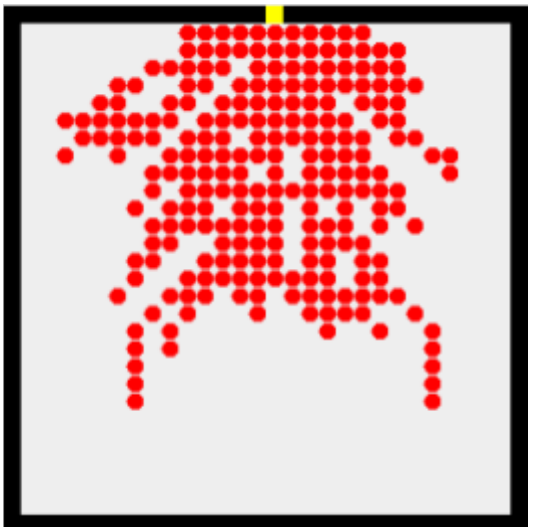


Projet JAVA
Simulation d'une évacuation de foule
sujet 1

Rapport d'analyse

ENSG Géomatique 2020 - 2021

Antoine Cornu
Axel Losco
Baptiste Rivière

Sommaire

Introduction

En février 2002, 5 jeunes personnes meurent asphyxiées au parc des expositions de Brest. Cet incident s'est produit lors d'une bousculade devant la sortie d'un festival. Ces exemples de mouvements de foule meurtriers sont nombreux et se sont multipliés après les attentats de 2015.

C'est pourquoi canaliser une foule lors de mouvements de panique ou d’évacuation dans des espaces restreints représente un enjeu majeur de santé publique. Elle permet également de fluidifier le trafic pour le confort de la foule.

L'objectif de ce projet est donc de pouvoir simuler l'évacuation d'une salle donnée afin de tester numériquement sa capacité à fluidifier une évacuation en cas de panique.

<u>Principes de la modélisation :</u> Chaque paragraphe désigne une classe

- **Espace**

Nous allons tout d'abord créer un espace essentiellement composé d'éléments fixes, possédant un couple de coordonnées immuable. Il peut s'agir :

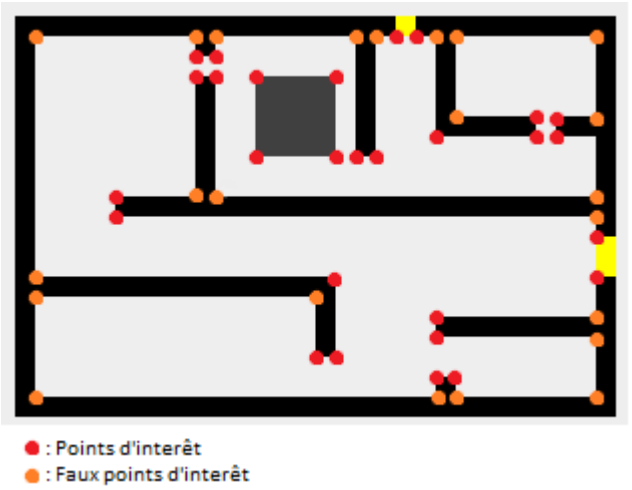
- d'obstacles : les individus ne pourront pas les traverser et devront donc les contourner. Il y aura plusieurs types d'obstacles (chaise, mur, table)
- de sorties : les individus chercheront à les atteindre pour sortir de la salle (une sortie peut être ouverte ou fermée)

- **Foule**

Dans cet espace, on disposera une foule aléatoirement ou non : chaque individu aura ses propres coordonnées à l'instant présent et un chemin, il s'agit d'une succession de coordonnées entières à parcourir pour atteindre la sortie le plus rapidement possible.

- **Graphe**

Pour planifier ce chemin, l'individu devra se repérer dans le "graphe" de l'espace et trouver le plus court chemin vers la sortie. Ce graphe a pour nœuds les points d'intérêt de l'espace, c'est-à-dire les "coins" des obstacles par lesquels les individus doivent passer. Les arcs du graphe associé à l'espace ne relient les points d'intérêts que s'ils peuvent se "voir" : c'est-à-dire s'ils sont reliés sans obstacle entre eux par une droite.
On va intégrer l'individu dans ce graphe comme source pour qu'il détermine le plus court chemin de sa position à une sortie, on utilisera l'algorithme de Dijkstra.



