Rapport de Projet
Système Multi-Agents
BuSyma

Zinedine Rebiai (rebiai_z)
Antoine Debrenne (debren_a)

June 2, 2018

Résumé

Le projet que décrira ce rapport a été réalisé dans le cadre du cours de Système Multi-Agents donné en 4ème année de l'EPITA pour la majeure SCIA en utilisant le logiciel Repast Simphony. Ce projet que nous avons réalisé à deux a pour but de comparer des stratégies pour les feux rouges dans un quartiers d'agents se déplaçant en transports ou à pied.

Contents

1 Représentation du quartier

La carte choisie est composée de deux zones d'où les humains apparaissent, et de 4 zones où ils se rendent. La route est composée de 2 croisements simples et il y a donc 6 feux de signalisation pour contrôler le traffic. Les différents agents de ce quartier seront décrits dans les parties suivantes.

Voici le visuel final de la carte :

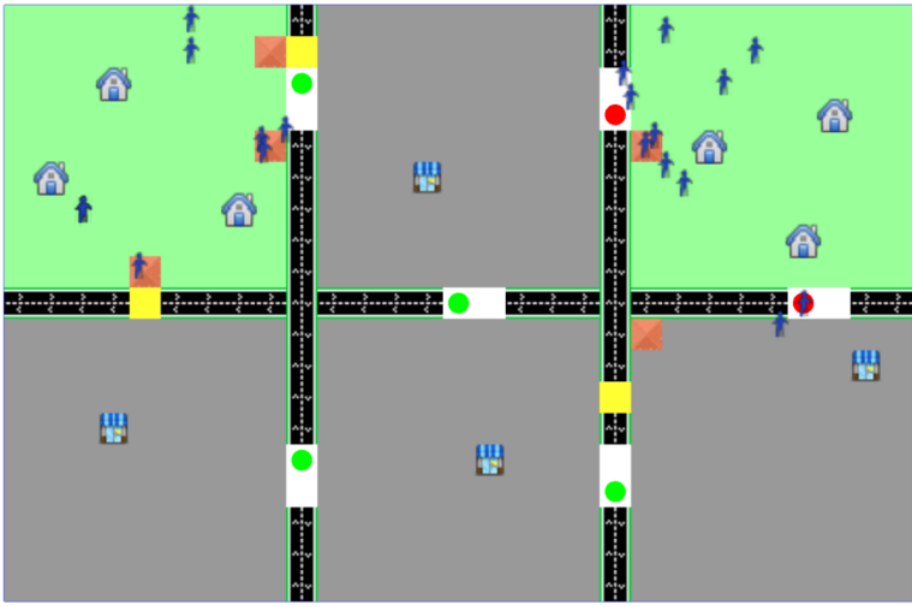

Figure 1 - Carte Finale

2 Descriptions des Agents et leurs interactions

Dans cette partie, nous commencerons par parler de la structure choisie pour le projet. Nous allons ensuite décrire le visuel choisi pour chaque agent, puis nous détaillerons les interactions entre ces derniers.

2.1 Point sur la structure du projet

Avant de décrire tous les types d'agents, il semble nécessaire de parler de l'agancement de ces derniers dans le contexte global.

Nous avons décidé de garder un contexte principal contenant tous nos agents. En effet, la plupart de nos agents actifs ont besoin d'intéragir avec beaucoup de types d'agents différents.

Dans ce contexte, nous avons ajouté deux projections :
Un espace qui est affiché à l'écran permettant d'avoir des mouvements plus réalistes, et une grille nous permettant de repérer nos agents et leur voisinage.

C'est principalement à l'aide de cette dernière qu'on va pouvoir déclancher la plupart de nos interactions entre les agents que nous allons maintenant présenter.

2.2 Agents Passifs

2.2.1 Maisons

Les maisons sont des agents passifs. Nous les avons ajoutés sur la carte pour en améliorer le visuel et la crédibilité. C'est un agent qui se trouve au milieu des parcs.

Figure 1 - Modèle d'une maison

2.2.2 Bâtiments

Les bâtiments sont égalements purement décoratifs. Nous avons imaginé qu'ils représentent les zones de travail des habitants et sont donc au milieux des zones dans lesquelles les humains se rendent.

