Université d'Aix-Marseille
Master 1 Informatique - Saint-Jérôme

Mai 2014
TER

Système Multi-Agents
Évacuation d'immeuble en cas d'urgence

Mehdi BOUAZIZ - Emir HRUSTIC - Mohammed-Lamine GUILIANO

Table des matières

1 Introduction

1.1 Contexte

En France, un incendie se déclare en moyenne toutes les deux minutes. Or, une flamme peut devenir un incendie non maîtrisable en quatre minutes. Les bâtiments doivent donc être conçus de manière à pouvoir être évacués le plus rapidement possible, conformément à ce que la loi stipule depuis un arrêté du 25 juin 1980.

Article R.123-4 du CCH (Code de la Construction et de l'Habitation) : « les bâtiments et les locaux où sont installés les établissements recevant du public doivent être construits de manière à permettre l'évacuation rapide et en bon ordre de la totalité des occupants... »

L'objectif de notre étude est de vérifier si le bâtiment de Polytech' Marseille a bien été construit suivant ces critères de sécurité.

1.2 Angles d'approche

Plusieurs paramètres entrent en jeu lors d'une évacuation :
- l'architecture du bâtiment
- le nombre d'occupants
- la qualité physique des occupants (la population d'une école n'est pas la même que celle d'une maison de retraite)
- le comportement des occupants (disciplinés ou paniqués)

Dans notre cas, l'architecture du bâtiment est celle de Polytech' Marseille. Nous utiliserons donc les plans existants.

Le nombre d'occupants est variable. Il dépend de l'emploi du temps des étudiants, des enseignants, ainsi que des évènements particuliers qui rythment l'année.

Au sein de l'école, nous retrouvons une majorité d'étudiants, puis une part d'enseignants et de personnels administratifs. La qualité physique des étudiants est en moyenne supérieure à celle des autres occupants, en raison de la différence d'âge.

Pour finir, lors d'une évacuation, les personnes réagissent de différentes manières. Certaines sont très calmes, alors que d'autres sont totalement paniquées.

Tous ces éléments constituent des angles d'approche différents pour tester l'efficacité du système d'évacuation de l'école.

2 Description du Sujet

Ce TER aborde la problématique de la prévention des risques, au travers de la définition de stratégies de conception de bâtiments au vu de leur capacité d'évacuation en cas d'incendie ou autre accident mettant en danger la vie et la santé de ses occupants.

Une des approches possibles consiste en la mise en ?uvre de simulations permettant de vérifier/valider les hypothèses faites lors de la conception d'un bâtiment (voire une fois construit). Il s'agit alors d'expérimenter des stratégies d'évacuation sur des modèles informatiques, en reproduisant l'environnement et les comportements des individus présents dans les bâtiments.

Les Systèmes multi-agents sont particulièrement adaptés pour simuler des processus distribués [Guessoum et al. 2012], [Drogoul et al. 2003], [Janssen 2012].

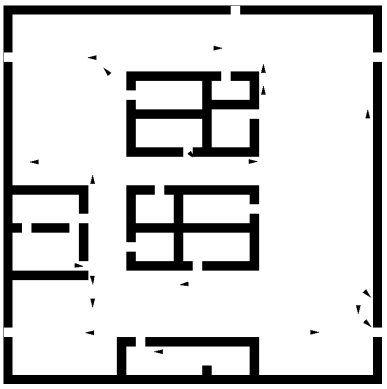
2.1 Objectif

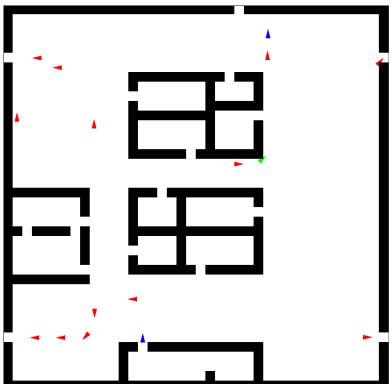
L'objectif de l'étude est d'exploiter un environnement multi-agents afin de modéliser l'évacuation d'un immeuble en cas d'urgence. Cet objectif est décrit en quatre points :

- Modéliser un bâtiment et les voies de circulation (celui du département Génie Industriel Informatique de Polytech' Marseille en l'occurrence).
- Définir une ou plusieurs stratégies d'évacuation (mode raisonné, panique, blocage de voie de circulation, etc).
- Modéliser les comportements individuels (fuite aveugle, recherche d'une sortie, etc).
- Simuler ces stratégies et produire des résultats graphiques facilitant leur interprétation.

