**Université de Cergy-Pontoise**

**RAPPORT**

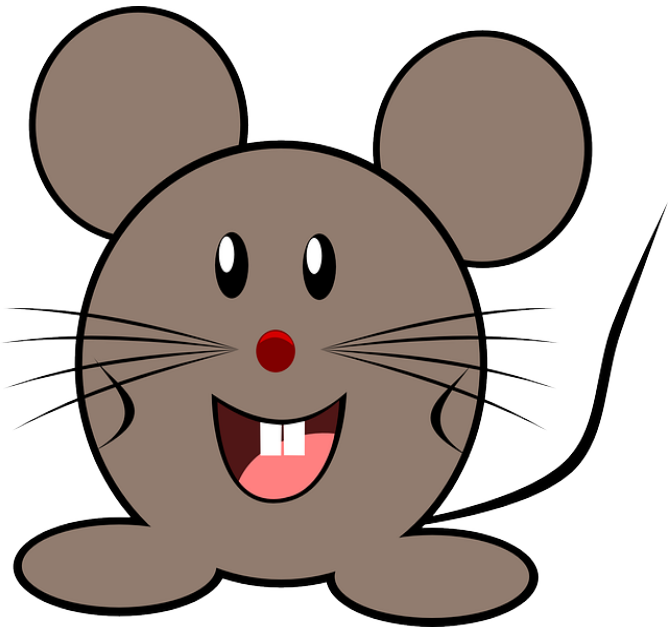
pour le projet Génie Logiciel
**Deuxième année de licence informatique**

sur le sujet

# Les souris coopératives

rédigé par

**AYAD Ishak, YAHIAOUI Anis, HACHOUD Rassem**



Avril 2018

# Table des matières

## Table des figures



## Remerciements


Avant tout développement de ce projet, il apparaît opportun d'addresser nos remirciements à tout ceux qui nous ont aidé pour la réalisation de ce projet. Nous tenons à remercier en premier lieu Monsieur Tianxiao Liu, notre encadrant lors de ce projet, auprés duquel nous avons pu bénéficier d'un grand soutien. Nous remercions également Monsieur Irenee Briquel pour son aide à la réalisation de ce document.

# 1 Introduction

**Contexte :** Dans le cadre de nos études en licence d'informatique, nous devons réaliser en trinôme, un projet de programmation Java pour le module de génie logiciel. Nous avons choisi le projet des souris coopératives car il est trés intéressant pour nous, d'étudier le concept de la communication, la gestion de mémoire et l'intelligence artificielle.

**Objet :** Créer une simulation où des souris évoluent sur une grille, en se nourrissant et en se communiquant.

**Outils de développement :** Nos outils de développement sont ceux qui nous ont été conseillés par notre enseignant, et que nous avons utilisé au cours de ce semèstre en Génie Logiciel et Projet. Nous avons utilisé la plateforme Java 9 ainsi que l'environnement de développement Eclipse. Nous avons synchronisé notre travail en utilisant le service web d'hébergement GitHub qui nous a beaucoup aidé pour tavailler en groupe. Enfin, ce rapport de projet a été rédigé avec LaTex sur le service web Overleaf.

**Structure du rapport** : Notre première partie concernera les spécifications de notre projet, elle contiendra toutes les fonctionnalités du logiciel. Ensuite nous parlerons de la partie réalisation, dans laquelle nous présenterons la conception détaillée de notre logiciel, et viendra après le manuel utilisateur où le fonctionnement sera expliqué. Puis nous aborderons la partie déroulement où le calendrier de notre travail et la répartition des tâches seront présentés, pour arriver ensuite à la conclusion où nous allons donner le bilan de notre projet.

# 2 Spécification

**Chapeau :** Nous avons présenté l'objectif du projet dans la section 1. Dans cette section, nous présentons la spécification de notre logiciel réalisé. Ceci correspond principalement au cahier des charges.

## 2.1 Environnement d'évolution des souris

Dans ce jeu, les souris évoluent sur une grille complexe, sur laquelle sont disposés plusieurs obstacles et diverses sources de nourriture. Cette grille est générée aléatoirement suivant certains paramètres donnés par l'utilisateur qui doit avoir un certain contrôle sur le déroulement de la simulation.

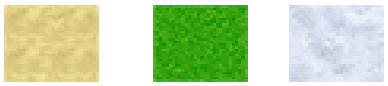


FIGURE 1 – Terrains

### 2.1.1 Les obstacles

Ils sont positionés aléatoirement sur la grille, avec la densité choisie par l'utilisateur. Ils empêchent les souris de se déplacer librement sur la grille. De plus, ils varient selon l'environnement : forêt, désert ou neige.

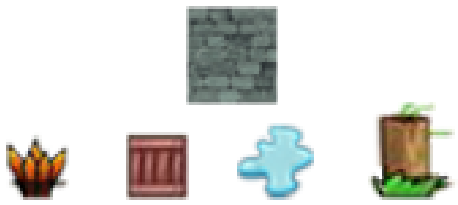


FIGURE 2 – Obstacles-Forêt

FIGURE 3 – Obstacles-Désert

FIGURE 4 – Obstacles-Neige

### 2.1.2 Les sources de nourriture

Elles apparaissent aléatoirement et spontanément au cours du temps sur la grille. Chaque source est composée de quelques unités de nourriture (quantité limitée), et chaque unité permet à une souris de survivre pendant un certain nombre de tours de jeu. Il est bien évident que plus une source de nourriture sera utilisée par les souris, plus elle s'épuisera vite.