Figure 2 - Modèle d'un bâtiment

2.3 Agents à faible interaction

Les agents décrits dans cette sections sont des agents qui modifient le comportement d'autres agents, mais qui n'ont pas de comportement particulier.

2.3.1 Passages Piétons

Le premier bloc cité ici est le passage piéton. Il est symbolisé par un bloc carré blanc, et ce sont les seules zones où les piétons peuvent traverser la route.

2.3.2 Arrêts de Bus

Les arrêts de bus ont deux fonctions principales à remplir :

- Accueuillir les humains attendant les bus.
- Forcer les bus à s'arrêter pour prendre des passagers pendant un certain nombre de ticks.

Figure 3 - Modèle d'un arrêt de bus

2.3.3 Parc

Les parcs sont les blocs dans lesquels les humains apparaissent sur la carte. Ils sont représentés par un carré d'herbe verte. Ces zones sont visibles sur les coins supérieurs de la carte présentée en section **1.2**

2.3.4 Quartier

Les quartiers sont les blocs vers lesquels les humains se rendent. Ils sont représentés par des blocs de béton gris. Ces zones sont visibles dans la partie centrale et les coins inférieurs de la carte présentée en section **1.2**

2.3.5 Route

Les blocs de routes sont les blocs séparant les différentes zones de la carte. Ils sont parsemés de passages piétons, de feux de signalisation et de bus.

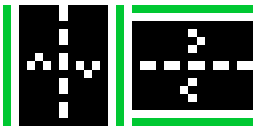

Figure 4 - Modèle d'une route

2.4 Agents à forte interaction

2.4.1 Feux de signalisation

Les feux de signalisation sont de simples ronds de couleur sur fond blanc au milieu de la route. Ces agents vont permettre de réguler le traffic routier sur la carte.

2.4.2 Bus

Concernant les bus, nous avons décidé de les représenter par un carré jaune (façon bus scolaire) afin de les voir facilement se déplacer sur la route. Leur vitesse de déplacement est assez élevée afin de rendre la simulation plus réaliste.

2.4.3 Humains

Pour les humains, nous avons utilisé un modèle personnalisé que vous pourrez voir ci-dessous. Ils vont se balader sur la carte et peuvent prendre le bus ou continuer leur chemin à pied.

Figure 5 - Modèle d'un humain

2.5 Interactions implémentées

2.5.1 Mouvements basiques

Les premières interactions mises en places ont été l'apparition des humains dans les parcs, et leur cheminement vers les quartiers industriels. Les coordonnées

d'apparition et de destination sont déterminées aléatoirement. Afin de leur permettre de se déplacer, nous avons utilisé une variante de l'algorithme Dijkstra appliqué sur notre grille.

2.5.2 Ajout de routes

Après l'ajout des agents de type Route, nous avons dû interdire les humains de les traverser hors passages piétons. Ce fût la seconde partie des interactions à implémenter. Ici, il s'agissait surtout de modifier l'algorithme de recherche de chemin.

2.5.3 Ajout des bus

Nous avons d'abord pensé à ajouter des bus qui ne s'arrêtent jamais, pour voir comment adapter nos algorithmes existants pour eux. Ils interagissaient donc uniquement avec les agents de route et nous avons mis en place un système de cycle qui fait réapparaitre un bus à son point de départ une fois sa destination atteinte.

2.5.4 Ajout des feux de signalisation

Pour commencer, nous avons du créer un modèle 2D pour les feux afin de permettre de changer dynamiquement leur couleur. Au départ, nous avons simplement décidé de faire un cycle de 6 ticks rouge/vert pour les feux.

L'ajout de ces feux a eu pour effet de modifier grandement l'algorithme de mouvement utilisé jusque là. Il faut en effet immobiliser les agents jusqu'à ce que leur tour arrive.

2.5.5 Ajout des arrêts de bus

Pour cette partie, il a fallu faire un choix de modélisation. Nous avons décidé que la moitié de nos humains prendraient le bus et l'autre moitié se rendrait à sa destination à pied. La distribution suit en vérité une loi aléatoire uniforme.

Il a fallu après ajout des arrêts de bus, forcer l'arrêt pendant un certain temps pour chaque bus, et ensuite modifier la destination des humains vers cet arrêt. Nous considérons l'humain comme arrivé à sa destination lorsqu'il entre dans le bus. On peut imaginer par exemple, qu'il voulait sortir du quartier représenté.

