

Université de Cergy-Pontoise

RAPPORT

pour le projet Génie Logiciel
Licence d'Informatique deuxième année

sur le sujet

Urbain

rédigé par

Matthieu VILAIN et Quentin GERARD



Mai 2015

Table des matières

1	Introduction	3
2	Spécification	3
2.1	Présentation du jeu	3
2.1.1	Contexte	3
2.1.2	Le Jeu	3
2.2	L'Environnement	3
2.3	Personnages	4
2.4	Fonctionnalités	4
3	Réalisation	5
3.1	Le model - présentation des classes	5
3.1.1	La ville	5
3.1.2	La population	6
3.2	Le moteur de jeu : mode normale	9
3.2.1	Gestion des jauge des personnages	9
3.2.2	Gestion de la routine	10
3.2.3	Gestion des itinéraires : Algorithme A*	11
3.2.4	Collecte de données statisqtiques	12
3.2.5	Fin de partie : enregistrement du score	13
3.3	Le moteur de jeu : mode autonome	14
3.3.1	Le Q-Learning	14
3.3.2	Notre implémentantion	15
3.4	Le moteur de jeu : Prototype	15
3.5	Interface Homme-Machine	16
3.5.1	La Map	17
3.5.2	Informations des personnages	17
3.5.3	Information des infrastructures	18
3.5.4	Informations de l'Horloge	19
3.6	Logging : Log4J	19
3.7	Phase de test	19
3.8	Leaderboards	20
3.8.1	Le Site Web	20
3.8.2	Principe du classement	20
3.8.3	Modélisation d'un classement	21
3.8.4	Ajout d'un score dans un classement	21
4	Manuel Utilisateur	22
4.1	Déroulement d'une partie	22
4.1.1	Mode Normal	22
4.1.2	Mode Autonome	22
4.2	Manuel de l'IHM	23
4.2.1	Information des Infrastructures	24
4.2.2	Information de l'Horloge	24
4.2.3	La Map	25
4.2.4	Liste des personnages	26
4.2.5	Les statistiques	28
4.2.6	La routine	28
4.3	Après la partie	30

5	Déroulement du projet	33
5.1	Répartition des tâches	33
5.2	Synchronisation du travail	33
5.3	Utilisation du temps	33
6	Conclusion	35
6.1	Problèmes Rencontrés	35
6.2	Améliorations Possible	35
6.3	Ressenti sur le projet	35

Remerciements

Les auteurs du projet voudraient remercier M. Tianxiao Liu.

1 Introduction

Le projet Le projet consiste à la création d'un jeu vidéo simulant une vie urbaine, dans lequel plusieurs individus vivent leur vie au sein d'une ville. L'utilisateur pourra influer sur le comportement des individus et les paramètres de la ville.

L'Équipe Notre équipe est composé de Matthieu VILAIN, étudiant en CMI SIC, ainsi que de Quentin GERARD, également étudiant en CMI SIC.

Nos motivations Nous avons choisi ce projet car il représente une opportunité pour chacun de nous d'explorer des domaines/notions qui nous intéressent, et dans lesquelles nous voulons nous perfectionner.

2 Spécification

2.1 Présentation du jeu

2.1.1 Contexte

Vous avez découvert un remède contre la vieillesse, pour vous remercier la population vous a élu maire de la ville, mais rapidement, d'autres problèmes vont arriver. En effet, maintenant que l'âge n'est plus une inquiétude, le morale de votre population semble les préoccuper de plus en plus. Certains cas de décès, ayant pour cause la fatigue mentale, l'insatisfaction de la situation financière, ou encore le manque de vie sociale, ont été recensé. Vous avez donc pour but de faire survivre votre population et de combler ses besoins.

2.1.2 Le Jeu

Le jeu consiste en une simulation de l'évolution d'une population dans un milieu urbain. Le but du jeu est de faire survivre la population le plus longtemps possible. En mode "normal", le joueur pourra influer sur les différents personnages de la population, afin de la faire survivre. En mode autonome, la population prend elle même des décisions afin de survivre.

2.2 L'Environnement

L'environnement est une ville composé de différentes infrastructures :

- Tracé (routes)
- Batiments :
 - Maison (domicile des personnages)
 - Travail (lieux de travail des personnages)
 - Loisir

Les différents batiments auront :

- Un nom
- Un nombre maximum d'utilisateurs
- Une influence variable sur l'émotion des personnages (exemple : Un bâtiment Loisir rendra le personnage plus heureux, au contraire, aller travailler le rendra plus malheureux)
- Une adresse
- Une taille

Chaque batiment possèderont ses caractéristiques propres :

- Maison
 - Un ou plusieurs propriétaires
- Travail et Loisir :
 - Des horaires d'ouvertures et de fermetures

- Un temps d'utilisation moyen
- Une file d'attente

Les tracés (routes) permettront aux personages de se déplacer dans l'environnement. L'environnement sera donc un quadrillage de différentes infrastructures, ainsi qu'une horloge permettant de simuler les différentes phases d'une journée (nuit, matin, après-midi, soir).

2.3 Personnages

Le réel but du projet est d'analyser l'évolution d'une population dans un milieu urbain et cela sous la forme d'un jeu. Le point le plus important sera donc la vie quotidienne des personnes.

Chaque personnage aura pour informations de base :

- Un nom
- Un age
- Un sexe
- Une adresse (Un domicile représenté par un Batiment de type Maison)
- Un travail, avec des horaires de travail

Chaque personnage suivra une routine modélisé par une suite d'action, laquelle pourra être modifiée par le joueur. Les différents types d'actions seront :

- Dormir
- Se divertir
- Aller travailler
- Se déplacer

Chaque personnage possèdera trois jauge : Émotion, Money et Family ; représentant respectivement son besoin en repos, travail, et divertissement. Chaque jauge diminuera au fur et à mesure du temps, le seul moyen de les augmenter est d'utiliser les batiments correspondant.