- **Simulation**

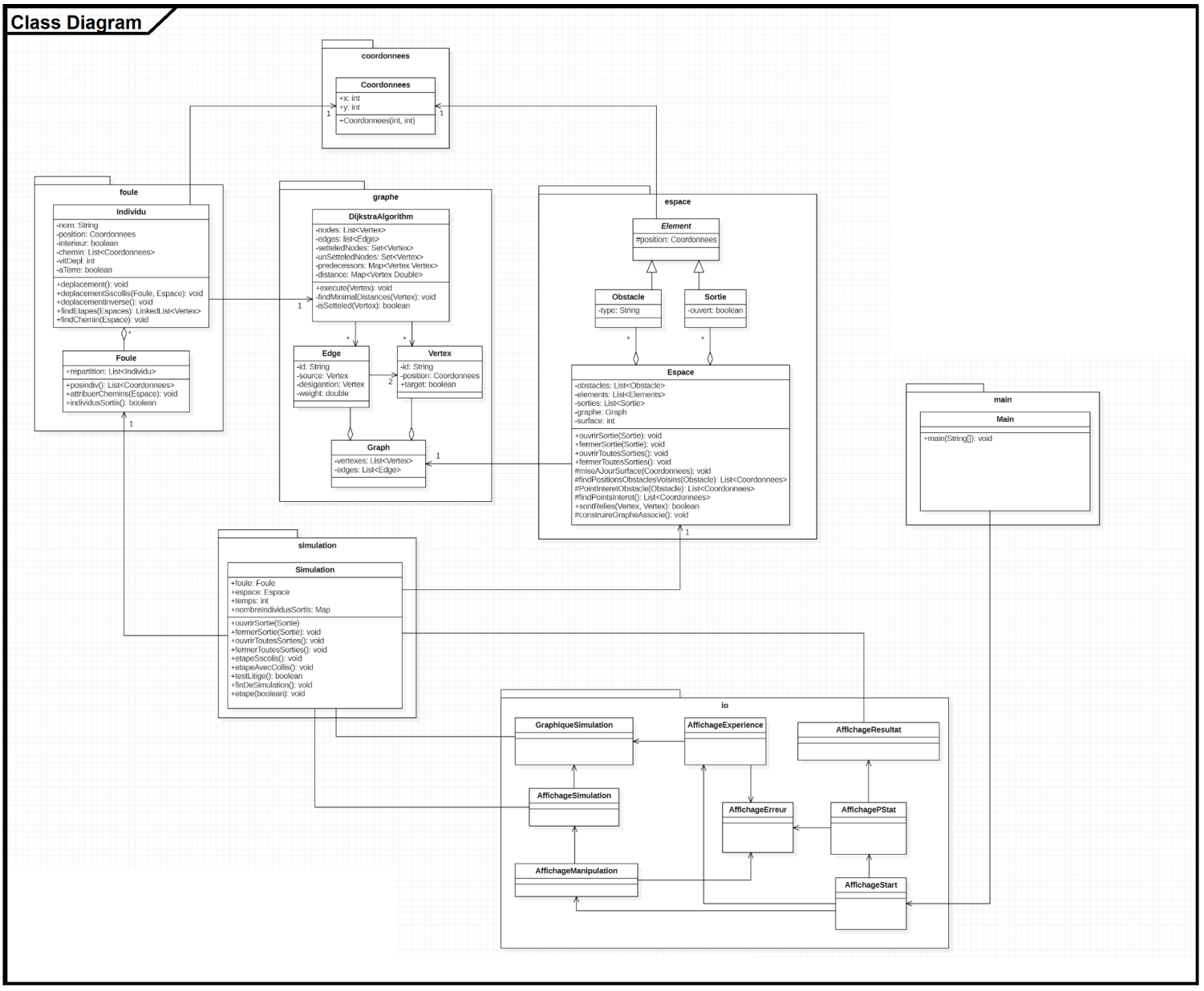
Une fois que chaque individu a planifié un chemin vers la sortie, on peut lancer la simulation:
Chaque étape est basée sur des règles simples:

- étape sans collision : Chaque individu passe à la position suivante si elle est libre, sinon il attend à sa place que la position se libère
- étape avec collision : On considère que les individus sont plus agités.
  Chaque individu tente donc d'avancer sur sa prochaine destination sans considérer ses voisins, cependant il peut y avoir un litige : deux personnes peuvent se retrouver sur la même case.
  On doit alors résoudre toutes ces collisions avant de passer à l'étape suivante, les individus ayant été repoussés sur leur ancienne position sont considérés comme "à terre".