3 Stratégies pour les feux

3.1 Temps fixe

L'idée de cette stratégie est de faire alterner les feux rouges et verts avec une durée fixe. La durée optimale semble être autour de 4 ou 5 ticks.

3.2 Dépendant du nombre de piétons/bus

La deuxième stratégie envisagée était de comparer le nombre de piétons attendant de passer, et le nombre de passagers dans les bus attendant au même feu. Ainsi, la priorité irait aux plus nombreux.

3.3 Couleur fixe

La stratégie suivante est basée sur l'idée que la couleur du feu ne devrait pas changer souvent. Ainsi, nous imposons aux feux de rester rouges pour que les piétons puissent passer. Lorsqu'un bus arrive, les feux passent au vert le temps du passage du bus et se remettent au rouge aussitot le bus passé.

3.4 Méthodes 1 et 2

Cette stratégie consiste tout simplement à choisir sur quels feux on applique la première ou la deuxième méthode. Dans les croisements près des arrêts de bus, on privilégie la seconde, puisque les bus transporteront plus de mondes, et dans les zones moins denses, on préfère la première méthode.

3.5 Méthodes 1 2 et 3

Similairement à la stratégie précédente, nous avons choisi à quels endroits il était judicieux d'appliquer nos méthodes. Ainsi, les 2 feux du haut sont à temps fixe, les 2 feux du bas de la carte sont à couleur fixe et les 2 autres sont dynamiques.

3.6 Résultats obtenus

Nous avons ici réalisé un tableau représentant le nombre d'humains arrivés à leur destination, en fonction du temps et de la méthode choisie :

Méthode / Nombre de ticks	1000	5000	10000	50000
Méthode 1	580	2900	5800	30000
Méthode 2	590	3100	6150	30000
Méthode 3	600	3000	6000	30300
Méthode 1 + 2	570	3000	5900	30000
Méthode 1 + 2 + 3	630	3150	6200	30500

Il est clair ici que la dernière méthode semble plus rentable, la méthode 3 a été appliqués aux endroits peu fréquentés par les piétons, la 2 aux grands carrefours et la 1 dans les autres zones.

4 Bonus réalisés

4.1 Parseur

Afin de pouvoir faire des tests rapidements nous avons pris la décision de créer un parseur de map. L'idée est simple, créer un fichier texte rapidement et avoir un visuel directement sur Repast. Le fichier d'origine est plutôt simple à faire, chaque caractère va correspondre à un type de plateforme :

- 'A' - Abris de bus (Bus Shelter)
- 'C' - Passage piéton (Crossing)
- 'S' - Spawn
- 'T' - Feux de route (Traffic Light)
- 'V' - Route verticale
- 'H' - Route horizontale
- 'X' - SideWalk
- '1' - Feux de route appliquant la méthode 1
- '2' - Feux de route appliquant la méthode 2
- '3' - Feux de route appliquant la méthode 3

Par exemple, la carte finale en version texte ressemble à celle ci

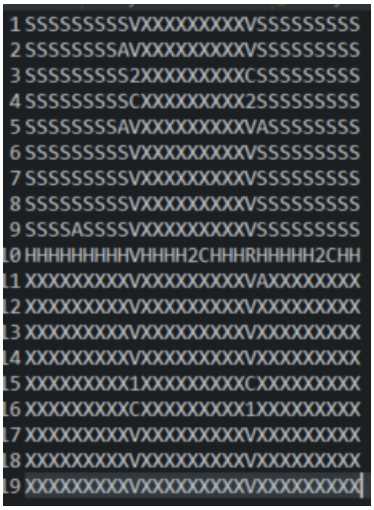

Figure 6 - Carte Finale (version texte)

Ce qui donne cela sur Repast Simphony :

