouvelle position lorsque l'action Move sera exécutée.

Display

Pour l'affichage de la simulation, nous avons choisi de passer par un Displayer abstrait, AbstractDisplay. Nous avons par la suite implémenté une sous-classe pour l'affichage en mode console, StandardDisplay. Cet affichage étant cross-plateforme, il est impossible de modifier la couleur d'écriture en fonction de l'entité, ou encore d'effacer la console.

C'est pourquoi nous avons ajouté 2 sous-classes de StandardDisplay : WindowsDisplay et LinuxDisplay, qui elles, permettent d'afficher les entités dans des couleurs différentes ou encore d'effacer la console, en fonction de l'OS de l'utilisateur.

Il est conseillé d'exécuter le programme dans une console externe à l'environnement de développement afin de voir le résultat réel.

Input

Pour la gestion des entrées utilisateurs, nous avons implémenté un KeyListener, ainsi que des événements Event. Le KeyListener attend grâce à la méthode getNextInput l'entrée utilisateur, et en fonction de l'entrée utilisateur, retournera un événement déclenchable ou un autre.

Il suffira de déclencher l'événement au moment souhaité grâce à la méthode trigger pour que la fonction désirée soit exécutée.

Si l'entrée utilisateur est fausse, getNextInput attendra une nouvelle entrée.

BuffyAndVampire

BuffyAndVampire est le contrôleur de simulations. Il permet l'affichage du Field principal, ainsi que la gestion des entrées utilisateurs et leur méthode associée.

Le constructeur de BuffyAndVampire demande un AbstractDisplay, la taille du Field, le nombre d'humains et le nombre de vampires. Il faut noter que les statistiques du pourcentage de victoire de Buffy sont effectuées dans les conditions envoyées en paramètres.

Conclusion

Au terme de ce laboratoire, nous sommes arrivés à implémenter une solution fonctionnelle de la simulation de Buffy la tueuse de vampires. Nous avons choisi de déléguer l'affichage à un Displayer, ce qui permet de modifier l'affichage facilement.

Nous n'avons pas jugé nécessaire de faire des tests, étant donné la nature de simulation du laboratoire et le peu d'interactions avec l'utilisateur.

Annexes

- Code source (.h et .cpp)
- UML du projet