2.4 Fonctionnalités

Fonctionnalités du programme :

- Le joueur aura plusieurs actions possibles afin d'influencer sur l'évolution de la ville :
 - Agir sur le temps, en stoppant l'horloge, mais aussi en l'accélérant ou la ralentissant
 - Accéder aux informations des bâtiments
- Le joueur aura plusieurs actions possibles afin d'influencer sur l'évolution de la population :
 - Accéder aux différentes informations des personnages :
 - Les informations de base (Nom, Age, Domicile, Travail)
 - Evolution du comportement du personnage dans le temps via des graphiques
 - La liste des actions à venir par le personnage
 - Agir sur le comportement des différents personnages :
 - Ajouter des actions à faire au personnage
 - Supprimer des actions prévues dans l'activité du personnage

3 Réalisation

3.1 Le model - présentation des classes

3.1.1 La ville

La ville est le premier élément qui constitue notre projet. Elle est composé de différentes infrastructures : Les routes, les maison, les bâtiments de travail et les divertissement. Chaque type d'infrastructure possède une utilité qui lui est propre :

- Les Routes permettent aux personnages de se déplacer
- Les Maisons permettent aux personnages de se reposer le soir et regagner de l'émotion
- Le Travail est une activité imposé à chaque personnage et sera la principale source de baisse d'émotion
- Les Divertissement permettent aux personnages lorsqu'ils ne dorment pas de regagner de l'émotion

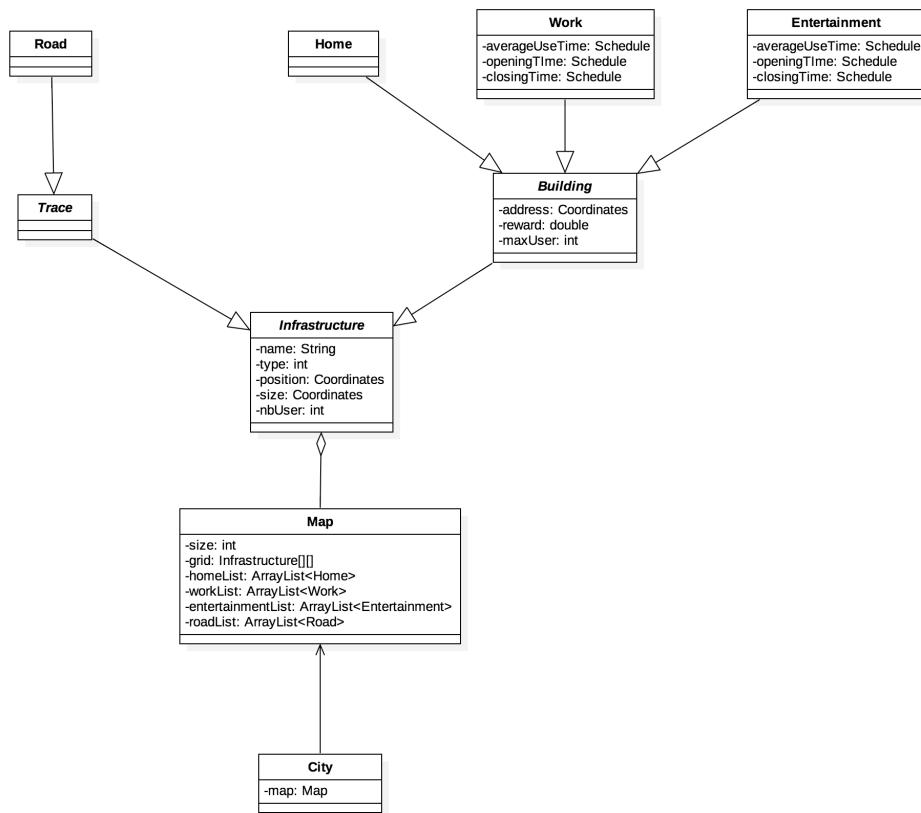


FIGURE 1 – Diagramme UML des composantes de la ville

Toutes les infrastructures possèdent en commun :

- Un nom de type String
- Un type au format int
- Une position dans la Map
- Un taille
- Et un nombre d'utilisateur courant

Et les bâtiments (hors routes) possèdent tous en commun ces caractéristiques :

- Une adresse, seule point d'entrée dans le bâtiment au niveau de la Map
- Une récompense, qui peut être positive ou négative suivant le type de bâtiment
- Un nombre maximum d'utilisateur

En plus de ces caractéristiques, les bâtiments de travail et les divertissement possède un temps d'utilisation moyen par les personnages, ainsi que des horaires d'ouvertures durant lesquelles les personnages pourront utiliser ces bâtiments. Il est également impossible pour un personnage d'utiliser un bâtiment lorsque celui-ci a atteint son nombre maximum d'utilisateur. Les routes et les maisons sont ouverts 24h/24.

La taille de la Map ainsi que la répartition des infrastructures est déterminé à l'avance dans un fichier CSV. La création de la Map dans la mémoire se fait grâce au pattern design Builder. Chaque ligne du fichier CSV renseigne, le type, l'adresse, la taille et sa position dans la Map. Ainsi l'objet MapBuilder va lire les lignes du fichier une par une grâce à la bibliothèque Apache Common CSV, et faire appelle au autres Builder correspondant à chaque type d'infrastructure.

Durant leur création, les Infrastructures de type Work ou Entertainement, se voient également attribuer un nom, des horaires d'ouvertures et leur récompenses. Ces informations sont aussi renseigné à l'avance dans un fichier CSV.