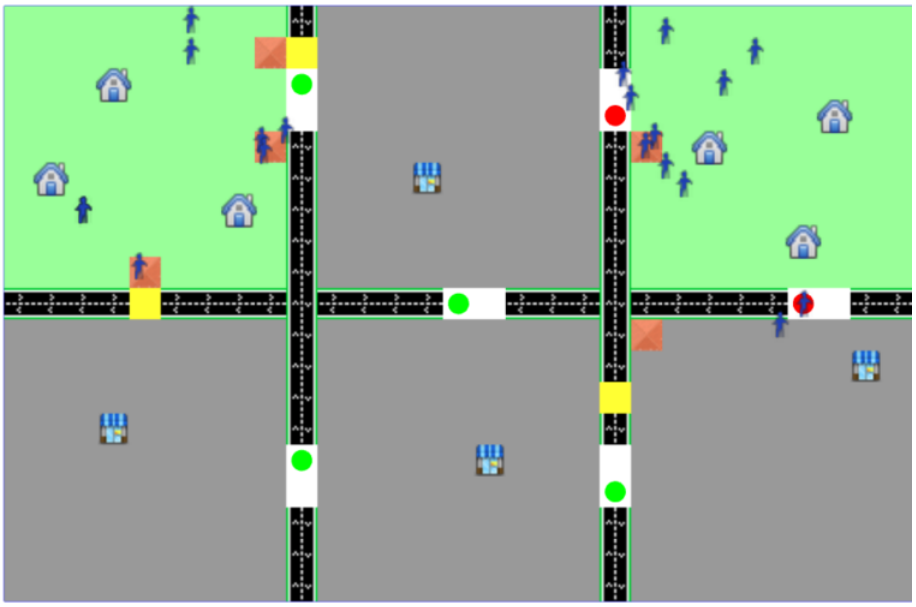

Figure 7 - Carte Finale (version visuelle)

4.2 Modification de paramètres au runtime

Afin de pouvoir simplifier nos tests et notre rendu graphique il était obligatoire de pouvoir ajuter nos paramètres au runtime afin de ne pas relancer la compilation entière du programme qui peut être longue.

4.2.1 Choix de la Map au runtime

L'utilisateur peut choisir la map à charger en spécifiant le nom de la map.

4.2.2 Choix du nombre d'humains au runtime

Cette option était primordiale afin de tester rapidement, le nombre d'humain et donc l'optimisation de notre ville.

4.2.3 Choix du temps d'attente des bus au runtime

Comme pour les humains, nous cherchions à optimiser le traffic en trouvant le temps d'attente le plus pertinent pour chaque bus aux arrêts. Ce temps est exprimé en nombre de ticks.

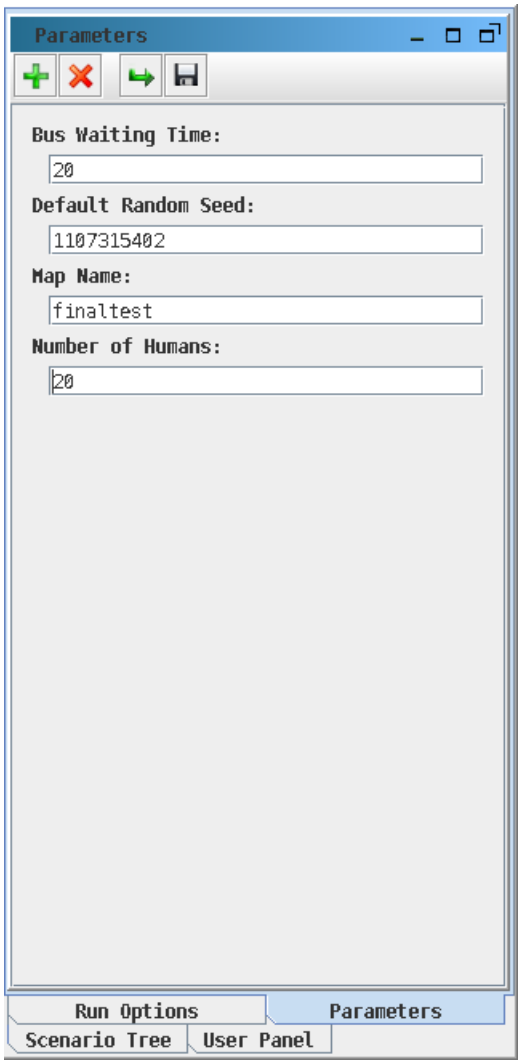

Figure 8 - Menu de customisation

Pour appliquer le changement des valeures modifiées, il suffit de cliquer sur la petite disquette pour enregistrer puis simplement de lancer le programme.

