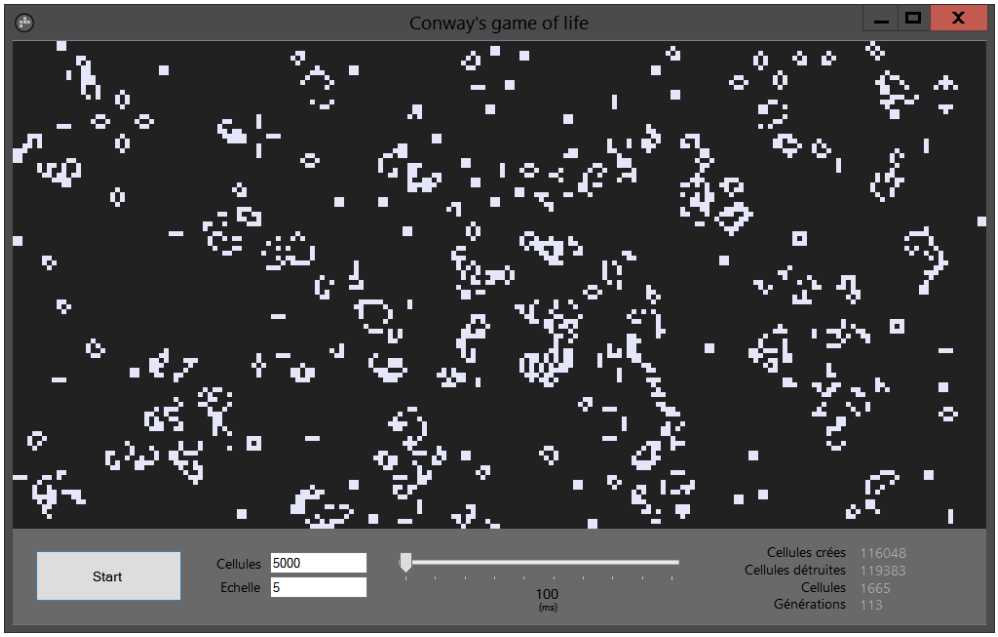
UNIVERSITÉ DE FRANCHE-COMTÉ

Projet APOO 2024 - Prédateurs et Proies

L1 Licence Informatique

Kilian GAILLARD, Matthéo GIRARDCLOS

Le but de ce projet était de créer un jeu à 0 joueur, ayant pour but de nous introduire quant à la tâche de réaliser un projet qui mêle algorithmique et programmation orientée objet, en autonomie.



Exemple de jeu à 0 joueur : Le jeu de la vie

Table des matières

1 Fonctionnalités implantées pour la version minimale

Avant de commencer, nous nous excusons, notre programme présente une erreur d'éxécution, que nous n'avons malheureusement pas pu résoudre. Le programme ne se lance pas toujours, et nécessite quelques fois d'être interrompu, puis réexécuter, jusqu'à ce que la phase initiale soit crée. Plus précisément, notre méthode generationEntites( ) ne s'éxécute pas toujours jusqu'au bout, et s'interrompt parfois en plein milieu, ce qui empêche l'éxécution complète du jeu. Nous nous excusons pour la gêne occasionée.

1.1 Génération de la grille et initialisation du jeu

La Grille de jeu est générée tout d'abord par une première méthode, nommée generationPlateau( ). Elle crée une matrice d'objets de type Case, qui sont initialisées par l'algorithme comme une Case vide, et, sur les bords de la grille de jeu, les bordures sont placées (lorsque soit la case consultée voit sa ligne ou sa colonne, être la première, ou la dernière).

Ensuite, une méthode nommée generationEntites( ) s'éxécute, et fait apparaître de manière aléatoire les entités, qui sont de type Poule, Renard, Lapin et Chasseur. Cela est réalisé à l'aide de boucles bornées (for), qui s'éxécute le nombre de fois que l'on souhaite mettre l'entité en question, et génère au cours de chaque passage dans la boucle une position aléatoire, à laquelle on vérifie que la case associée soit vide. Si tel est le cas, l'entité est placée, dans le cas contraire, une nouvelle génération d'une position aléatoire est réalisée, jusqu'à ce que la case en question soit vide, et que l'entité soit placée (condition réalisée à l'aide d'un while).