- **Diagramme de classe**

Le diagramme de classe ci-dessous est simplifié pour plus de clarté.
On a retiré

- les constructeurs
- les redéfinitions des méthodes hashcode, equals, toString
- les méthodes add et delete
- les getters et les setters
- les classes test présents dans chaque package et la classe Test permettant de les appeler
- les classes moteur permettant de générer une foule, un espace
- les arguments et méthodes dans tout le package io (voir paragraphe suivant)



- **Fonctionnalités (interface, package io)**

On proposera 3 utilisations possibles

- **Simulation**

Il s'agit d'une simulation dans une salle déjà préparée par nos soins.
On peut y choisir * la salle désirée dans une liste déroulante
* le choix d'une simulation avec ou sans collision
* le nombre de personnes dans la simulation

- **Expérience**

Il s'agit d'une simulation avec plus de liberté dans la taille de la salle (rectangulaire)
on peut y choisir * la taille de la salle (longueur et largeur)
* le choix d'ajouter un obstacle devant la sortie
* le choix d'une simulation avec ou sans collision
* le nombre de personnes dans la simulation

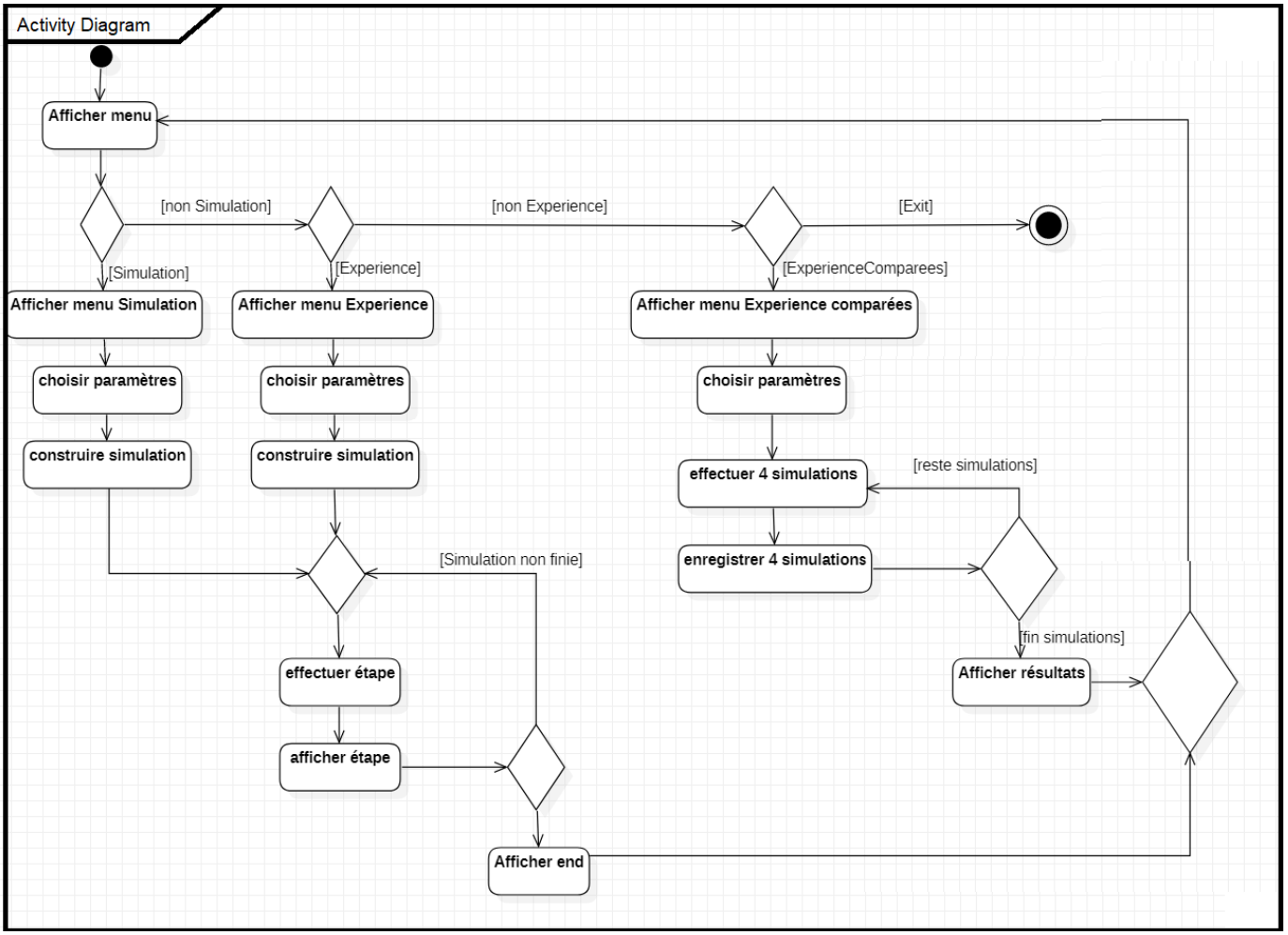
- **Expériences comparées**

Il n'y a pas d'affichage de la simulation, l'algorithme simule plusieurs évacuations de la même foule dans le même espace. On renvoie la moyenne du temps mis pour sortir avec le même nombre d'individus pour 4 cas différents : avec ou sans obstacle devant la porte, avec ou sans collision.
On peut y choisir * la taille de la salle (longueur et largeur)
* le nombre de personnes dans la simulation
* le nombre de simulations

- **Lors d'une simulation (avec affichage) :**

La salle s'affiche (il peut être nécessaire d'agrandir la fenêtre)
On lance la simulation en cliquant sur "Play" (le bouton du bas)
On peut également la mettre en pause en cliquant sur "Stop"
Une fois la simulation finie, ce bouton devient "End"
On ferme la fenêtre en cliquant dessus.
On aura également la possibilité de fermer et ouvrir les sorties et on aura une légende

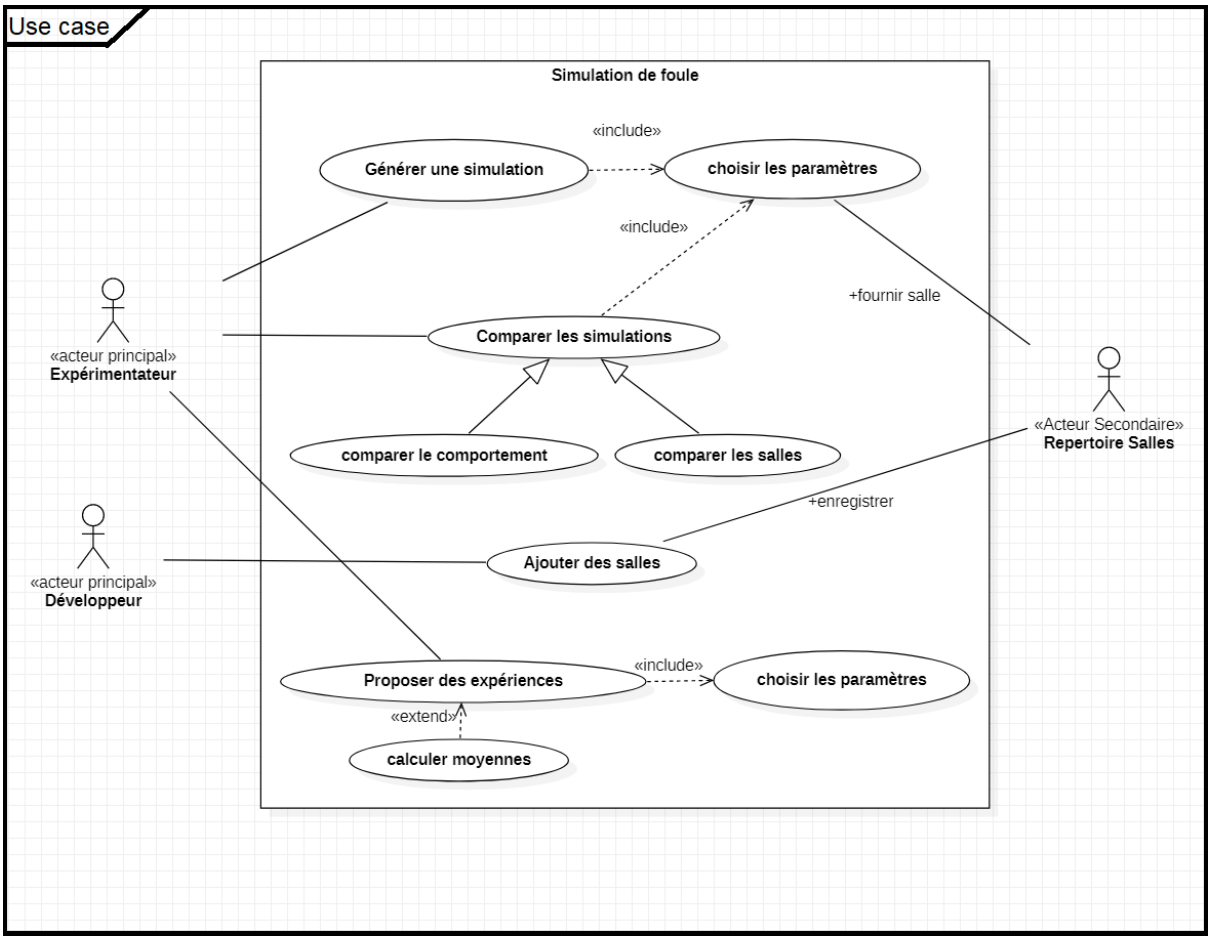
Ci dessous le diagramme d’activité du programme