2.2 Modélisation

2.2.1 Environnement





Nous modélisons le bâtiment de Polytech' Marseille sur un étage seulement, afin de conserver un cadre en 2 dimensions plus adapté aux simulations multi-agents.

Nous représentons les murs par des carrés noirs. Les sorties sont indiquées par l'absence de mur.

Les personnes, pour les simulations standard, sont représentées par des triangles isocèles noirs dont le sommet principal indique la direction.

Lors de simulations plus évoluées, nous prenons en compte l'âge et la fonction des personnes, ce qui se traduit par des triangles de différentes couleurs :
- rouge : étudiants
- bleu : administratifs et enseignants
- vert : personnes âgées
- violet : personnes malades

2.2.2 Comportements

Chaque agent mobile représente une personne. Chaque personne possède deux caractéristiques principales :
- Vitesse : une personne va plus ou moins vite (jeunes, vieux, sportifs, etc).
- État psychologique : une personne est dans un état *raisonné* ou dans un état de *panique*.

Une personne raisonnée se dirige vers la sortie la plus proche, à sa vitesse.

Une personne paniquée agit de manière irrationnelle : elle se précipite à toute vitesse vers une sortie au hasard, en bousculant les personnes devant elle. Son état de panique est transmis aux personnes raisonnées qui entrent dans son champs d'interaction.

La panique étant un état psychologique difficilement modélisable mathématiquement, nous avons choisi de la définir selon quatre caractéristiques :

1. Sortie aléatoire (une personne paniquée perd le sens de la logique)
2. Vitesse accrue (la montée d'adrénaline améliore les capacités physiques)
3. Transmission de l'état de panique (une personne paniquée communique son stress aux autres)
4. Bousculades des voisins (une personne paniquée devient violente)

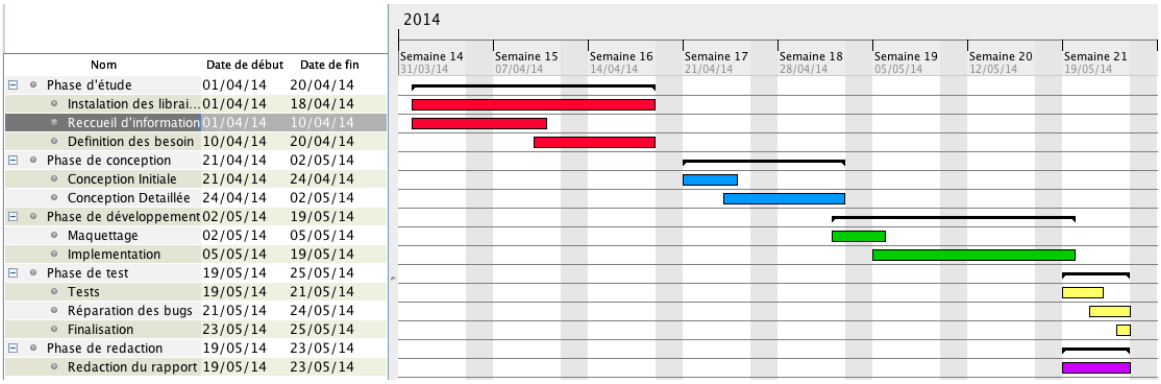
2.3 Outils logiciels

Le développement s'effectue en C++ et s'appuie sur plusieurs librairies :
- Repast : environnement de développement complet pour les systèmes multi-agents.
- Boost-MPI : distribution des calculs sur plusieurs processus.
- SDL : interface graphique.

L'ensemble du programme est compilé à l'aide d'un *Makefile*. L'exécution s'effectue soit à l'aide d'un script shell *./go.sh*, soit directement à partir d'Eclipse.