FIGURE 5 – Nourriture

## 2.2 Caractéristiques et comportements des souris

### 2.2.1 Recherche de la nourriture

Les souris se déplacent sur la grille case par case, à la recherche des sources de nourriture. Au delà d'un cetrain temps bien déterminé, si une souris n'a pas mangé, elle meurt. Les sources de nourriture n'étant pas inépuisables, il est vital pour les souris d'explorer régulièrement la grille afin de trouver d'autres sources de nourriture, et de veiller au cours de leur exploration d'être toujours à portée d'un point de nourriture connu afin d'y retourner s'il le faut.

### 2.2.2 Mémoire et champ de vision

Les souris ont une vision limitée à quelques cases autour d'elles mais ont une excellente mémoire, elles se souviennent du chemin qu'elles ont parcouru, l'emplacement des sources de nourriture et des obstacles trouvés, ainsi que les souris rencontrées et les informations échangées.

### 2.2.3 Communication

Quand deux souris se rencontrent sur la même case ou sur deux cases adjacentes, elles peuvent communiquer leurs connaissances à propos de l'emplacement des sources de nourriture. Cela dépend de la mémoire des souris et leurs comportement, comme ce qui est illustré dans le tableau ci-dessous.D'autre part, si une souris reçoit plusieurs fois des informations éronnées de la part des autres, elle peut changer son comportement (réceptive devient nihiliste, cooperative devient égoïste).

| Comportemnts | Réceptive | Nihiliste |
|---|---|---|
| Coopérative | donne / reçoit les informations | donne / ne reçoit pas les informations |
| Égoïste | ne donne pas / reçoit les informations | ne donne pas / ne reçoit pas les informations |

FIGURE 6 – Souris coopérative

FIGURE 7 – Souris égoïste

### 2.2.4 Reproduction

Une souris bien nourrie pendant un certain nombre de tours de jeu donne naissance à une autre souris de comportement identique.

— Reproduction sexuée : quand deux souris de sexe opposé se rencontrent sur la même case ou sur deux cases adjacentes.
— Reproduction asexuée : dupliccation sans besoin de partenaire.

### 2.3 Exigences

— Permettre à l'utilisateur d'entrer un ensemble de paramètres pour initialiser la grille : (densités des obstacles, fréquence d'apparition de la nourriture et quantité, nombre de souris), et d'exécuter tour par tour la mise à jour de la grille (action des souris, apparition/épuisement des gisements de nourritures).
— Permettre à l'utilisateur de régler plus finement le comportement des souris (degré de coopération et degré de confiance en fonction de la taille et du nombre de gisement de nourriture, de sa propre faim, du nombre de fois où elle a été induite en erreur...etc).
— Permettre la simulation afin de montrer l'évolution de la population au fil des reproductions, en affichant les informations statistiques.
— Ajouter un système de journal à la première personne, une souris va raconter sa vie : de la naissance au décès, les échanges, les enfants, et toutes les activités réalisées. Ce journal doit être dynamique ou sous forme graphique avec des animations.

# 3 Réalisation

**Chapeau :** Nous avons présenté les spécifications du projet dans la section 2. Dans cette section, nous détaillons la conception de notre logiciel et les techniques informatiques utilisées.