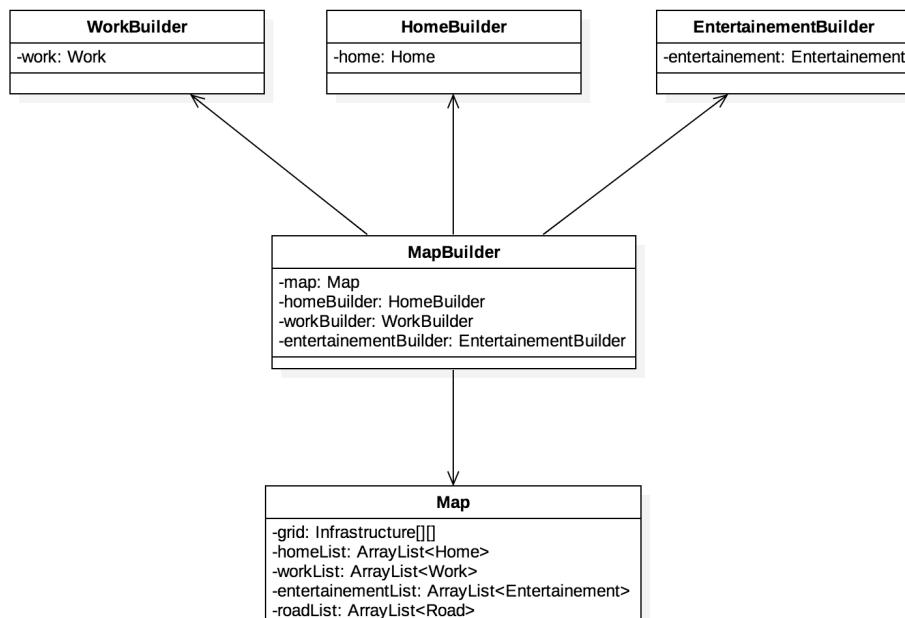


FIGURE 2 – Diagramme UML de la construction de la ville

3.1.2 La population

La population est le second ensemble qui constitue notre projet. Nous l'avons voulu le plus adaptatif possible. Cependant la création de la population dépend de la ville car, pour le mode normal, chaque maison ainsi que chaque lieu de travail ne peut contenir qu'un seul personnage (par soucis de réalisme et de bonne répartition de la population sur la carte). Pour le mode autonome, au lancement du jeu, chaque maison ne peut avoir qu'un seul personnage. Au cours de l'évolution de la population plusieurs personnages pourront habiter dans une même maison. Notre population est donc limitée à 15 personnages maximum pour le mode normal (car 15 bâtiments de travail) et 60 en mode autonome (car 60 maisons). Si l'on décide d'augmenter la taille de la ville ou d'enlever les limitations de personnages par maison, la ville pourrait contenir une plus grande population. Cependant, une trop grande population provoquera un ralentissement de l'interface graphique mais le moteur de jeu est parfaitement capable de faire tourner une grande population.

Création des personnages Un personnage est composé d'information de base comme d'un nom, d'un prénom, d'un sexe, d'un âge et d'un numéro d'identité.

Mode normale	Mode autonome
niveau easy : 1	[1 - 60]
niveau normale : 3	
niveau hard : 5	
niveau pro : 15	

TABLE 1 – Taille de la population

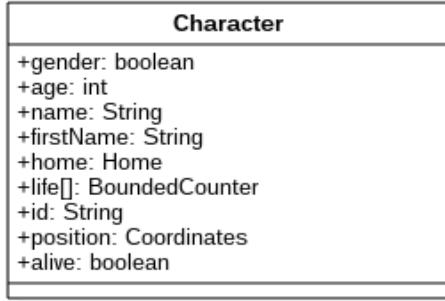


FIGURE 3 – Classe abstraite Character

Pour le nom, le prénom et le sexe du personnage, ils sont initialisés à partir de fichier CSV. Le premier fichier contient une liste de 300 noms de familles et le second une liste de 200 prénoms, 100 masculin et 100 féminin. Lors de la création du personnage, le programme va prendre au hasard un nom dans le fichier name.csv et un prénom (associé à un sexe) dans le fichier firstName.csv. L'âge du personnage est simplement choisi aléatoirement entre 10 et 100 ans. Le numéro d'identité du personnage est unique, il est calculé à partir de toutes les informations du personnage grâce à un code de hashage. Ce code nous permettra de reconnaître le personnage. Ce grand nombre de choix nous permet de garantir une grande diversité au sein de la population.

Comme le montre la figure 1, un personnage possède également une maison. Cet élément est également attribué aléatoirement via une recherche dans la liste de lieux d'habitation de la carte de jeu.

Un personnage possède également un tableau de jauge contenant :

- une jauge d'émotion
- une jauge d'argent
- une jauge de famille

Ces jauge peuvent varier de 0 à 100. Ces jauge représentent les critères de vie du personnage. Elle est initialisé à 75 en début de partie mais elle variera en fonction des actions des personnages. Si l'une des jauge du personnage arrive à 0 il meurt.

Nous avons voulu rendre nos personnages le moins statique possible. Ainsi d'une partie à l'autre, la chance de tomber sur des personnages avec les mêmes propriétés est très faible.

Les différents types de personnages La classe abstraite Character possède deux classes filles avec des fonctionnalités différentes. La première, NCharacter, représente les personnages utilisés dans le mode de jeu normal. La seconde, QCharacter, représente les personnages utilisés dans le mode de jeu autonome.

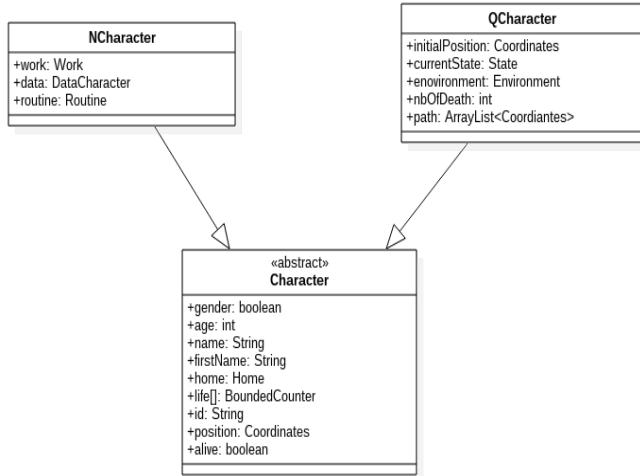


FIGURE 4 – Classe abstraite Character et ses classes filles

Les NCharacters possèdent un lieu de travail qui est attribué aléatoirement via une recherche dans la liste des lieux de travail de la carte en début de jeu. Ils possèdent également un objet de classe DataCharacter pour sauvegarder des données sur le personnage pendant le jeu (voir partie statistique de jeu). Enfin, les NCharacter possèdent une routine, une liste d’actions qu’il executera tout les jours. (voir routine partie moteur)

Les QCharacters possèdent une position initiale qui sera leur maison à chaque réapparition pour leur permettre d’y retourner en cas de besoin de sommeil. Ils possèdent également un environnement qui est une représentation binaire de la carte (0 si la case est praticable, 1 si c’est un obstacle). Cet environnement est couplé à un état courant qui représente la position courante du personnage dans la carte simplifié. On compte également le nombre de mort de chaque personnage pour faire des statistiques avec . Enfin, un QCharacter possède une liste de coordonnées qui lui permettra de retourner à sa maison si il a un besoin en sommeil.