2.4 Gestion de projet

Nous avons choisi de segmenter le projet en cinq phases. Le diagramme de Gant suivant présente le travail effectif réalisé :



Nom	Date de début	Date de fin
Phase d'étude	01/04/14	20/04/14
Instalation des librai...	01/04/14	18/04/14
Reccueil d'information	01/04/14	10/04/14
Definition des besoin	10/04/14	20/04/14
Phase de conception	21/04/14	02/05/14
Conception Initiale	21/04/14	24/04/14
Conception Detaillée	24/04/14	02/05/14
Phase de développement	02/05/14	19/05/14
Maquettage	02/05/14	05/05/14
Implementation	05/05/14	19/05/14
Phase de test	19/05/14	25/05/14
Tests	19/05/14	21/05/14
Réparation des bugs	21/05/14	24/05/14
Finalisation	23/05/14	25/05/14
Phase de redaction	19/05/14	23/05/14
Redaction du rapport	19/05/14	23/05/14

1/ Phase d'Etude

Au début du projet, nous nous sommes plongés dans les cours sur les systèmes multi-agents que M. Tranvouez nous avait donnés. Puis nous avons rapidement commencé l'installation des librairies Repast (environnement agent) et Boost-Mpi (systèmes distribués), comme requis par le guide Repast. Nous avons mis un certain temps à compiler Repast, en raison de bugs internes à la librairie. La librairie ne fournissant pas de sortie graphique, nous avons fait le choix d'installer aussi SDL dans le but d'offrir un aperçu dynamique des simulations.

Un exemple d'application des systèmes multi-agents est fourni. Il est basé sur la contamination d'une population saine par des zombies. Deux types d'agents sont donc en contact : les humains et les zombies. Nous nous sommes inspirés de cet exemple pour imaginer notre approche orientée agent sur l'évacuation d'immeuble.

En effet, lors d'une évacuation, des personnes perdent leur calme et tendent à communiquer leur stress à leur entourage proche. Elles agissent alors comme les zombies de l'exemple Repast.

2/ Phase de Conception

Nous prenons deux semaines pour concevoir en détail notre modèle. Il sera composé d'agents possédants différentes caractéristiques qui agiront soit de manière raisonnée, soit de manière paniquée. Nous aurons donc besoin d'implémenter deux comportements distincts, dont l'un déteint sur l'autre.

Au niveau de l'environnement graphique, nous prenons les plans du bâtiment de Polytech' Marseille afin de représenter notre fenêtre de simulation le plus fidèlement possible.

Pour finir, Repast est conçue pour les systèmes distribués et il nous est impossible de faire tourner la librairie sur un seul processus. Nous divisons le plan en 4 cadrans dont chacun s'appuie sur son propre processus.

Nous faisons le choix d'utiliser des fichiers de configuration pour lancer nos simulations. La librairie utilisée est assez lourde et il nous apparaît donc indispensable de faciliter la prise en main de notre programme par des utilisateurs non-initiés. Une matrice représentant l'état de l'immeuble avant l'évacuation sera prise en entrée et un fichier de configuration supplémentaire permettra de modifier les autres paramètres de la simulation (option "âge" activée ou non, temps de la simulation).

3/ Phase de Développement

Étant donné que nous connaissions C++ mais que nous appréhendions encore mal les différentes librairies installées, nous heurtions notre phase de développement à un facteur risque non-négligeable. C'est pourquoi nous nous sommes orientés vers une implémentation incrémentale.

Nous avons commencé par générer une fenêtre simple avec quatre agents statiques. Puis nous avons fait bouger les agents dans leurs processus (cadrans) respectifs et ensuite entre les différents processus. Nous avons ajouté les murs et une sortie et nous avons appris à nos agents comment sortir de la salle. Nous avons peu à peu augmenté le nombre d'agents et complexifié l'architecture du bâtiment. Pour finir, nous avons implémenté différentes caractéristiques physiques pour nos agents.

4/ Phase de Test

Des tests ont été conduit tout au long du projet, mais c'est véritablement à la fin de la phase de développement que nous avons concentré nos efforts sur le traitement des cas particuliers résiduels qui généraient des *SegFault*. Nous avons terminé en traitant les fuites mémoires du programme.

Une fois que le programme a été nettoyé, nous avons lancé nos simulations afin de générer des résultats graphiques sur notre étude.