## 3.1 Modélisation UML

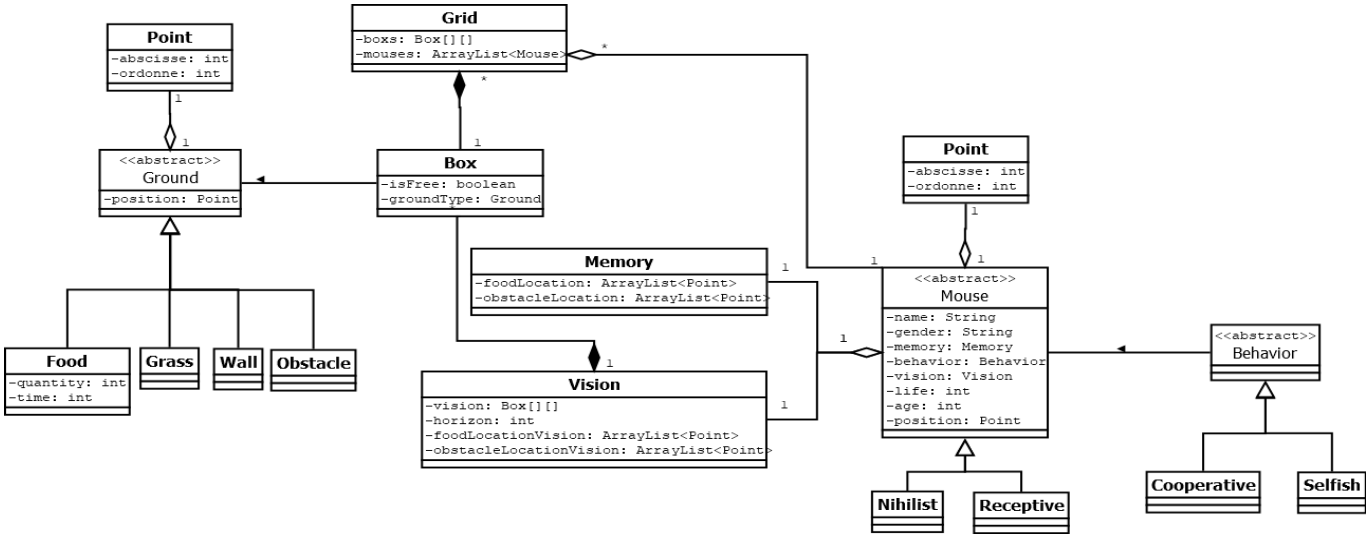



FIGURE 8 – Diagramme UML des classes de données

## 3.2 Conception détaillée

### 3.2.1 Génération de la grille

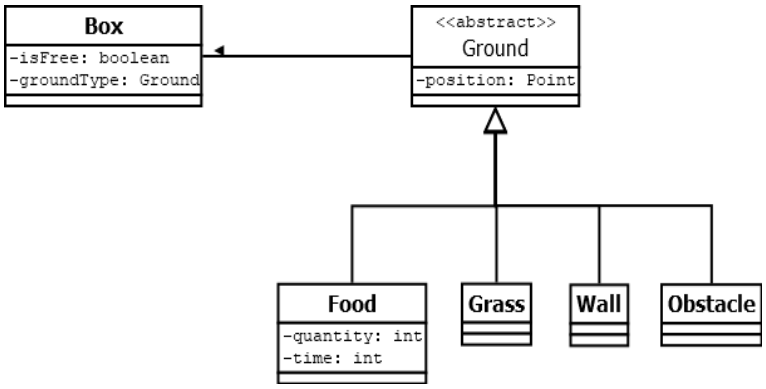



FIGURE 9 – UML-Terrain

— Fabriquation des éléments de la grille : les différentes sources de nourriture et les obstacles sont crées par la classe "BoxFactory" : design pattern "Simple factory".
— Pour faciliter la gestion des différents types de cases, on a mis en place le design pattern "Bridge".
— L'objet de type "Grille" est relativement complexe. On a consacré une classe entière : "GridBuilder" pour le construire, en séparant sa construction de sa représentation : utilisation de design pattern "Builder".
— Pour générer la grille, on a créé d'abord un tableau à 2 dimensions, dans lequel on a placé les objets représentons les éléments de la grille : le sol, les murs au 4 cotés de la grille. Ensuite, on a placé

aléatoirement les obstacles en laissant une certaine distance entre eux. Enfin, on a placé les sources de nourriture et les souris dans les cases libres (ne contenant pas un obstacle).

### 3.2.2 Les souris

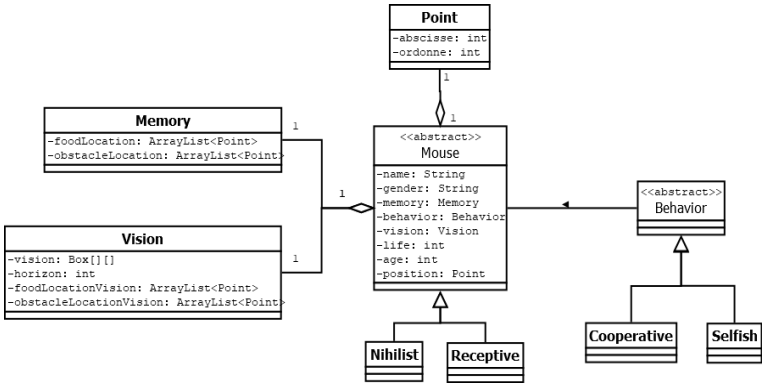



FIGURE 10 – UML-Souirs

— Création : Afin de centraliser la création des objets de type "Souris" dans un seul endroit du projet, on a utilisé le design pattern "Simple Factory".
— Comportement : Pour assurer une certaine souplesse dans la gestion des comportements des souris, on a utilisé 2 classes abstraites "Mouse" et "Behavior" : mise en place de design pattern "Bridge".
— Mémoire : On a créé une classe "Memory" dont les attributs sont principalement des "ArrayList" permettant de stocker les positions des obstacles rencontrés par la souris, et des sources de nourriture qu'elle a déjà consomées, ainsi que les événements de sa carrière.
— Vision : On a créé une classe "Vision" dans laquelle on a mis en place une méthode qui parcourt juste la matrice de cases qui se situent à une distance précise autour de la souris, puis elle stocke les positions des obstacles et des sources de nourriture dans deux "ArrayList<Point>" différentes, qui seront ensuite stockées au niveau de la mémoire de la souris.
— Déplacement : Pour gérer le déplacement des souris, on a créé une classe "Direction" , qui est une classe d'énumération, elle définie 4 constantes de déplacement de type "String" : Up,Down,Right,Left. Ensuite, au niveau de la classe "Souris" on a mis en place un attribut de type "Direction", et pour déplacer une souris, il suffit de changer la valeur de cette attribut.
— Pour dériger une souris vers une source de nourriture, on calcule d'abord par la règle de "Pythagore" la distance entre la souris en question et toutes les sources de nourritures stockées dans sa mémoire. Ensuite, pour trouver le bon chemin, on utilise notre algorithme qui traite les différents cas de l'emplacement de la souris par rapport à la source de nourriture la plus proche. La souris peut se situer dans 8 positions différentes par rapport à la source de nourriture, et pour chaque cas, on applique un traitement bien précis :