Organisation de la création de la population La création des personnages se fait grâce au design pattern builder (figure 5).

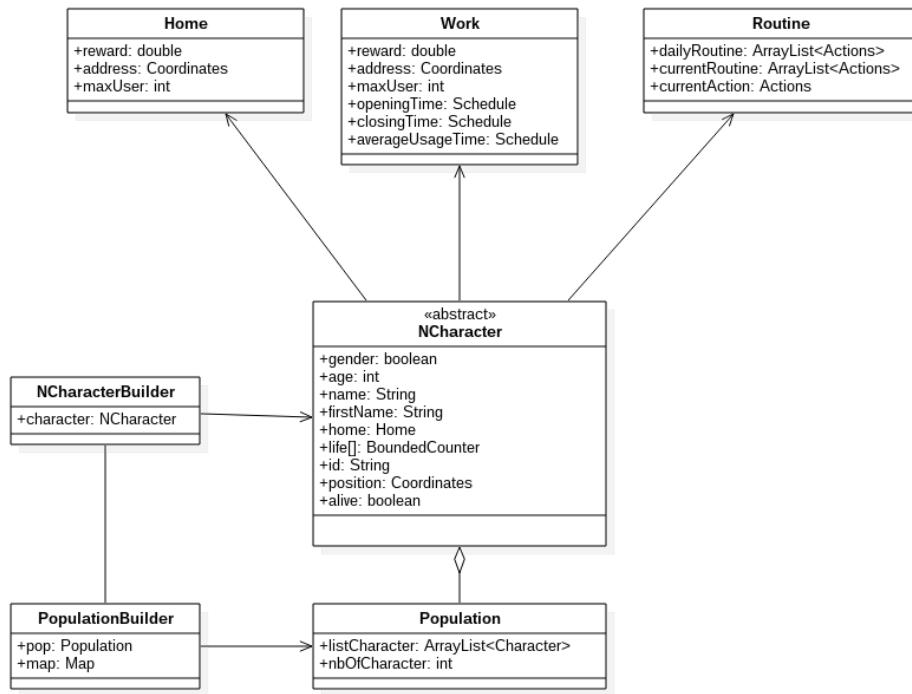


FIGURE 5 – Architecture de création des NCharacters et de la population

La classe PopulationBuilder va construire une population grâce à la carte de jeu (pour assigner les résidences) et grâce au CharacterBuilder. Ce dernier va faire la lecture dans les fichiers CSV grâce à la bibliothèque Apache-commons-csv et va faire les choix aléatoires pour initialiser les différentes composantes de nos personnages.

On utilise la même architecture pour créer la population de QCharacter dans le mode autonome.

3.2 Le moteur de jeu : mode normale

3.2.1 Gestion des jauge des personnages

Les jauge représentent les critères de vie des personnages, elles varient de 0 à 100 avec le temps. Si l'une d'elles atteint 0, le personnage meurt. L'évolution de ces jauge est totalement dynamique, elle se fera automatiquement en fonction des actions faites par le personnage. Certaines actions vont apporter un bonus différent sur les jauge du personnage alors que d'autres vont apporter des malus. Pour certaines actions le bonus est constant (par exemple dormir apportera toujours +30 en famille) mais pour d'autres le bonus/malus est variable en fonction du lieu dans lequel se fait l'action. Par exemple travailler dans un atelier auto apportera un malus de -25 en émotion car la tache est difficile alors que travailler dans une boutique d'électronique apportera un malus de -15. Même fonctionnement pour les bonus des loisirs. Ces valeurs sont initialiser lors que l'initialisation des bâtiments, elles proviennent donc d'un fichiers CSV. Enfin les rewards de sont pas toujours effectif au même moment. Pour la plupart des actions, le personnage perçoit le reward en fin d'action. Pour l'action de déplacement, dans un soucis de mieux représenter la réalité, le malus est retiré à chaque itération de temps. C'est à dire que plus le chemin que le personnage a à faire est long, plus il perd de l'émotion.

Action	Emotion	Money	famille	Effectivité
Sleeping	+15	0	+30	En fin d'action
Chilling	0	0	+5	En fin d'action
Shifting	-1	-0.33	-0.33	A chaque itération de temps
Working	Malus variable [-25 ;-10]	Bonus variable [+15 ;+30]	-5	En fin d'action
Entertain	Bonus variable [+5 ;+20]	Malus variable [-15 ;0]	-5	En fin d'action

TABLE 2 – Répartition des rewards

3.2.2 Gestion de la routine

La routine est un enchaînement d'actions que va exécuter le personnage. Le personnage peut exécuter 5 types d'actions différentes regrouper en familles : les déplacements et les occupations. Les actions d'occupation sont reliées à un lieu alors que les actions de déplacement sont reliées à un lieu de départ, un lieu d'arrivé et un chemin entre les deux.

