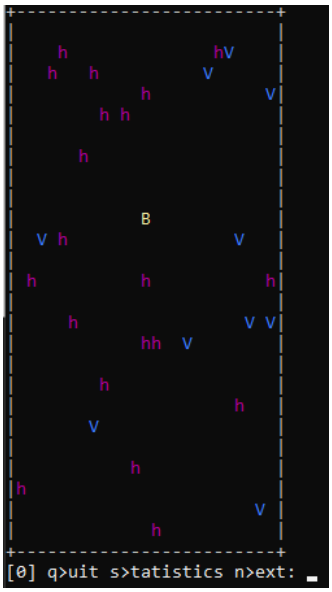
POO2 - Laboratoire 4
Buffy

Cours : POO2 | Auteurs : Zwick Gaétan / Maziero Marco | 24.05.2021

Table des matières

Introduction



Pour ce laboratoire, le travail à réaliser est un jeu se déroulant sur une grille de jeu possédant plusieurs humanoïdes se déplaçant et ayant des comportements différents selon la classe d'humanoïde.

Il est possible de faire avancer les tours de jeu grâce aux touches `n` et `return`. Le mode statistiques peut être lancé en appuyant sur la touche `s`. Finalement, le jeu peut être quitté avec un appui sur la touche `q`.

Il existe trois types d'humanoïdes :

- Buffy (B), chasse le vampires (V)
- Les vampires (V), chassent les humains (h)
- Les humains (h), se déplacent sur la grille

Structure du code

Diagramme de classes (UML)

Le diagramme de classes UML est disponible en annexe.

La classe Game et les instances de Field

La classe Game s'occupe de tout ce qui correspond à l'affichage d'une instance de jeu et de la gestion des entrées utilisateur.

La fonction `start()` démarre le jeu en créant une instance de `Field`, en l'affichant et en bloquant en attente de l'entrée utilisateur. Le code réagit ensuite en fonction de l'entrée choisie :

- `q` : Terminer le programme
- `s` : Lancer le calcul des statistiques de succès de Buffy
- `n` ou `return` : Tour suivant pour l'instance de Field affichée

L'affichage d'un Field

Une surcharge de l'opérateur `<<` est disponible pour la classe `Field` afin de permettre l'affichage de la grille de jeu et des humanoïdes s'y trouvant.

Un tour peut toutefois se jouer sans que la grille soit affichée. L'affichage étant un processus long, ne pas l'obliger permet d'effectuer des statistiques de façon assez rapide en effectuant tous les tours d'une partie sur un `Field` donné.

La classe ConsoleManager

Une petite classe `ConsoleManager` est utilisée pour gérer l'affichage dans la console. Elle offre une fonction membre statique permettant de déplacer le curseur dans la console. Cela est très pratique pour gérer l'affichage des humanoïdes ou tes textes de l'interface.

Fluidité de l'affichage dans la console

Plusieurs techniques ont été testées pour obtenir un affichage fluide de la grille et des humanoïdes. Au début, pour chaque appel à l'opérateur `<<`, l'intégralité de la grille et des humanoïdes était redessinée (des espaces vides étaient écrits là ou il n'y avait aucun élément). Cependant, cette technique provoquait un clignotement de la console car tout réécrire prenait du temps.

Puis finalement, une solution a été trouvée en ajoutant un attribut liste `oldDisplayCoords` à la classe `Field`. Cette liste est utilisée pour stocker les coordonnées des éléments dessinés dans la console. Lorsqu'un humanoïde est dessiné dans la console, ses coordonnées sont ajoutées à cette liste.

Lorsqu'il faut réafficher la grille de jeu, avant de dessiner, le programme va parcourir cette liste de coordonnées, et pour chacune d'elles, il va effacer l'éléments qui étaient à cette position.

Cela efface complètement le clignotement de la console car au lieu de redessiner 50x50 caractères pour une grille de taille 50 contenant 20 humanoïdes, il ne faudra qu'effacer 20 caractères puis en dessiner 20 autres (40 écritures au total).

Recherche de l'humanoïde le plus proche

La classe `Field` possède une fonction `findNearby()` prenant en paramètre l'humanoïde qui recherche et le type d'humanoïde recherché.

Pour chaque humanoïde de la liste, le type est vérifié et une distance euclidienne est calculée (grâce aux outils offerts par la librairie `<cmath>`). L'humanoïde se trouvant à la plus petite distance est retourné à la fin de la fonction.