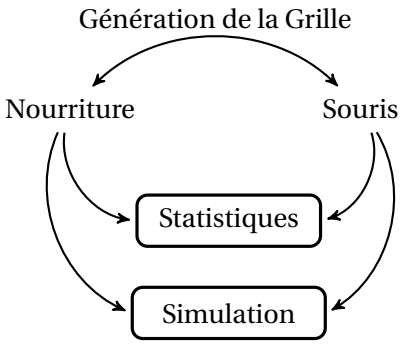
1. Nord-Est : la souris se dirige à gauche, si elle trouve un obstacle elle se dirige en haut.
2. Nord-Ouest : la souris se dirige en bas, si elle trouve un obstacle elle se dirige à gauche.
3. Sud-Est : la souris se dirige en haut, si elle trouve un obstacle elle se dirige à droite.
4. Sud-Ouest : la souris se dirige à droite, si elle trouve un obstacle elle se dirige en bas.
5. Nord : la souris se dirige en bas, si elle trouve un obstacle elle se dirige à gauche.
6. Ouest : la souris se dirige à droite, si elle trouve un obstacle elle se dirige en bas.
7. Sud : la souris se dirige en haut, si elle trouve un obstacle elle se dirige à droite.
8. Est : la souris se dirige à gauche ,si elle trouve un obstacle elle se dirige en haut.

— Communication :

1. Pour réaliser cette partie on a utilisé le design pattern "Template method", en définissant la squelette d'un algorithme qui nous sera utile pour la diffusion des informations, c'est à dire une opération en différentes étapes pour les sous-classes, ceci permet à ces dernier de redéfinir certaines étapes de cet algorithme.
2. Les souris ont une variable d'instance qui varie entre 0 et 10, il va représenter le degré de sa confiance. Si ce paramètre est > à 5 alors la souris est coopérative. Sinon elle sera égoïste.
3. Ce paramètre varie au cours de la simulation si une souris reçoit une information erronée, on décrémente.Sinon, si elle reçoit une information correcte, on l'incrémente.
4. Reproduction : pour simuler la grossesse d'une souris on lui associe un attribut qui va représenter le temps de la grossesse. Si une souris est enceinte, on décrément le temps de la grossesse, et si il atteins 0 ,a souris se duplique.

### 3.2.3 Déroulement de la simulation

— La classe "Simulation" a été créée pour gérer le moteur de jeu, elle permet de manipuler les objet qui ont une certain action à réaliser (Nourriture, Souris ... etc). Pour cela, l'ensemble de ces objet est stocké dans des ArrayList, pour faciliter leur manipulation.
— Au commencement, la simulation est instanciée et la génération de la grille est lancée. La boucle de simulation se lance alors, le compteur de tours est incrémenté.
— notre moteur de jeu consiste à parcourir ces ArrayList est associée à chaque objet une action a effectuée.
— Les souris agissent une par une selon leurs environnement et leurs comportement comme il est indiquer dans section 2.
— Les souris venant de mourir quittent la grille en changeant de direction vers le haut jusqu'à ce qu'elles dépassent le mur, puis on la supprime de la grille. Parcontre les souris venant de naître sont ajoutées à la grille avec une taille plus petite que les autres.
— Pour générer les sources de nourriture on décrémente le compteur de génération de la nourriture. Si le compteur vaut zéro, des sources de nourriture plus ou moins importantes apparaissent sur la grille en fonction du tour de jeu. Le compteur revient à la valeur initiale qui a été sélectionnée par l'utilisateur au début voir Figure 13', une fois la source de nourriture est totalement consommée par les souris ou bien si son temps est expiré la source sera supprimée de la grille.
— Les valeurs statistiques sont chargées puis on boucle afin de revenir à l'incrémentation du nombre de tours.




*Déroulement de la simulation et statistiques*

### 3.2.4 Statistiques

Pour stocker toutes les statistiques de la simulation, on a créé une classe "Statistics" ayant une seule instance : design pattern singleton. Les valeurs statistiques sont stockées dans les variables de cette instance, elles sont mises à jours à chaque tour de jeu. La visualisation de ces statistiques est effectuée à l'aide de la librairie "JFreeChart".

### 3.2.5 Système de journal

Ce système permet à une souris de raconter sa vie de la naissance jusqu'à la mort.Il est réalisé en plusieurs étapes, sous forme graphique avec des animations.

**Enregistrement des événements :** On a créé une classe "MouseEvent" ayant uniquement 2 attributs de type String, l'un contient une description de l'événement : (ex : à quel tour de jeu,dans quelle case,quelle souris...etc), l'autre contient le nom de l'image associée à l'événement (exp : "walk.png","eat.png"...etc). Dans la mémoire de chaque souris, une "ArrayList<MouseEvent>" est mise en place. À chaque action effectuée par la souris, un objet de type "MouseEvent" est créé puis stocké dans cette "ArrayList<MouseEvent>".