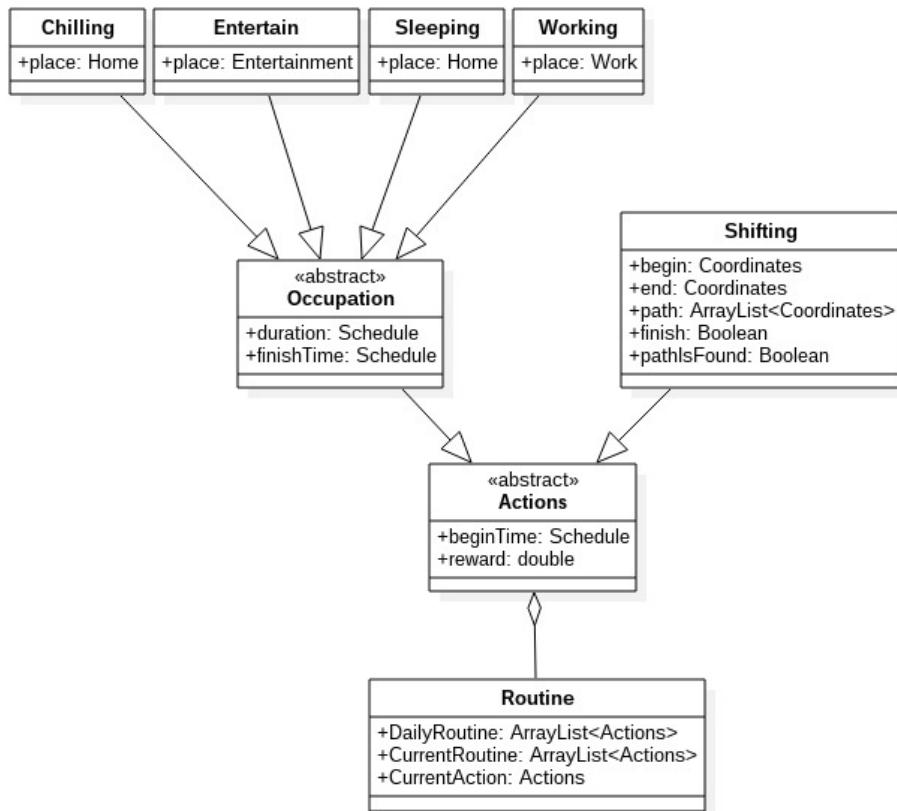


FIGURE 6 – Architecture de gestion des routines

L'un des dilemme du projet est l'interaction entre les différentes actions pour que chaque personnage puisse suivre une suite d'actions indépendantes mais que l'utilisateur puisse ajouter des actions à faire sans perturber le bonne enchaînement des actions.

Nous avons décidé de décomposer ce problème en 3 partie :

- Une partie statique qui organise les grandes lignes des journées du personnage : métro/boulot/dodo
- Une partie dynamique qui évolue au court de la journée et qui représente la liste d'actions que le personnage a à faire
- Une partie utilisateur associé à différente options d'ajout/suppression d'actions

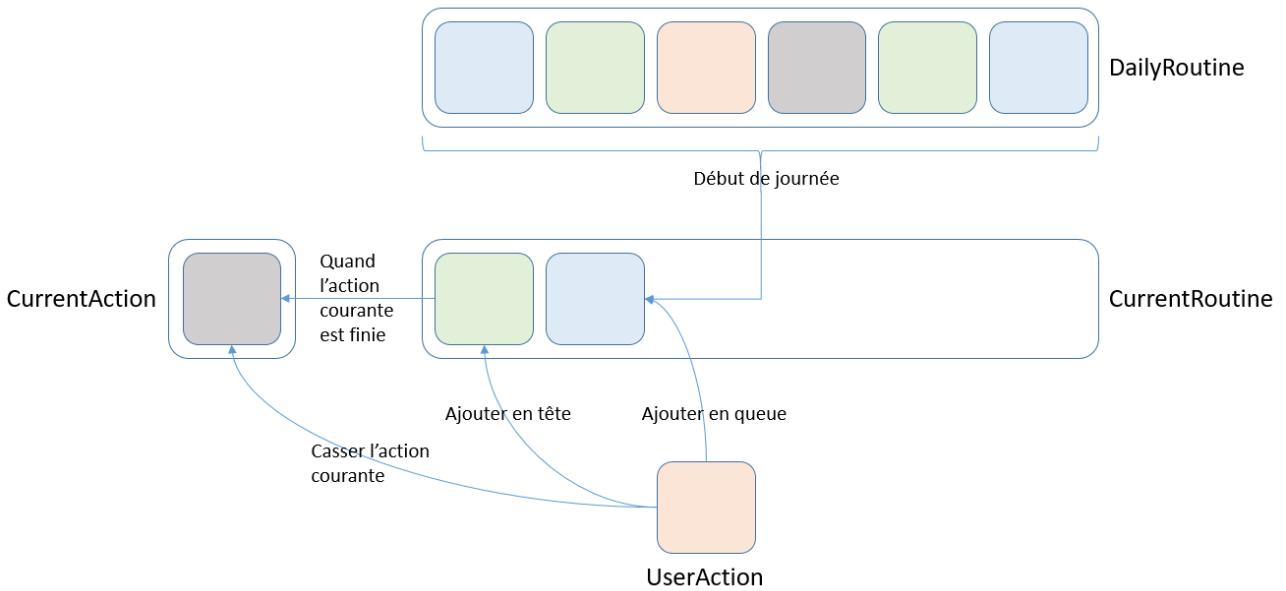


FIGURE 7 – Architecture de gestion des routines

La liste d’actions communes à chaque journées est stocké dans la liste DailyRoutine. A chaque début de journée, on ajoute toutes les actions de cette liste à celle de la journée courante, CurrentRoutine. La currentRoutine à les même propriétés qu’une files mais avec plus de possibilité d’ajout d’actions. On défile currentRoutine et on ajoute cette action à l’action courante, currentAction. Cette action est exécuté par le personnage et lorsqu’elle est finie, on défile à nouveau la currentRoutine. Enfin l’utilisateur peut soit ajouter un action en tête ou en queue dans la currentRoutine.

3.2.3 Gestion des itinéraires : Algorithme A*

L’utilisateur peut demander à un personnage de se déplacer d’un point A à un point B mais il n’a pas besoin de lui spécifier exactement le chemin à faire. En effet le personnage est capable de trouver le chemin le plus court pour aller d’un point A à un point B grâce à l’algorithme de path finding A*.