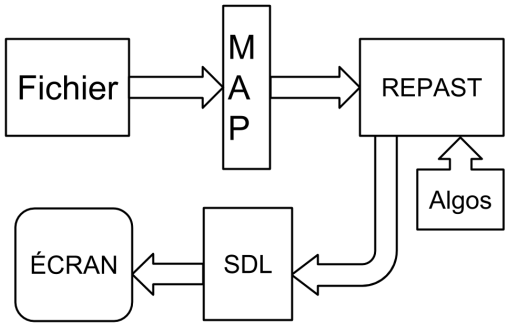
5/ Phase de Rédaction

Cette phase s'est articulée autour de la rédaction du rapport et de la mise au propre du code. Des commentaires et un *ReadMe* ont été ajoutés.

3 Travail Réalisé

Dans cette partie, nous détaillons l'implémentation de notre programme.

3.1 Organisation du programme



Le programme prend en entrée une matrice (fichier) décrivant la configuration initiale de la simulation. Cette matrice est alors parsée dans une *map* en mémoire. Nous parcourons alors cette map pour transmettre leur position initiale à nos agents Repast.

Puis nous appliquons nos algorithmes de recherche d'une sortie à nos agents. Nous affichons leurs déplacements à l'aide d'une interface graphique SDL. Les déplacements sont rafraichis toutes les 40ms, ce qui correspond à une fréquence de 25Hz (fréquence de perception de l'oeil humain).

3.2 Structures et classes

3.2.1 Fichier d'entrée

Ce fichier comporte une matrice qui décrit l'état de la salle à l'état $t = 0$:
- 0 : une case vide
- 1 : un mur
- 2 : un agent (un occupant du bâtiment)
- 3 : une sortie

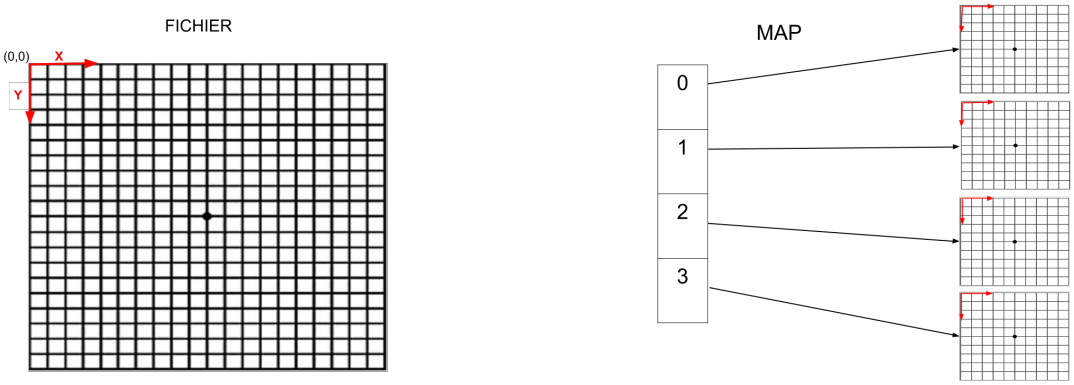
Chaque chiffre représente une case de 20x20 pixels sur la sortie graphique.

3.2.2 Map

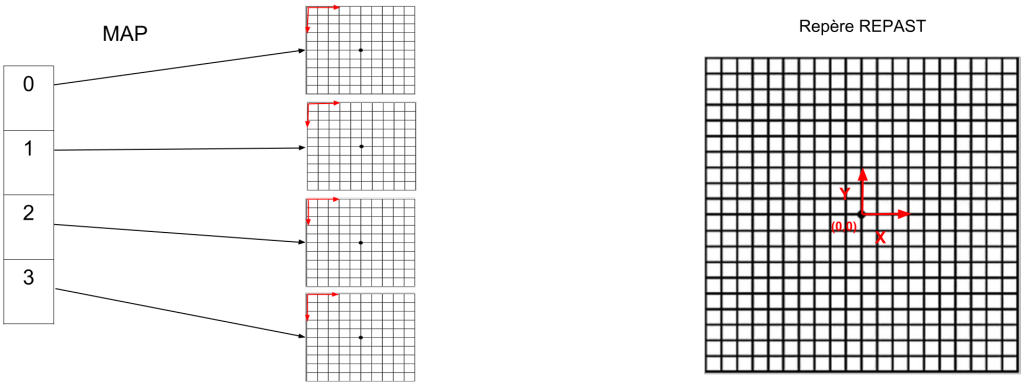
La map sert d'interface entre le fichier d'entrée et Repast. Nous avons essayé de nous en défaire, mais son utilisation s'avère indispensable. En effet, il nous a été impossible de parser la matrice d'entrée directement dans Repast. Nous utilisons 4 processus pour traiter notre programme. Lors du parsing, les 4 processus lisent le fichier en même temps et mettent à jour les données Repast en même temps. Cela a conduit à un cumul de bugs ingérables.