**Réalisation de l'animation et synchronisation :** En implémentant l'interface "Runnable", On a créé deux "Threads" différents. Le premier, parcours lentement une "ArrayList<MouseEvent>", et à chaque itération, il indique au deuxième "Thread" le texte et les images correspondantes à l'événement en question, qui doivent alors être affichées, puis il se met en pause pendant un certain temps. Pendant ce temps, le deuxième "Thread" dessine les images dans un "JPanel", avec une grande vitesse en changeant leurs coordonnées d'une manière très précise afin d'avoir une animation cohérente.

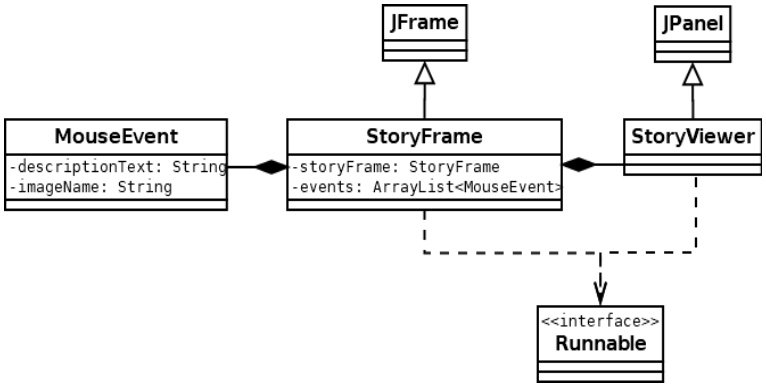



FIGURE 11 – UML-Système de journal

## 3.3 Fonctionnalités supplémentaires

- Choix de l'environnement d'évolution des souris : désert, neige, forêt.
- Affichage de l'historique de la simulation en temps réel.
- Modification manuelle des cases de la grille.
- Prise de contrôle d'une souris.
- Affichage de champ de vision d'une souris.
- Utilisation des testes unitaires automatisés (JUnit) ainsi qu'un système de log (Log4j).

# 4 Manuel Utilisateur

**Chapeau** Cette section est dédiée au manuel utilisateur.

## 4.1 Introduction au jeu


FIGURE 12 – Fenêtre d'introduction

1. Permet de lancer la fenêtre de parametrage de jeux.
2. Permet d'afficher des informations à propos de jeu (langage de programmation utilisé,environnement de développement,version...etc)
3. Permet d'afficher des informations à propos des concepteurs de logiciel.

## 4.2 Paramétrage de jeu

Cette fenêtre permet à l'utilisateur d'initialiser les paramètres de jeu.


FIGURE 13 – Fenêtre de paramétrage

1. Permet de choisir l'environnement d'évolution des souris : désert,neige ou forêt.
2. Permet d'entrer la fréquence d'apparition des sources de nourriture.
3. Permet d'entrer la fréquence des obstacles.
4. Permet de saisir le nombre initiale de souris.
5. Permet de lancer la simulation.
6. Permet de revenir à la fenêtre précédente.

## 4.3 Fenêtre principale

1. Panel permettant d'afficher la mappe pour voir l'évolution des souris.
2. Panel permettant d'afficher les statistiques de la simulation
3. Panel permettant de contrôler le déroulement de la simulation et d'accéder à des informations précises sur la grille, les souris...etc
4. Permet de mettre en pause ou de reprendre la simulation.
5. Permet de passer au tour de jeu suivant.
6. Permet de revenir à la fenêtre d'accueil.