4.3 Création d'un installeur

Pour faciliter l'utilisation du logiciel pour de potentiels utilisateurs, nous fournissons un installeur, qui va permettre très facilement de tester le programme en l'ouvrant. Pour plus d'informations, la partie manuel d'utilisation l'explique plus en détails.

5 Manuel d'utilisation

5.1 Installeur automatique

Dans l'archive, vous trouverez un fichier "setup.jar". Si vous êtes sur Windows, vous pouvez cliquer dessus. Si vous êtes sur Mac ou Linux, un simple "java -jar setup.jar" va lancer le jar, vous devriez avoir cette fenêtre :

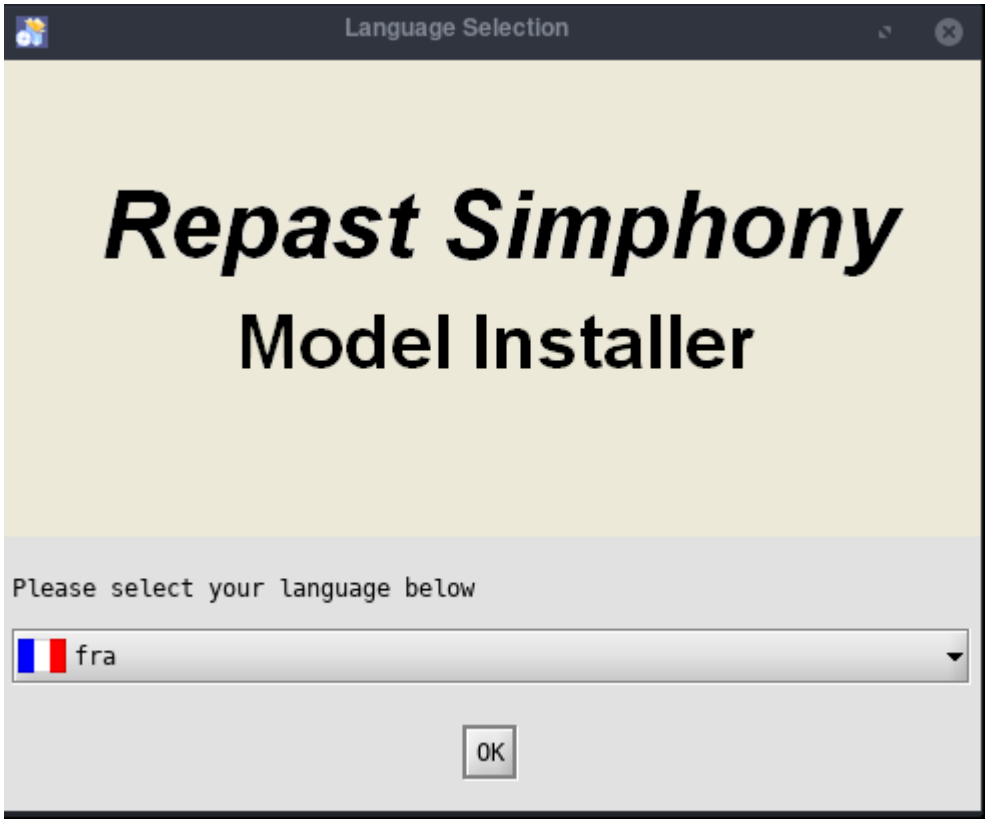

Figure 9 - Fenêtre d'installation

Selectionnez votre langue, puis cliquez sur OK.

Ensuite cliquez sur deux fois sur Suivant, acceptez les termes, cliquez sur Suivant.

Choisissez le chemin d'installation.

Vous devriez être à l'étape 5, veillez à cocher les 3 cases et cliquer sur Suivant.

Une fois l'installation terminé, vous allez pouvoir choisir si vous désiez des raccourcis ou pas.

Enfin, cliquez sur Suivant puis Terminer et rendez vous dans votre répertoire d'installation.

Si vous êtes sur Windows, lancez start_model.bat. Si vous êtes sur Linux/Mac lancez "start_model.command". Puis passer directement a la section Ecran de simulation de ce manuel.