- **Qui utilisera le logiciel ? pour quoi faire ?**

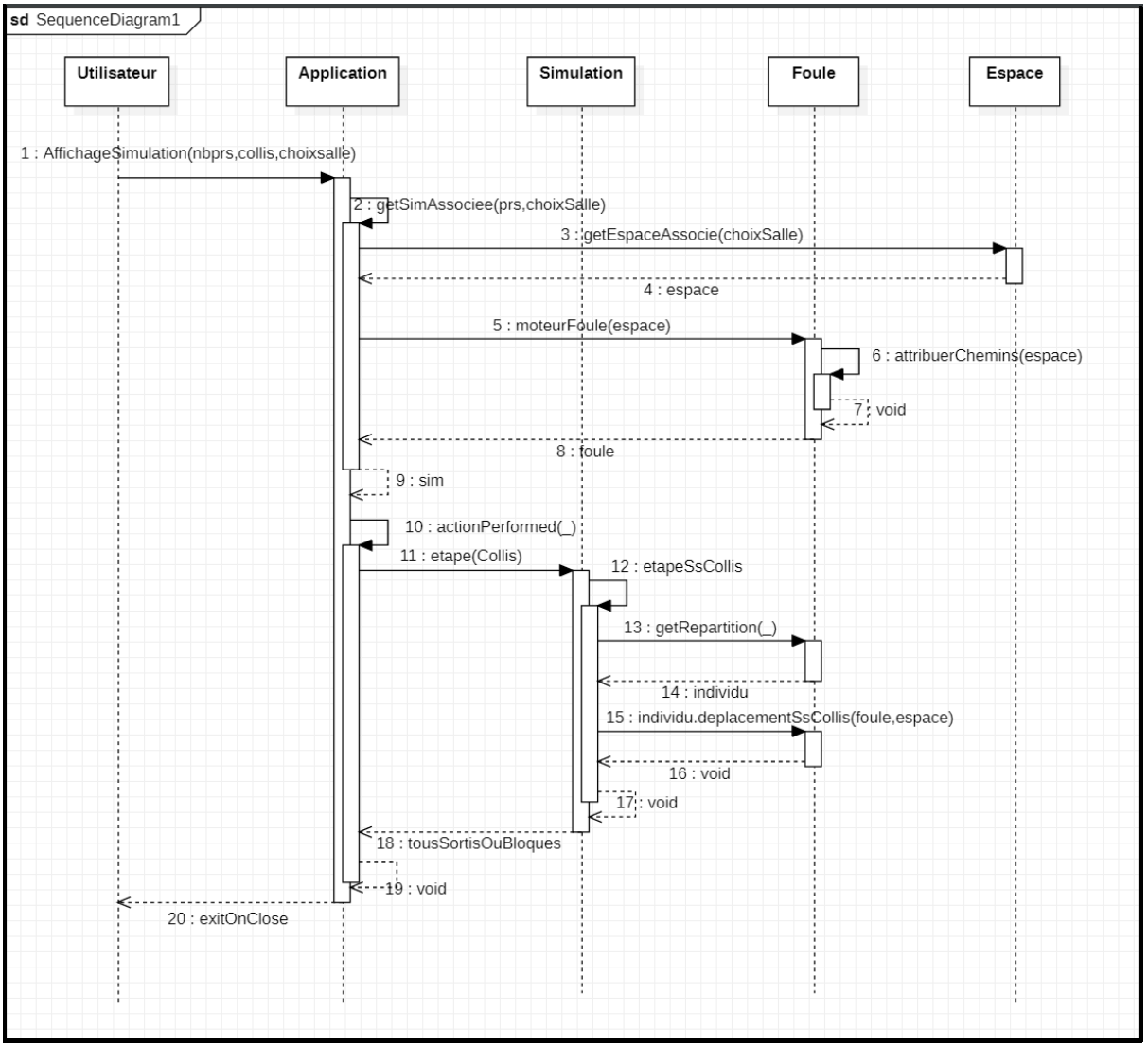
Ce logiciel est destiné aux chercheurs, architectes, urbanistes, organisateurs d'événements ou toute personne souhaitant tester la capacité d'une salle à être évacuée et quelles seraient les solutions à mettre en œuvre pour pouvoir fluidifier le flux de la foule vers la sortie.