1.2 Gestion des intéractions entre entités

Ensuite, nous avons créer une méthode ayant pour rôle de gérer toutes les intéractions entre toutes les entités, nommée InteractionFinale( ). Pour rendre le fonctionnement de cette méthode plus simple, le polymorphisme y est utilisé. Cette fonction est tout d'abord définie, une première fois, dans notre classe Case : une super-classe, qui possède comme sous-classes toutes les entités présentes dans le jeu : les poules, lapins, renards, chasseurs, et même nos bordures.
Cette méthode est ensuite re-définie dans chaque sous-classe, et est réécrite, ré-adaptée à chaque cas différent : par exemple, dans le cas d'une Poule, si une entité est rencontrée, on fuit immédiatement, dans le cas contraire, soit on se retrouve face à une Case vide, et alors le déplacement est possible, soit on se retrouve face à une bordure, et alors on rebondit sur cette dernière. Ces deux dernières manières d'intéragir avec des Cases non-entités, sont traités dans une seule et même méthode, elle aussi employant le polymorphisme, nommée Redirection( ).

1.3 Déplacements dans la Grille de jeu, et rebond avec une Bordure (méthode Redirection)

Cette méthode gère donc les déplacements d'une entité sur notre grille de jeu, et également les rebonds effectués sur les bordures. Elle est tout d'abord définit dans notre super-classe Case, et s'éxécute lorqu'elle est appeée avec comme entrée, un objet de type Case, qui est une case vide. Celle-ci supprime l'entité qui souhaite se déplacer, crée une nouvelle Case vide à sa place, puis se supprime elle-même de la matrice, pour laisser à place à l'entité qui désirait se déplacer. Cette Case vide, sur laquelle est appelée la méthode redirection( ), est en fait la case sur laquelle l'entité désire se déplacer.
Si l'objet qui appelle la méthode redirection( ) n'est pas seulement une Case, mais également une Bordure, alors la méthode sera appelée dans la classe Bordure, qui y présente la fonction vide, car elle appele ici, à son tour, la méthode du même nom, mais dans l'une de ses sous-classes : BordureHorizontale, BordureVerticale, ou Coin
Lorsque la méthode est appelée ici, elle s'éxécute de manière à créer cet effet de rebond désiré.

2 Fonctionnalités pour la version supplémentaire

Aucun ajout, ou fonctionalité supplémentaire n'a été réalisé au cours de ce projet.

3 Classes du projet

3.1 Les différentes classes

3.1.1 Classe Plateau

La classe Plateau a principalement pour rôle, la créaton du plateau de jeu, ainis que l'initialisation de ses paramètres, de l'affichage du plateau de jeu, et même de gérer l'ajout, ou la suppression de Cases en son sein. Tout d'abord cette classe possède trois attributs : un nombre de lignes, un de colonne, et un tableau à deux dimensions de Cases. Ensuite cette classe possède également une méthode qui permet de calculer le nombre d'entités restantes sur le plateau.

3.1.2 Classe Position

La classe Position a tout simplement pour but de créer des objets Position, qui sont des couples de valeurs : un numéro de ligne, et un numéro de colonne. Cette classe a pour but, taut du long de notre projet, d'accéder et de gérer, modifier l'emplacement d'une entité, de retrouver celle-ci. Cette classe possède tout simplement les deux attributs mentionnés, ainsi qu'un getter pour chacun d'eux.

3.1.3 Classe Direction

Cettte classe Direction a, elle, pour but de créer, et de donner accès à la Direction d'un personnage, composée de deux attributs, une direction verticale rowDir, et une direction horizontale colDir. Ces attributs peuvent prendre pour valeur -1, 0 ou 1. Cette classe possède également des getters pour accéder aux attributs, un setter pour modifier la Direction d'une entité en plein cours d'une éxécution, ansi qu'une méthode d'inversion des valeurs des deux attributs, et d'une méthode permettant la génération d'une direction aléatoire.