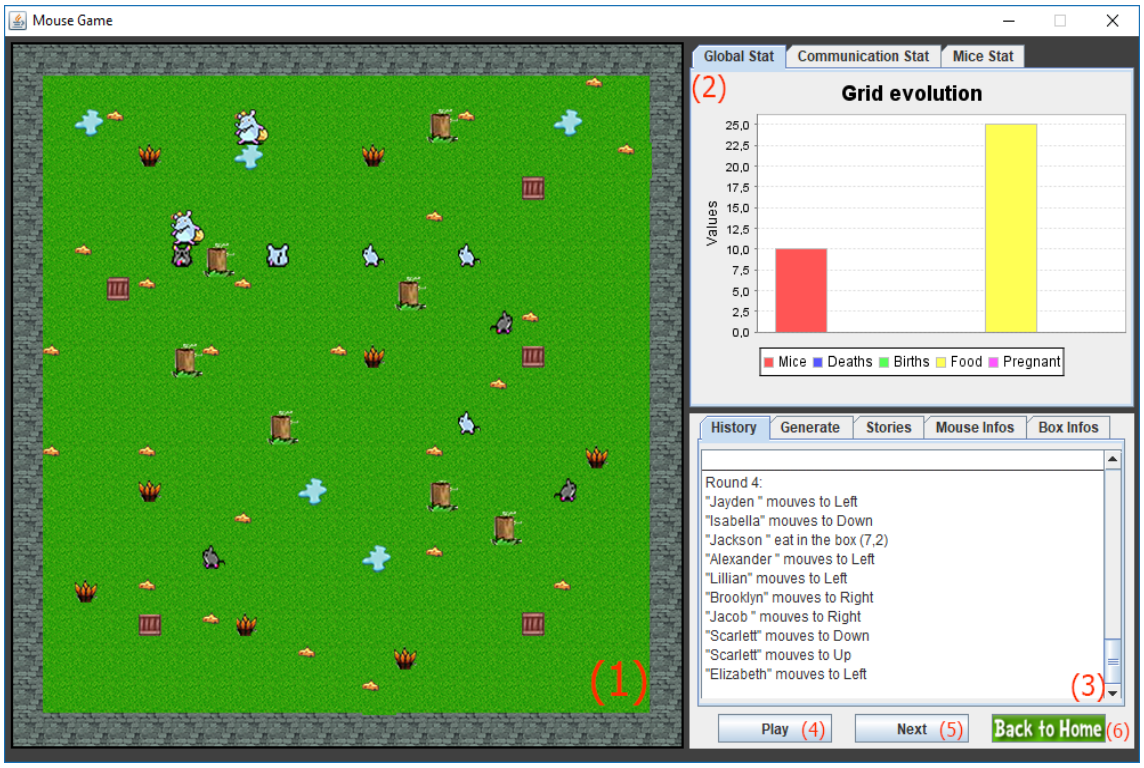



FIGURE 14 – Vu d'ensemble

## 4.4 Panneaux d'informations et de contrôle

Panel permettant d'afficher en détails tous les événements de chaque tour de jeux.

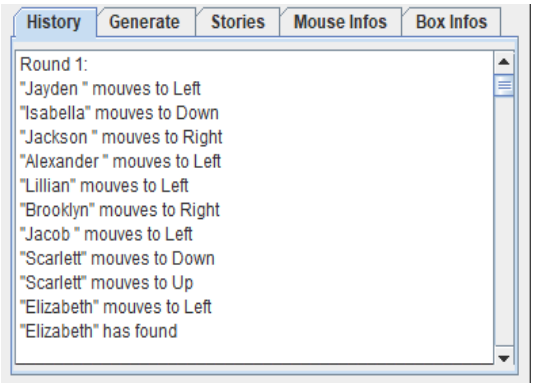



FIGURE 15 – Historique de la simulation

Panel permettant d'ajouter des souris, des obstacles, et des sources de nourriture, dans la grille.

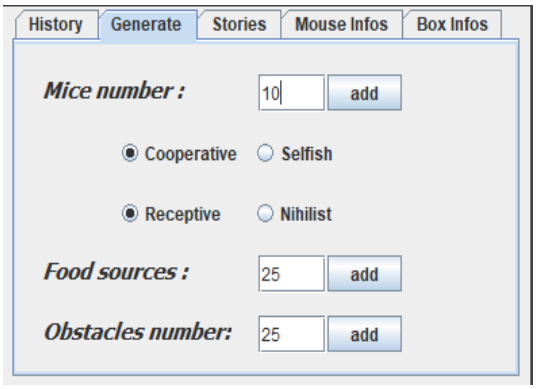



FIGURE 16 – Panneaux de génération

Panel permettant d'afficher des informations détaillées sur une souris sélectionnée dans la grille (par double clique).Il permet également de prendre contrôle de cette souris avec les touches de clavier (1), afficher son champ de vision (2), et de visualiser sa communication en affichant des petites bulles de discussion(3).

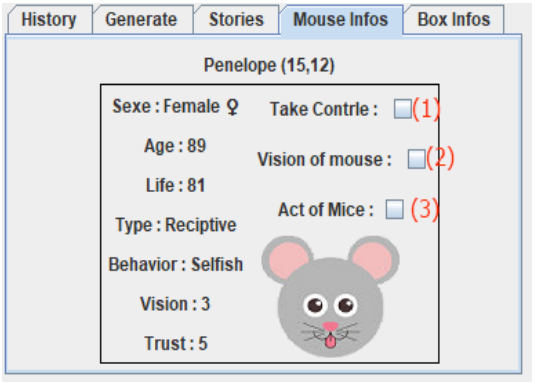



FIGURE 17 – Carte d'identité de la souris

Panel permettant de modifier une case sélectionné dans la grille (par simple clique) : changer le type de l'obstacle (1), changer la quantité d'une source de nourriture (2), et mettre à jours la case (3).

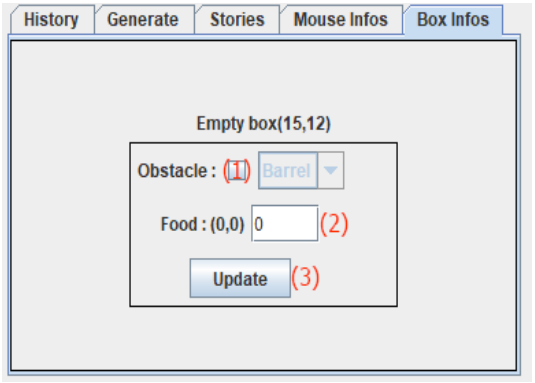



FIGURE 18 – Information sur la case

Panel permettant d'afficher des informations détaillées sur une source de nourriture sélectionnée dans la grille (par double clique)