Ci dessous le diagramme de cas d'utilisation du programme



Déroulement des actions et informations utilisées

Ci dessous le diagramme de séquence simplifié (par packages)
lors de la simulation d'une étape sans collision où l'utilisateur choisit la salle, le nombre de personnes et le type de la simulation (sans collision)



Hypothèses :

Pour ces hypothèses nous nous sommes principalement appuyés sur la vidéo “Le dilemme de l’évacuation” de Mehdi Moussaïd, chercheur à l’institut Max Planck de Berlin.

L’hypothèse majeure dans notre simulation est la discrétisation des coordonnées des objets : On considère l’espace comme une grille.
Bien qu’elle nous simplifie le code, cette hypothèse demande beaucoup d’efforts pour se rapprocher d’un comportement réaliste des individus sans se baser sur les règles de la physique. Voici les hypothèses qui en découlent :

- Une collision coûte 1/4 d’unité de temps
  En effet selon Mehdi Moussaïd une foule doit “se hâter avec lenteur” pour sortir au plus vite. Le chercheur explique qu’une foule en état de stress (avec collisions) met 25% de temps en plus qu’une foule plus calme (sans collisions). Avec cet ajout de temps nous espérons nous rapprocher de ce chiffre.
- Le chemin entre l’individu et la sortie ne sera pas le plus court car il sera discrétisé
  C’est-à-dire que bien que l’algorithme de Dijkstra nous donne les étapes de passage de l’individu pour arriver au plus vite à la sortie, c'est-à-dire les angles d’obstacles, il faudra trouver le chemin case par case entre chacune de ces étapes.

Nous proposons ces autres hypothèses simplificatrices :

- Un individu a une connaissance parfaite de l’espace, il sait exactement où il est et où est la sortie la plus proche.
- Si une “file d’attente” se forme, les individus ne cherchent pas à la dépasser. Ce comportement n’est pas naturel en cas de stress intense, c’est ce que montre Mehdi Moussaïd : chaque individu sort à la bonne vitesse, mais si tout le monde “marche” le dernier individu aura intérêt à courir pour être le premier sorti. "L'intérêt de l’individu et celui du groupe sont incompatibles”.
- Chaque individu a une vitesse, corpulence et comportement similaires. Certaines personnes seraient en réalité plus ou moins courageuses, plus ou moins rapides…

Programme / Répartition des tâche:

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Analyse du sujet	Antoine Axel Baptiste	Antoine Axel Baptiste				
Mise à jour du rapport d'analyse	Axel	Axel	Axel	Axel	Axel	Baptiste
Implémentation de la structure globale du code		Antoine Axel Baptiste				
Implémentation des méthodes permettant à l'individu de planifier son chemin		Baptiste	Baptiste			
Implémentation package Simulation		Antoine	Antoine	Baptiste		
Implémentation interface graphique				Antoine Axel	Antoine	Antoine
Renforcement et déboguage du code				Baptiste		
Correction, documentation et finalisation du code					Baptiste Antoine	
Rédaction du rapport de synthèse						Antoine

Legend: Antoine Axel Baptiste

Responsable Analyse : Axel L.
Responsable code : Baptiste R.
Responsable Synthèse : Antoine C.

Sources de code :

- Nos cours de Java et UML à l'ENSG
- Le package Graphe a été en majeure partie tirée du site Vogella
  Dijkstra's shortest path algorithm in Java - Tutorial
  Ces classes ont donc été étudiées, commentées et complétées pour répondre à nos attentes
- Les différents forums d'aide sur internet (notamment StackOverflow, OpenClassroom...)
- L'apprentissage et la conception du code de l'affichage graphique (Swing) s'est fait avec différents tutoriels sur youtube