3.1.4 Classe Case

Cette classe a pour rôle de décrire, de créer une Case, l'élément fondamental de notre jeu, et celui de gérer tout ce qui y a attrait, telle que les intéractions entre elles, les déplacements, les affichages. Cette classe possède cinq attributs : une direction, une position, un booléen attestant de si la case consultée a déjà joué ce tour-ci ou non, mais également un symbole (un caractère), permettant le bon affichage de celle-ci, ainsi qu'un plateau, qui nous permet de garder à tout moment, dans chaque méthode de la classe, ou de ses sous-classes, l'accès au plateau de jeu. Cette classe contient donc un grand nombre de méthodes, parmi les plus importantes de notre jeu, et est la véritable plaque tournante, de l'éxécution de notre programme.

3.1.5 Classe Personnage

Cette classe a simplement pour rôle de lier nos deux classes Prédateurs et Proies, tout en les différenciant d'une case vide, ou d'une bordure.

3.1.6 Classe Predateurs

Cette classe nous sert de liaison entre nos deux entités de type Prédateurs : le Renard et le Chasseur.

3.1.7 Classe Proies

Cette classe nous sert également, à la manière de la classe Prédateurs, de liaison entre nos deux entités de type Proies : la Poule et le Lapin.

3.1.8 Classe Bordure

Cette classe a pour rôle de définir, de gérer nos cases de type Bordure, qui sont des cases présentes à la limite de notre plateau de jeu. Elle y présente une méthode redirection( ) dont nous avons parlé, qui traite tous les cas d'intéractions et de rebonds avec une Case (qui sera systématiquement un Personnage). Cette méthoe est vide, car elle utilise le plymorphisme, et son foctionnement prend effet dans les trois sous-classes de Bordure, dont nous allons dès à présent parler.

3.1.9 Classe BordureHorizontale

Cette classe permet de créer chaque Bordure, qui sont présentes dans ce que l'on appelle une bordure horizontale (ayant sa valeur de ligne à 0, ou à la dernière ligne de la matrice). Cette classe contient la méthode Redirection( ), dont son fonctionnement a été modifié afin de rediriger toute entité qui tenterait de se déplacer sur cette Bordure, en changeant sa Direction (ici en inversant la valeur de sa direction de ligne en ligne : rowDir).

3.1.10 Classe BordureVerticale

Cette classe, de manière similaire à la précédente, permet de créer chaque Bordure, qui sont présentes dans ce que l'on appelle une bordure verticale (ayant sa valeur de colonne à 0, ou à la dernière colonne de la matrice). Cette classe contient la méthode Redirection( ), dont son fonctionnement a été modifié afin de rediriger toute entité qui tenterait de se déplacer sur cette Bordure, en changeant sa Direction (ici en inversant la valeur de sa direction de colonne en colonne : colDir).

3.1.11 Classe Coin

Enfin, cette troisième et dernière sous-classe de Bordure, crée 4 différentes cases, qui sont les 4 coins de notre matrice. La méthode redirection( ) y est redéfini, mais de manière différente : cette fois-ci, lorsqu'une entité tente de se déplacer sur un des coins, sa direction est totalement inversée : son attribut rowDir et son attribut colDir sont inversés.

3.1.12 Classe Poule

Cette classe a tout simplement pour rôle de créer une Poule et de gérer les intéractions qu'elle doit avoir avec une autre Case : elle possède donc une méthode, interactionFinale( ), dont nous avons déjà parlé, qui permet à la Poule soit de s'enfuir en cas de rencontre avec un autre Personnage, ou d'appeler la méthode redirection, qui s'appele dans classe Case, ou Bordure, en fonction du type d'entité auquel la Poule a affaire.

3.1.13 Classe Lapin

Cette classe Lapin est évidemment presque identique à la précédente, puisque les Lapin et les Poule, se doivent d'intéragir de la même manière avce leur environnement, à la seule différence que ben évidemment, celle-ci crée une case de type Lapin.