Fonctionnement de l’algorithme Le principe de l’algorithme est de toujours se rapprocher le plus possible du point d’arrivé. Cependant on n’oublie pas les solutions qui nous éloignent directement de l’objectif car elles pourraient faire partie du chemin final. On représente notre carte sous forme de graph avec des noeud praticable (routes et entrées de bâtiments) et des noeud obstacle (bâtiments).

On utilisera 2 liste chainées :

- une liste ouverte contenant tout les noeuds étudiés
- une liste fermé contenant les noeuds qui semblent faire partie du chemin solution.

Pour noter la pertinence d’un noeud, on calcule sa distance à vol d’oiseau au carré avec l’objectif grâce à l’équation :

$$P = (x_{arrivee} - x_{debut})^2 + (y_{arrivee} - y_{debut})^2$$

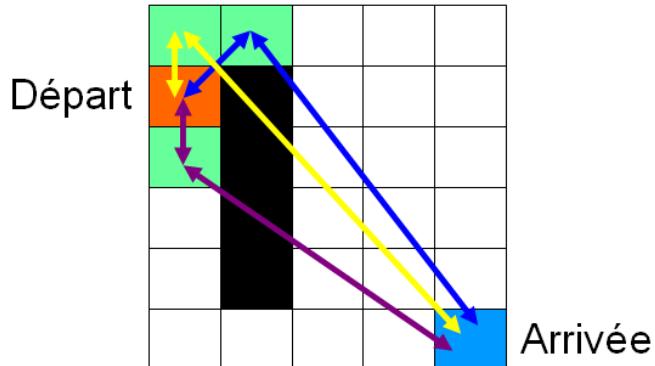


FIGURE 8 – A* : calcul de la pertinence d'un noeud

Enfin on repète les étapes suivantes pour construire le chemin solution :

1. On prend comme noeud courant le noeud de départ
2. On regarde tout les noeuds voisins
 - Si un noeud voisin est un obstacle, on l'oublie
 - Si un noeud voisin est déjà dans la liste fermé, on l'oublie
 - Si le noeud voisin est dans la liste ouverte, on le met à jour si il a une meilleure pertinence que celui déjà présent dans la liste ouverte (on change sa pertinence et ses parents)
 - Sinon, on ajoute le noeud voisin dans la liste ouverte avec comme noeud parent le noeud courant
4. On cherche le meilleur noeud de toute la liste ouverte (si cette dernière est vide, pas de solution, fin de l'algorithme)
5. On le met dans la liste fermé et on le retire de la liste ouverte
6. Ce noeud devient le noeud courant et on recommence (sauf si c'est le noeud d'arrivé, fin de l'algorithme)

Pour récupérer le chemin optimal il suffi juste de prendre le noeud et de regarder successivement tout les parents jusqu'au point de départ.

Cet algorithme nous assure de trouver le chemin optimal pour nos personnages.

3.2.4 Collecte de données statisqtiques

Pour permettre au joueur d'élaborer des stratégies et d'avoir une meilleure vision sur les actions passé des personnages, il a à sa disposition un certain nombre de statistiques.

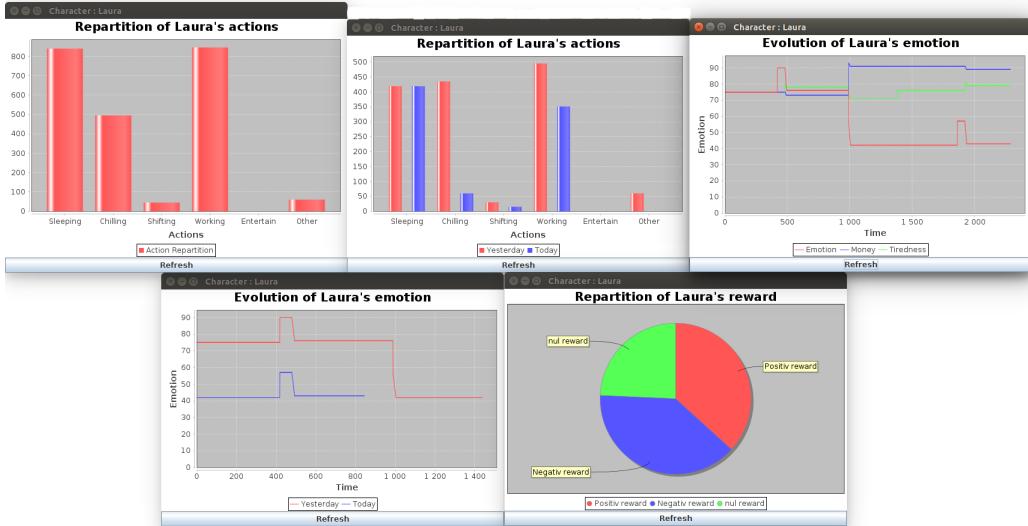


FIGURE 9 – Statistiques pour chaque personnage

Il peut accéder à :

- La répartition des différents types d’actions depuis le début du jeu
- La comparaison entre la répartition des types d’actions de la journée courante et celle de la journée précédente
- L’évolution des 3 jauge depuis le début du jeu.
- La comparaison de l’évolution de l’émotion entre la journée courante et la journée précédente
- La répartition des rewards en émotion (reward positif, reward négatif, reward nul)

Toutes ces statistiques sont gérées par le moteur et sont stockées dans l’objet `DataCharacter` de chaque personnages.