5.2 Installation manuelle

Tout d'abord, il faut ouvrir Eclipse.
Allez dans "File" puis "Open Projects from File System".
Cette fenêtre devrait apparaitre :

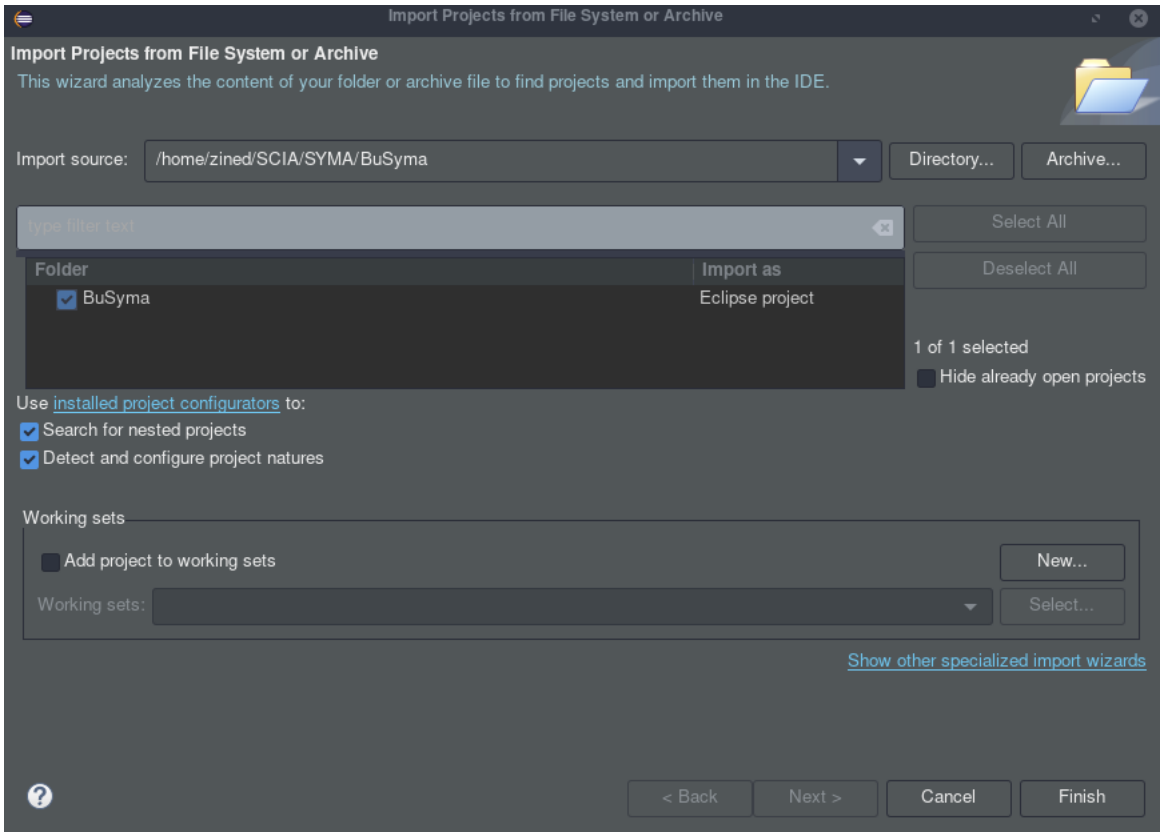

Figure 10 - Fenêtre d'installation

Choissiez simplement le repertoire du dossier BuSyma en cliquant sur le bouton "Directory..." et cliquez sur le bouton "Finish"
Une fois le projet importé, cliquez sur la petite flèche à droite du bouton Play de couleure verte et choisissez "BuSyma Model".

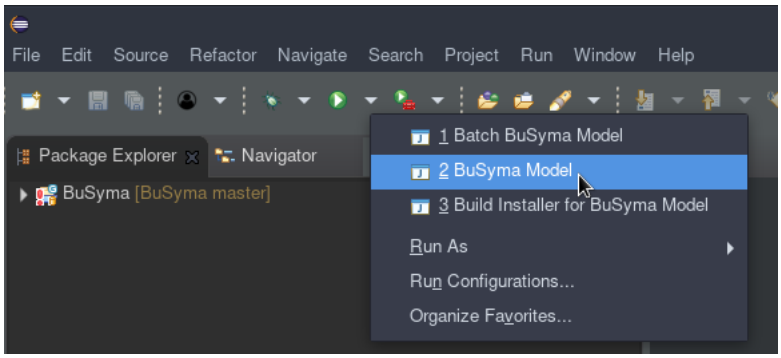

Figure 11 - Fenêtre d'installation

La fenêtre de simulation devrait s'ouvrir, passer à la section "Ecran de simulation" de ce manuel.