L'emploi d'une map pour faire le lien entre le fichier d'entrée et Repast nous a donc permis de mettre un terme à ces différents soucis.

Toutefois, il nous a fallu effectuer deux changements de référentiel supplémentaires. Dans la map, nous découpons la matrice initiale en 4 cadrans qui correspondent aux 4 processus que notre programme utilise, ce qui nous vaut un premier changement de référentiel :



Chaque processus traite sa propre portion de la map pour mettre à jour la position de ses agents dans l'espace Repast. Le dernier changement de référentiel intervient alors :



3.2.3 Agents Repast

Nos agents héritent de la classe *Turtle* (une entité de Repast/Relogo). Les *Turtles* elles-mêmes héritent de la classe Repast.

Repast fournit les attributs de base des agents (position dans l'espace, identifiant, processus de création, etc), ainsi que les différents éléments nécessaires à leur communication au sein du même processus ou entre les différents processus.

Relogo fournit des agents plus spécifiques qui sont notamment capables de se déplacer. Des méthodes sont déjà implémentées pour gérer les mouvements. Ainsi, les *Turtles* sont des agents munis de ces fonctions d'orientation et de déplacement, en plus de toutes les caractéristiques des agents Repast.

Nos agents sont donc des humains qui descendent de la tortue, avec quelques attributs de classe supplémentaires :

- typePersonne : vitesse et couleur en fonction de la vitesse
- angle : nous avons redéfini le calcul des angles car Relogo gère les déplacements dans un espace toroidal, ce qui n'est pas notre cas

– cheminX : permet de trouver l'abscisse de la sortie
– cheminY : permet de trouver l'ordonnée de la sortie
– etapeChemin : étape à laquelle il se trouve, entre sa position initiale et la sortie
– sorti : bouléen qui indique si l'agent est sorti ou non
– etatPsychologique : raisonné ou paniqué

L'état psychologique n'a malheureusement pas pu être implémenté en raison d'un manque de temps. Le travail requis aurait été facilement géré mais nous aurions eu besoin de quarante huit heures supplémentaires pour mettre en place cet attribut et gérer les modifications comportementales au cours de la simulation. Rappelons qu'une personne paniquée contamine les personnes raisonnées qui se trouvent dans son entourage proche.

3.2.4 Sortie graphique

Pour afficher nos simulations dynamiquement, nous créons trois objets à partir de SDL :
– cadran : fenêtre blanche qui représente un quart du bâtiment
– mur : carré noir de taille 20x20 pixels
– humain : triangle isocèle noir ou de couleur centré dans un carré de 20x20 pixels

3.3 Algorithmes de sortie

D'un point de vue comportemental, une personne qui connaît les lieux va évacuer l'immeuble en évaluant les différents chemins puis en choisissant le plus court d'entre-eux. Ainsi, cette personne aura tendance à se comporter comme l'algorithme de Dijkstra [voir Références].

Une personne qui ne connaît pas les lieux aura tendance à tâtonner et à se diriger vers la première sortie qu'elle trouvera, même si cette dernière n'est pas la plus courte. L'essentiel étant de trouver une sortie le plus rapidement possible. Ce comportement correspond davantage à l'algorithme A* [voir Références].

Nous avons fait le choix d'implémenter l'algorithme A* pour des raisons purement logiciels. En effet, dans le cadre de notre simulation, la majorité des occupants de Polytech' Marseille sont des étudiants et des enseignants ou administratifs de l'école. Ils connaissent donc le bâtiment et ils auront ainsi tendance à suivre Dijkstra.

Toutefois, les calculs nécessaires à la résolution de Dijkstra s'avéraient beaucoup trop lourds pour notre programme. Il aurait fallu que chaque agent calcul le plus court chemin à l'initialisation. Puis, sachant que des agents sont amenés à entrer en collision et à se gêner mutuellement, il aurait fallu relancer des salves de calculs à plusieurs moments, ce que le programme n'aurait pas supporter.