Les humanoïdes

La super classe `Humanoid` représente un humanoïde pouvant effectuer des actions dans la grille de jeu. La classe possède plusieurs fonctions membre :

- `moveRandomly()` : Observe l'état du jeu et génère une action de mouvement
- `chaseHumanoid()` : Observe l'état du jeu et génère une action de poursuite
- `attackHumanoid()` : Génère une action d'attaque sur un autre humanoïde
- `executeAction()` : Exécute l'action stockée comme attribut de la classe

Elle possède aussi des fonctions virtuelles pures à implémenter :

- `getDisplayChar()` : Donne le caractère à afficher dans la console
- `getDisplayColor()` : Donne la couleur du caractère à afficher
- `getSpeed()` : Donne le nombre de cases à avancer par tour
- `setAction()` : Définit le comportement de l'humanoïde en générant sa prochaine action

La fonction `setAction()` est très importante car c'est elle qui va définir le comportement de l'humanoïde en générant sa prochaine action. Buffy va par exemple chasser le vampire le plus proche ou bouger aléatoirement. Tout cela est calculé dans `setAction()`.

Les actions

Les actions sont représentées par diverses classes héritant de la super classe Action. A chaque tour, toutes les actions sont calculées pour chaque humanoïde, puis elles sont toutes exécutées. Cela permet de ne pas avantager certains humanoïdes qui seraient en premier dans la liste.

Cette implémentation correspond en fait au patron de conception comportemental `Commande`. Les humanoïdes possèdent une action a effectuer, qu'il vont calculer, puis un appel à la fonction `execute()` lancera par liaison dynamique l'exécution de la bonne action.

Pour ce jeu, il existe 3 types d'actions différentes :

- `Move` : permet de bouger un humanoïde vers une destination donnée. Les humanoïdes peuvent occuper la même case.
- `Kill` : Tue un humanoïde donné
- `Convert` : 50% de chances de transformer l'humanoïde donné en vampire, 50% de chances de le tuer

Les humanoïdes, dans leur fonction `setAction()`, vont utiliser les fonctions membres `moveRandomly()`, `chaseHumanoid()` et `attackHumanoid()` de la classe `Humanoid` pour générer des actions qui seront stockées dans l'humanoïde concerné et exécutées par un appel à `executeAction()`.

Statistiques

Lors d'un appui sur la touche `s`, le programme va lancer 10'000 simulations de parties. Cela est fait dans la classe `Game` grâce à la fonction `calculateBuffySuccess()` qui retourne un pourcentage de succès de Buffy.

Buffy remporte une partie lorsqu'elle a tué tous les vampires et qu'il reste des humains en vie. Cette vérification est possible grâce à la fonction `getNbentity()` de la classe `Field` qui retourne le nombre d'entités d'un type donné encore présentes dans la grille.

La fonction `calculateBuffySuccess()` va créer une instance de `Field`, jouer tous les tours et regarder l'issue du jeu. Elle fera ceci 10'000 fois afin d'obtenir un nombre correspondant au parties gagnées par Buffy.

Le retour de la fonction correspond au pourcentage de succède de Buffy :

$$\frac{nbVictoires}{nbSimulations} * 100$$

Résultats

Pour 10'000 simulations sur un grille de taille 50 (murs non compris) contenant 20 humains et 10 vampires le résultat est approximativement de **24%**

```
Calculating statistics, simulation : 10000/10000
Percentage of Buffy wins : 24.37%
```

Tests

Description du test	Résultat attendu	Réussi ?
Buffy poursuit le vampire le plus proche	Buffy se déplace de 2 cases vers le vampire le plus proche	OK
Quand elle est a portée, Buffy tue le vampire	Le vampire disparait	OK
Un vampire tué ne peut convertir un humain	Disparition du vampire, l'humain a portée reste	OK
Un vampire poursuit l'humain le plus proche	Le vampire se déplace de 1 case vers l'humain le plus proche	OK
Un humanoïde tué disparait de la grille de jeu	Disparition de l'humanoïde	OK
Terminer la partie lorsqu'il n'y a plus de vampires et qu'il reste des humains	Affichage du message de succès de Buffy	OK
Terminer la partie lorsqu'il n'y a plus de vampires et qu'il n'y a plus d'humains	Affichage du message de fin de partie	OK
Terminer la partie alors qu'il reste des vampires et des humains	Aucun message affiché	OK
Appui sur la touche n	Tour suivant	OK
Appui sur la touche return	Tour suivant	OK
Appui sur la touche s	Démarrage du calcul des statistiques	OK
Appui sur la touche q	Fin du programme	OK
Un humain à portée d'un vampire est parfois tué	L'humain disparait	OK
Un humain à portée d'un vampire est parfois transformé en vampire	L'humain disparait et un vampire apparait à la même position	OK
Buffy se déplace aléatoirement lorsqu'il n'y a plus de vampires	Buffy se déplace de 1 case aléatoirement	OK
Les humains se déplacent aléatoirement	Les humains se déplacent de 1 case aléatoirement	OK
Les vampires sont immobiles lorsqu'il n'y a plus d'humains	Le vampires ne se déplacent plus	OK

Annexes

Diagramme de classes UML