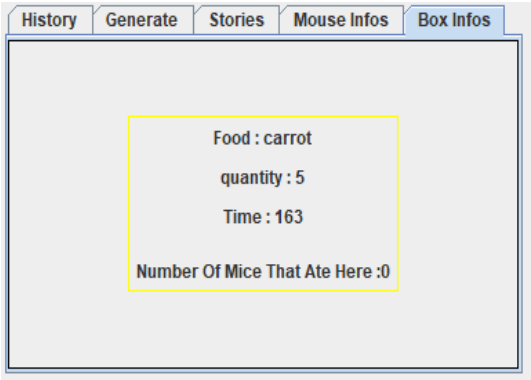



FIGURE 19 – Information sur la nourriture

## 4.5 Panneaux de statistiques

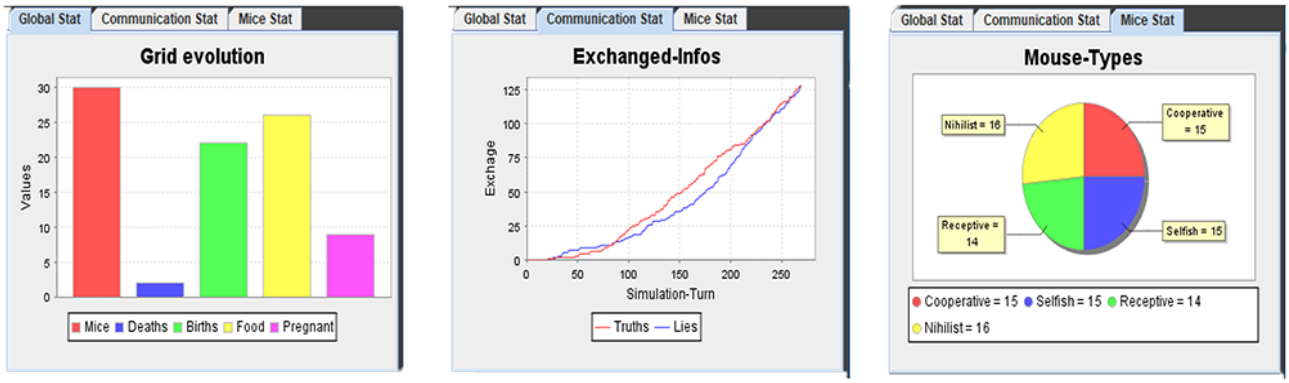



FIGURE 20 – Statistiques

## 4.6 Système de journal

Panel permettant d'afficher la liste des histoires des souris mortes. La visualisation d'une histoire sélectionnée dans cette liste, s'effectue dans une fenêtre séparée comme ce qui est illustré dans dans les figures ci-dessous.