Par conséquent, l'algorithme A* était beaucoup plus approprié à notre environnement multi-agents pour trouver une sortie. Nous avons de plus géré les collisions entre agents en modifiant l'itinéraire d'arrivée de ces derniers, afin de les orienter vers une case proche de la sortie, tirée aléatoirement, de façon à ce que des agents de vitesses différentes ne se freinent pas les uns les autres : ils finissent par se contourner.

Techniquement, l'implémentation de ces algorithmes a nécessité l'écriture complète d'une structure de graphes sur laquelle les algorithmes s'appliquent. Les chemins de sortie calculés sont alors stockés dans des listes que Repast utilise pour diriger les agents.

4 Expérimentations

4.1 Choix du contexte

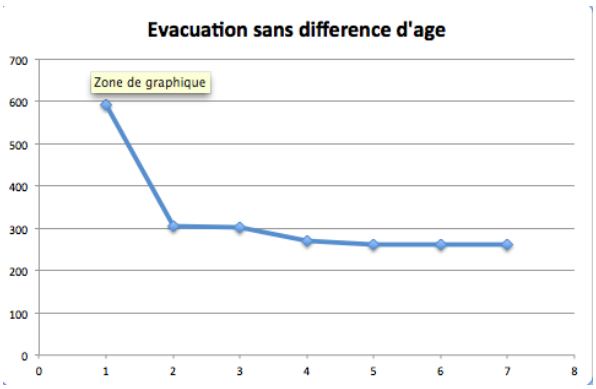
Notre objectif est de vérifier si le bâtiment de Polytech' Marseille est adapté aux évacuations d'urgence. Il s'agit donc de vérifier sa conception architecturale.

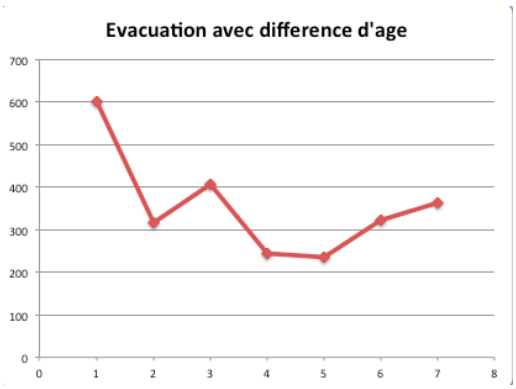
Ainsi, nous basons nos simulations sur le temps d'évacuation de l'immeuble en fonction du nombre de sorties fonctionnelles. Cela permet de tester l'efficacité du choix de conception du bâtiment, en observant si le nombre de sorties choisi est optimal ou non.

Nous nous basons donc sur des simulation de 600 tics, au cours desquels les positions initiales des agents sont toujours les mêmes. Seul le nombre de sorties change : nous commençons avec une sortie, puis nous ajoutons d'autres sorties les plus équitablement réparties dans le bâtiment.

4.2 Résultats

Nous traçons le temps d'évacuation de l'immeuble en nombre de tics en fonction du nombre de sorties de l'immeuble. Le premier graphe est simulé avec un ensemble d'agents possédants tous les mêmes caractéristiques, alors que le second graphe fait intervenir des personnes avec des vitesses différentes.





Comme nous le constatons, lors d'une évacuation avec différence d'âge, le temps d'évacuation ne se stabilise pas vraiment. En effet, les personnes d'âges différents sont générées aléatoirement, donc les simulations comportant un agent malade qui se déplace très lentement finissent toujours plus tard. Ces données, quoique se voulant plus réalistes, ne permettent pas de réelle interprétation.

Par contre, les résultats basés sur des agents de mêmes caractéristiques (pas de différence d'âge), présentent des résultats intéressants. En effet, le temps d'évacuation se stabilise à partir de 4 sorties. Ceci s'explique par le positionnement des sorties : les agents évacuent par celles se trouvant à proximité d'eux, quitte à faire un peu la queue derrière les autres.