5.3 Ecran de simulation

Après les étapes précédentes, cette fenêtre devrait apparaitre :

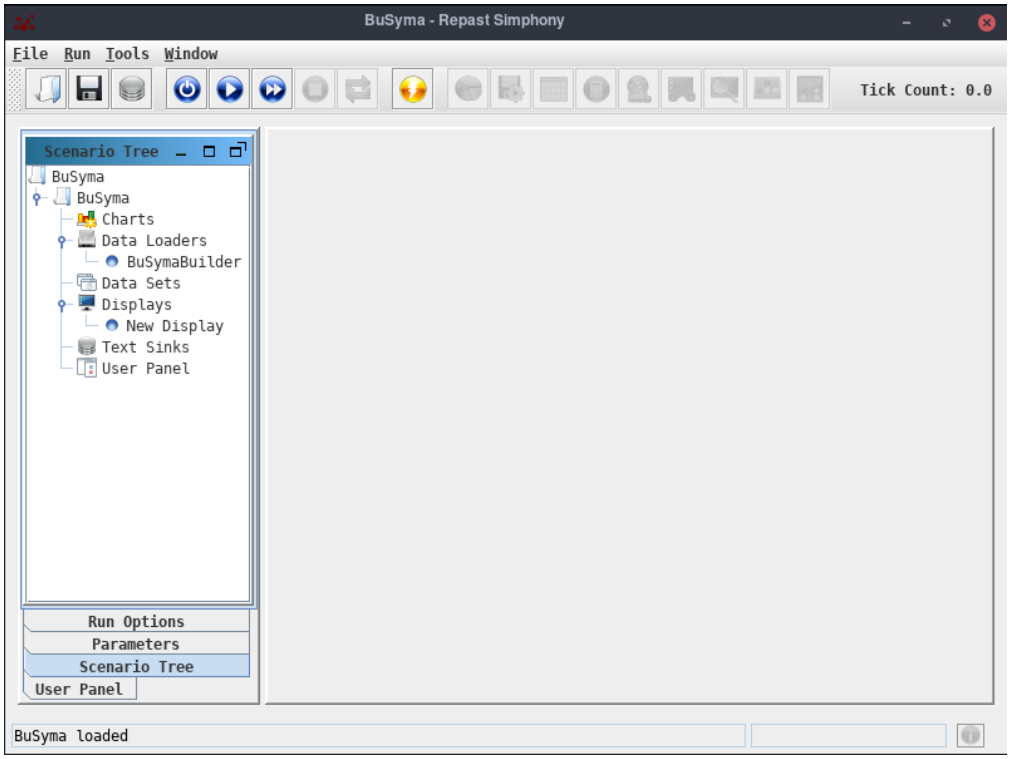

Figure 12 - Fenêtre d'installation

Vous pouvez choisir vos configurations dans "Parameters" et modifier la vitesse de la simulation dans "Run Options". Cliquer ensuite sur le boutton bleu "Play" et la simulation démarrera.

5.4 Démonstration en vidéo

Nous avons fait une petite vidéo qui explique comment installer le programme via l'installeur automatique et qui donne un aperçu de nos bonus ainsi qu'une démo d'une de nos maps de tests. Le lien pour aller voir la démo est celui-ci : `https://youtu.be/GqcMOrpS1jM`

6 Conclusions

6.1 Conclusion Générale

Sachant que nous étions deux, il était de notre devoir de s'organiser correctement et de travailler sérieusement afin de compenser le membre en moins. Pour cela, nous avons directement pensé à des façons de tester nos algorithmes. C'est pour cela que nous avons commencé par créer le parseur de map ainsi que la modification des attributs au runtime (nombres d'humains, bus etc). Le brainstorming fait avant de commencer à coder a été très bénéfique. Il a permis de se mettre d'accord sur l'architecture du code, les stratégies qu'on allait implémenté et la repartition des tâches.