3.1.14 Classe Renard

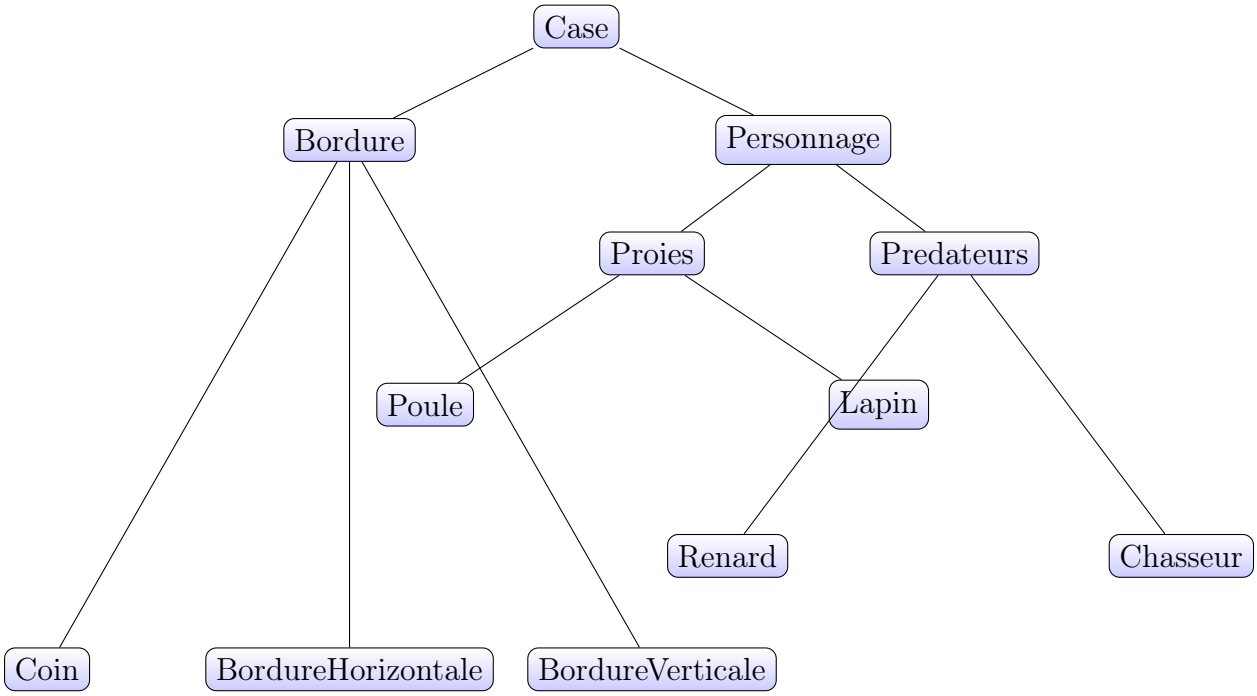
La classe Renard créer une case de type Renard, et possède également en son sein, la méthode interactionFinale( ) qui comme les deux classes précédentes, gère sa manière d'intéragir avec son environnement, à la différence que celle-ci est capable de supprimer un Personnage du plateau, si clui-ci se trouve sur sa route, et s'il s'agit là d'une Poule ou d'un Lapin.

3.1.15 Classe Chasseur

La classe Chasseur est donc quasiment identique à la précédente, excepté qu'elle crée une case de type Chasseur, et lui permet d'éliminer une entité sur son passage, s'il s'agit d'un Lapin ou bien d'un Renard.

3.2 Relations d'héritages entre les classes

Comme mentionné précédemment, c'est la classe Case qui se retrouve au sommet de la hiérarchie de notre héritage. En dessous d'elle, la classe Bordure, et la classe Personnage. La classe Bordure possède elle-même trois sous-classes, qui écrivent plus précisément les bordures du plateau, ce qui nous est fortement utile quant à l'implémentation de la méthode qui permet de gérer tout rebond avec une bordure. Et bien évidemment, comme expliqué auparavant, les différents types de personnages sont eux aussi hiérarchisés, par appartenance au type Proie, ou Prédateur.



4 L'"initialisation des paramètres du jeu

Pour l'initialisation des paramètres du jeu, qui sont : la taille du plateau (nombre de lignes et nombre de colonnes), ainsi que le nombre d'entités présentes, notre programme demande leur saisie, à l'utilisateur, celui-ci est donc libre de jouer avec les paramères qu'il désire.
Quand à notre condition d'arrêt du jeu, elle est saisie de manière prédéfinie, par le programmeur, afin de fixer quand le jeu s'arrêtera, afin d'empêcher toute erreur d'éxécution, dans le cas où l'utilisateur aurait souhaité jouer avec ce paramètre, ce qui pourrait conduire à une erreur d'éxécution.