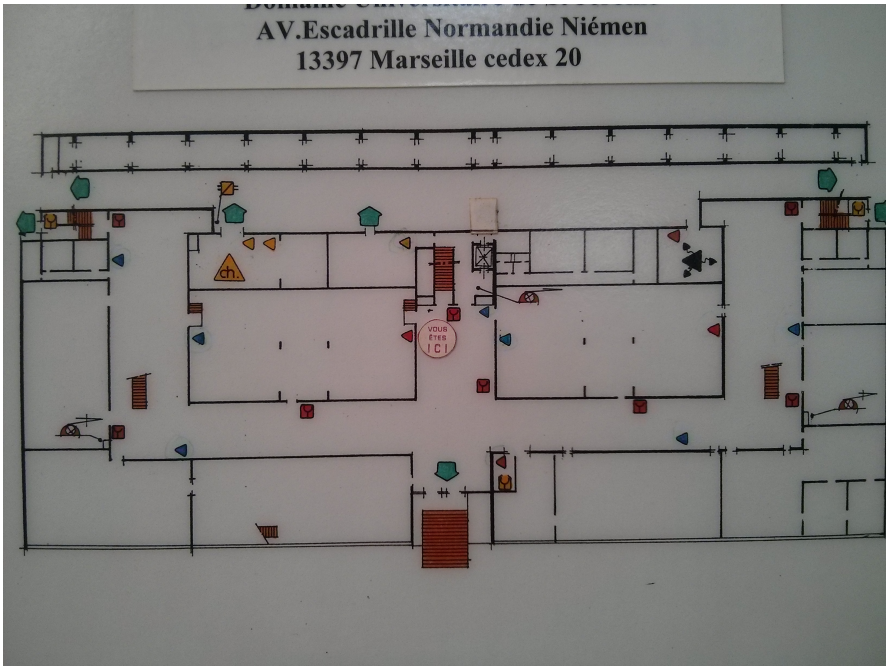


FIGURE 1 – Plan d'évacuation de Polytech' Marseille

Si nous analysons le plan de Polytech' Marseille, nous voyons qu'il dispose de cinq sorties principales : deux à l'est, deux à l'ouest et une au sud.

Notre simulation tendrait donc à valider l'efficacité de la conception du bâtiment en cas d'urgence, car le nombre de sorties est supérieur au nombre de sorties optimal que nous avons déterminé.

5 Conclusion

Cette étude nous a permis d'appréhender un environnement multi-agents pour la validation d'un choix de conception. Les agents sont particulièrement bien adaptés pour simuler les interactions humaines que l'on retrouve lors d'une évacuation d'urgence. Des émotions comme le stress et la peur se communiquent facilement d'une personne à l'autre. Le choix d'un chemin ou d'un autre est directement lié à l'état d'esprit et à la connaissance de la personne. Une personne qui connaît l'itinéraire le plus court peut le communiquer à son voisin si ce dernier ne le connait pas.

Il nous a donc fallu comprendre ces nombreuses interactions, afin de les modéliser et de les implémenter.

Même si nous n'avons pas eu le temps d'aboutir sur le développement d'un état psychologique différent pour nos agents, nous sommes parvenus à créer tout le contexte qui s'y prête et à saisir les problématiques liées aux démarches de recherche.

Avec plus de temps, nous aurions par exemple pu prendre contact avec des psychologues et des sociologues afin de mieux comprendre l'état d'esprit des occupants d'un immeuble en cas d'évacuation d'urgence. Nous aurions ainsi été capables de modéliser plus finement les interactions avant de les implémenter. La marge d'amélioration reste donc grande sur ce projet.

6 Références

6.1 Auteurs sur les Multi-Agents

[Nguyen 2009] T. T. N Nguyen, "Simulation à base d ?agents d ?évacuation de bâtiment dans les cas urgents"

[Drogoul et al. 2003] A. Drogoul, D. Vanbergue et T. Meurisse, "Simulation Orientée Agent : où sont les agents", Actes des Journées de Rochebrune, Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels, 2003

[Guessoum et al. 2012] Z. Guessoum, R. Mandiau, P. Mathieu, O. Boissier, P. Glize, M.Hamri, S. Pesty, G. Picard, J.P. Sansonnet, J.-P. Tessier et Tranvouez E., "Systèmes multiagents et Simulation" , in : GDR I3, Information, Interaction, Intelligence : le point sur le i[3], Cépaduès Editions, pp. pp. 76-120, juin 2012

[Janssen 2012] Marco A. Janssen, "Introduction to Agent-Based Modeling", 2012. http ://www.openabm.org/book/introduction-agent-based-modeling

6.2 Algorithmes de plus court chemin

[Algorithme de Dijkstra] http ://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra

[Algorithme A*] http ://fr.wikipedia.org/wiki/Algorithme_A*