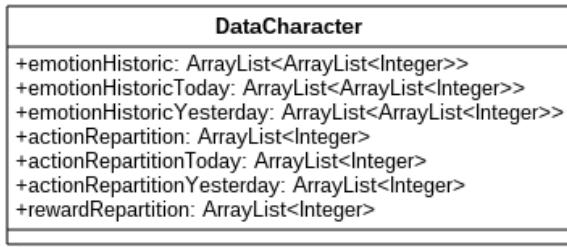


FIGURE 10 – Classe DataCharacter

3.2.5 Fin de partie : enregistrement du score

A la fin de la partie ou quand le joueur décide de revenir au menu, le jeu lui propose d’enregister son score sur notre site internet (voir partie leaderboard). Son score est calculé en fonction du temps de jeu. Plus longtemps il réussira à faire vivre sa population, plus son score sera grand. Une fois son score enregistré, le moteur génère une URL renvoyant vers notre site et contenant le niveau de difficulté et le score réalisé. Ces données seront interprétée par un formulaire sur le site et le score pourra être enregistré. Si l’utilisateur veux directement refaire une partie et ne pas perdre de temps à enregister son score sur sa machine de jeu, il peut scanner un QRCode avec son téléphone pour accéder à la version mobile du site. Il peut ainsi enregister son score sur son téléphone en même temps de refaire une nouvelle partie sur sa machine de jeu.

Le QRCode est une conversion en binaire d’une chaîne de caractères. Ce code binaire est ensuite convertit sous format image : les petit carrés noir représente les 1 et les blancs les 0. Nous transformons

donc notre URL en QRCode grâce à la bibliothèque Zxing de Google. Lors du scanne, le lecteur de QRCode proposera à l'utilisateur de suivre le lien vers notre site afin de s'enregistrer.

3.3 Le moteur de jeu : mode autonome

le mode autonome permet de simuler l'évolution d'une population dans un milieu urbain. Chaque personnage devra apprendre à se construire sa propre routine quotidienne afin de maximiser le niveau de ses 3 jauge de vie. Si l'une de ses jauge arrive à 0, le personnage déménage pour se faire une nouvelle vie (mais n'oublie pas se qu'il a appris) et change donc de maison.

Pour que les personnages puissent apprendre de leur expérience, nous avons décidé d'utiliser un algorithme d'apprentissage par renforcement, le Q-Learning.

3.3.1 Le Q-Learning

Le Q-Learning est un algorithme qui va permettre à un agent d'apprendre à trouver une récompense dans un environnement qu'il ne connaît pas. Pour chaque état de l'environnement, l'agent a la possibilité de faire plusieurs actions qui vont le mener à d'autres états de l'environnement. C'est ces actions pour aller d'état en état qui possède 3 valeurs, une pour chaque jauge, qui lui indique la qualité de cette action pour satisfaire cette jauge. Ces valeurs sont initialisées à 0. Grâce à l'utilisation de l'équation de Q-Learning, l'agent va mettre à jour ces valeurs afin de quantifier l'apport en émotion/en argent/en famille que cette action pourra lui apporter dans le futur.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\underbrace{r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a)}_{\substack{\text{reward} \\ \text{discount factor} \\ \text{estimate of optimal future value}}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

FIGURE 11 – Equation de Q-Learning

La fonction $Q(s, a)$ représente donc la quantification de l'espoir de recevoir un reward si l'on effectue l'action a en l'état s .

L'algorithme de Q-Learning L'algorithme suit les étapes suivantes :