Globalement le projet était interessant. Il nous a permis de refaire du Java et de découvrir un nouveau paradigme et une autre façon de résoudre un problème.

6.2 Difficultés rencontrées

6.2.1 Coordonnées entières de la grille / Déplacements continus

Dans les difficultés que nous avons pu rencontrer au cours de la réalisation de ce projet, on parlera d'abord d'un aspect de modélisation un peu particulier. Nous voulions essayer de faire se déplacer nos personnages sur une grille mais de manière continue (pas simplement case par case) et pour cela nous avons mis en place un espace en plus de la grille, qui nous sert à afficher les déplacements. Cela dit, il était difficile de savoir à quelle coordonée d'espace correspond une coordonnée de la grille, et se souvenir de la mettre à jour systématiquement.

6.2.2 Dijkstra et déplacements en zigzag

Ensuite, comme dit précédemment, nous avons dû implémenter nous même un algorithme Dijkstra pour se faire déplacer nos personnages, et dans un soucis de réalisme, nous voulions autoriser les déplacements diagonaux. Cependant,

les chemins trouvés parraissaient peu naturels et semblaient dévier de la trajectoire optimale en ligne droite. Les trajectoires trouvées étaient toujours les plus courtes, mais semblaient mauvaises en terme de ressenti. Nous avons donc décidé d'appliquer un poids plus important sur les diagonales pour observer des déplacements plus naturels.

6.2.3 Manque de Documentation

C'est le principal problème rencontré lors de la réalisation de ce projet. Repast Simphony permet d'avoir un rendu assez rapidement, et de facilement modifier son modèle. Cependant, il est très difficile de trouver des solutions à des problèmes spécifiques. La documentation liste les classes disponibles via l'api mais les exemples d'utilisations sont très rares. Même pour des choses assez basiques, impossible de trouver des exemples concrets... Nous voulions par exemple créer un graphique qui affiche le temps de trajet d'un humain vers sa destination. Ainsi, on ne peut avoir de données qu'à la fin du trajet de l'humain. Or nous n'avons trouvé aucune manière de fournir des données de cette façon. Nous avons donc contourné le problème et avoir des résultats beaucoup moins visuels.

6.2.4 Modèles 2D

Nous avons voulu créer un modèle pour l'affichage des feux de signalisation, la couleur devant changer dynamiquement. La tâche s'est révélée être assez dure, notamment à cause du manque de documentation. Nous avons réussi à afficher un cercle qui change de couleur comme nous le voulions mais il est sur fond blanc, aucun moyen dans la documentation pour savoir comment mieux personnaliser un modèle 2D avec l'api repast simphony pour java.

6.3 Conclusions Personnelles

6.3.1 Debrenne Antoine

J'ai trouvé ce projet très intéressant à faire. L'idée d'implémenter pas à pas les interactions jusqu'à avoir un rendu complet a été très satisfaisant. Le concept de pouvoir lancer des simulations et en varier les paramètres pour voir les changements était intéressant aussi.
Ce projet étant à faire en groupe, il m'a appris à bien gérer mon temps, au milieu de nos autres responsabilités et ce fut une bonne chose. Cependant, nous avons réalisé ce projet en binome uniquement, et le manque d'un troisième membre abaisse la qualité de notre rendu, c'est une de mes déceptions concernant ce projet.
Nous avons voulu utiliser Repast Simphony pour nous familiariser avec ce logiciel et refaire un peu de java car ça n'est pas le point fort de notre formation. Cependant, il manque encore vraiment de notoriété et ainsi il est difficile de se renseigner dessus quand nos problèmes étaient très précis.

6.3.2 Rebiai Zinedine

Projet plutôt intéressant et pédagogue. Pouvoir faire des simulations, les étudier et faire des mathématiques dessus est un travail que nous serons sûrement ménés à faire plus tard.
En ce qui concerne l'aspect technique, nous avions voulu utiliser Repast Simphony puisque nous l'avions étudiés en cours et nous ne le connaissions pas. Ce logiciel est puissant mais assez méconnu ce qui implique un manque cruel de documentation en cas de bug.
En terme de groupe, l'entente était plutôt bonne et nous étions organisés. Nous sommes plutôt satisfait du projet bien qu'avec un troisième membre nous aurions pu aller plus loin.