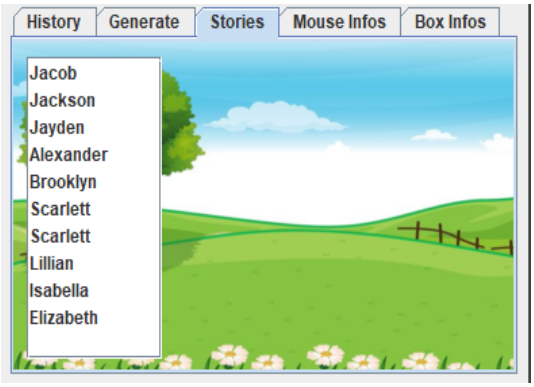


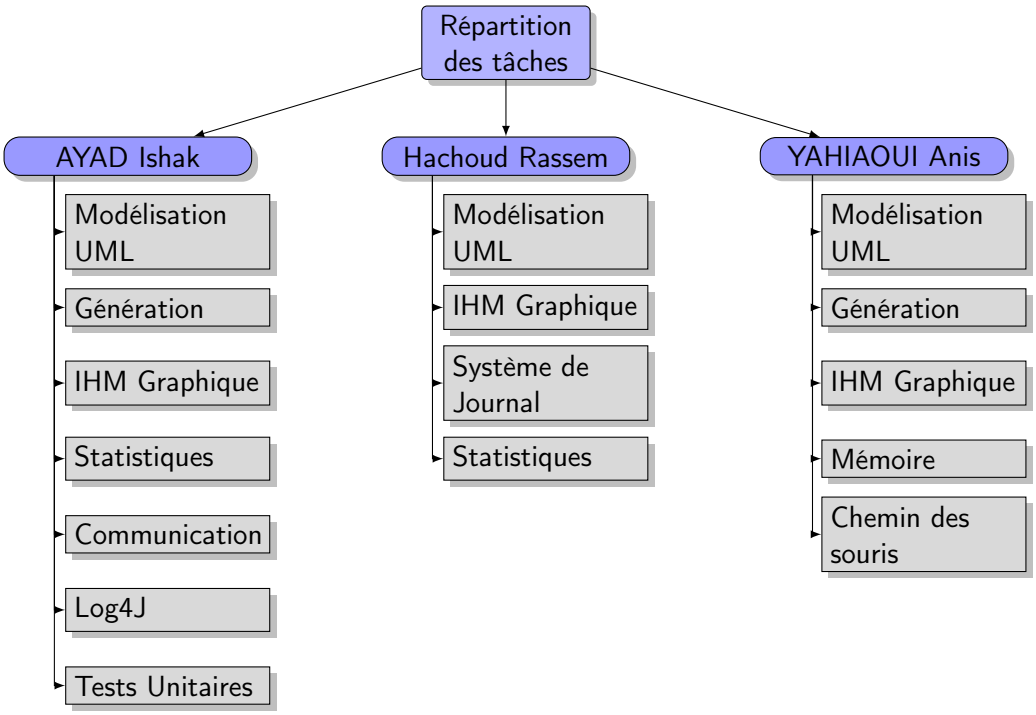

FIGURE 21 – Liste des histoires

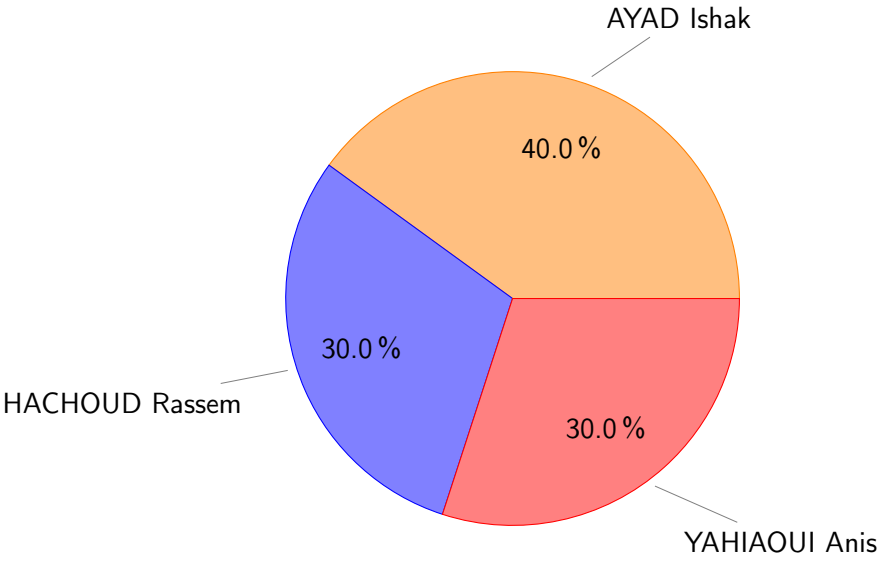

FIGURE 22 – histoire d'une souris

# 5 Déroulement du projet

**Chapeau** Dans cette section, nous décrivons comment le projet a été réalisé en équipe : la répartition des tâches, la synchronisation du travail entre membres de l'équipe, etc.

## 5.1 Répartition des tâches




## 5.2 Synchronisation du travail

— Pour la réalisation du projet nous avons utilisé le service web d'hébergement et de gestion de développement de logiciels GitHub pour synchroniser nos travaux.Ce dernier nous a permis d'effectuer des commites à un rythme élevé au commencement du projet pour que tous les membre de l'équipe aient accès aux classes de donnée ,puis après la répartition des tâches le rythme a diminué.

— Pour la rédaction de ce document nous avons utilisé la plateforme web Overleaf pour répartir les taches, le rythme de rédaction a été très élevé à la fin du projet.

## 5.3 Calendrier

| Semaine | AYAD Ishak | HACHOUD Rassem | YAHIAOUI Anis |
|---|---|---|---|
| 5 Février | Classes de donnés | Classes de donnés | Classes de donnés |
| 9 Février | Génération de la grille | Début IHM graphique | Génération de la grille |
| 15 Février | Concertation | Concertation | Concertation |
| 19 février | Point d'avancement 1 | Point d'avancement 1 | Point d'avancement 1 |
| 25 Février | Déplacement des souris | IHM | Chemin le plus court |
| 2 Mars | Statistiques | Statistiques | Vision |
| 10 Mars | IHM | Statistiques | Vision |
| 15 Mars | Concertation | Concertation | Concertation |
| 19 Mars | Point d'avancement 2 | Point d'avancement 2 | Point d'avancement 2 |
| 25 Mars | Communication | Système de journal | Mémoire des souris |
| 1 Avril | Test unitaires et log4j | Système de journal | IHM |
| 5 Avril | Concertation | Concertation | Concertation |

FIGURE 23 – Calendrier

# 6 Conclusion

Au final, nous avons donc réalisé une simulation d'une société de souris, en prenant en compte leurs différents besoins, ainsi que leurs comportements et leurs environnement. Les souris explorent régulièrement la grille pour chercher la nourriture afin de survivre, et une fois bien nourries, elles peuvent assouvir leur besoin de reproduction. De plus, quand deux souris se rencontrent, elles peuvent échanger leurs connaissances à propos des sources de nourriture, cela dépend de leurs comportements et leurs mémoires, et comme dans toutes les sociétés, nous avons mis en place différents niveaux de confiance et de fiabilité. En outre, le comportement d'une souris évolue en fonction de son environnement et les caractères de leur entourage. Enfin, chaque souris peut raconter sa vie, de la naissance jusqu'à la mort.