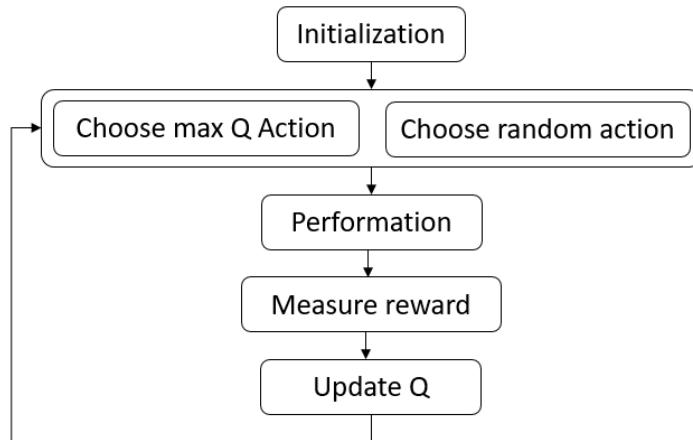


FIGURE 12 – Les étapes du Q-Learning

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Repeat (for each step of episode):
        Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
        Take action  $a$ , observe  $r, s'$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ ;
    until  $s$  is terminal

```

FIGURE 13 – Un algorithme possible de Q-Learning

3.3.2 Notre implémentation

La priorité L’agent ne peut pas assouvir tout ses besoins en même temps, il doit donc se focaliser sur un seul de ces besoins. Nous avons fait le choix de mettre la priorité sur la jauge ayant la valeur la plus basse. Ainsi si il à un besoin en argent, il ne prendra en compte que les valeurs d’argent des actions pour décider de la direction à suivre.

Problème rencontré Comme notre agent ne possède qu’une seul maison (en revanche il peut travailler et se divertir où il veut), il aurait difficile de lui faire apprendre à rentrer chez lui avant que sa jauge de famille arrive à 0. Nous avons donc combiné le Q-Learning avec l’algorithme A*. Ainsi il apprend à trouver un travail et un loisir mais quand il à besoin de rentrer chez lui, il cherche juste le chemin le plus court grâce à A* et le suit.

Pour plus de détails sur ce mode de jeu et le Q-Learning vous pouvez voir la vidéo que nous avons réalisé sur le sujet ou voir la recherche bibliographique de Matthieu Vilain (disponible jour de la soutenance). (Lien vers la vidéo : <https://www.youtube.com/watch?v=U7T4wBj0xHU>)

3.4 Le moteur de jeu : Prototype

Les algorithmes d’IA présentés au dessus (A* et Q-Learning) sont complexes à implémenter. Afin de se focaliser sur le fonctionnement de l’algorithme et de s’affranchir des contraintes de notre programme, nous avons d’abord réalisé des prototypes avec le minimum de graphique.

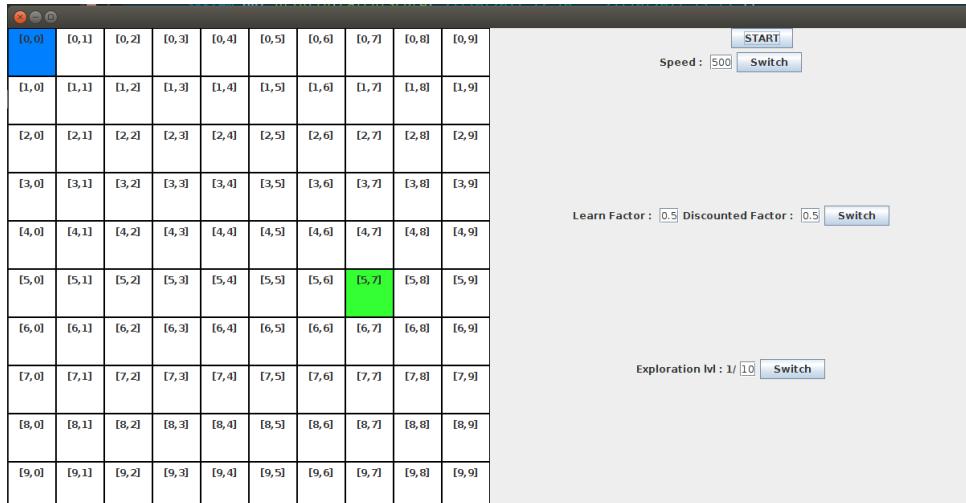


FIGURE 14 – Prototype pour le Q-Learning

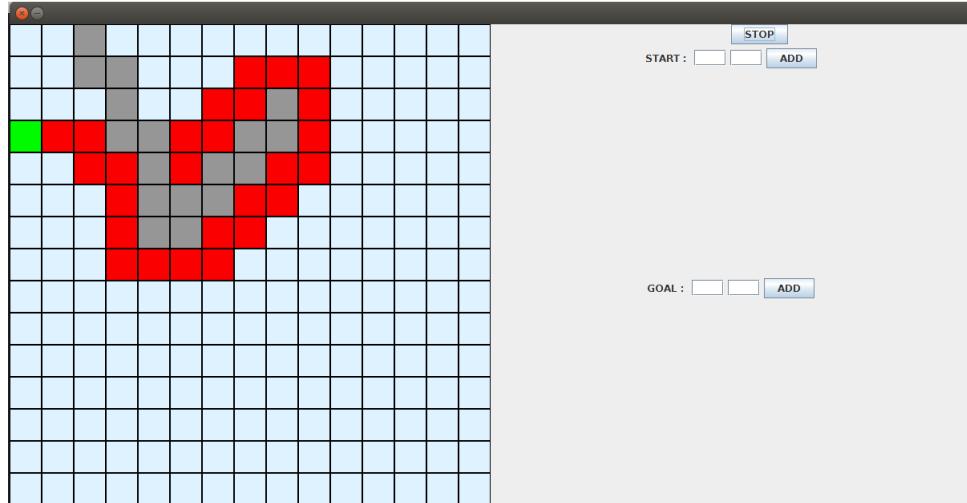


FIGURE 15 – Prototype pour A*

3.5 Interface Homme-Machine

L'interface graphique est réalisé grâce à Java Swing/AWT/Java 2D et est composé de plusieurs éléments :

- L'horloge du jeu, permettant de savoir quelle est la date et l'heure dans notre ville fictive
- La map, ou nous pouvons voir les infrastructures, et les personnages se déplacer
- La liste des personnages accompagné de l'état de leur barre d'émotion
- Les informations du batiments sélectionné par l'utilisateur

Chacun de ces éléments sont séparé dans différentes classes qui héritent de la classe JPanel , ce qui permet un assemblage facile de l'interface graphique, et une permutation plus facile entre différentes versions d'un même éléments.

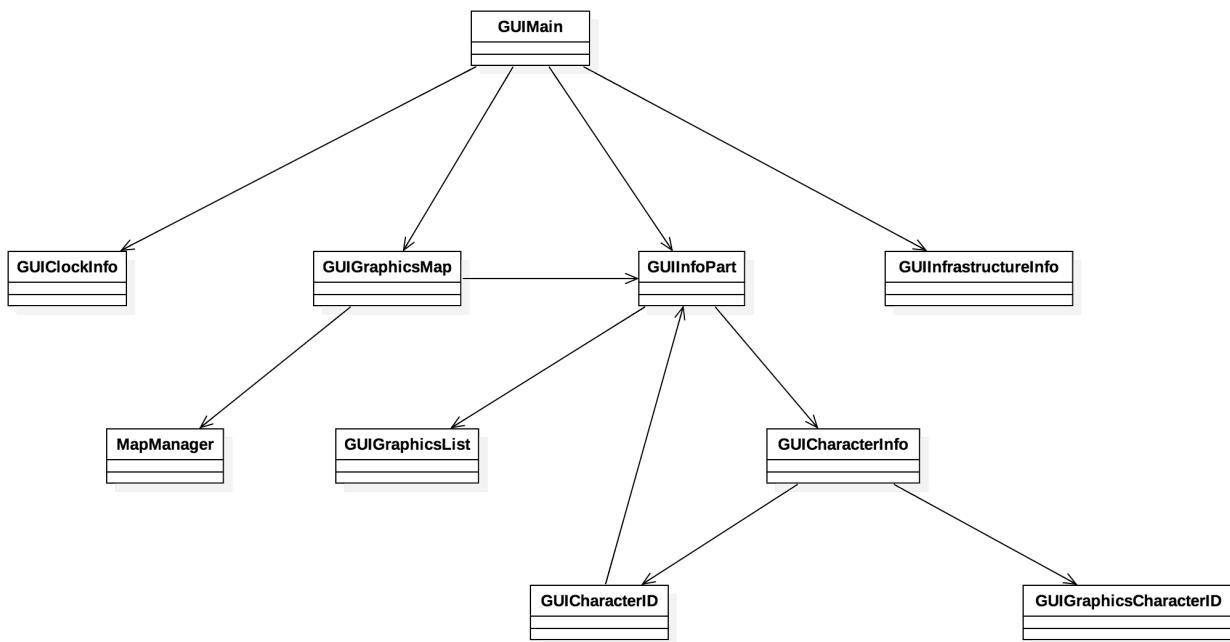


FIGURE 16 – Diagramme UML de l'IHM

Nous pouvons voir dans le diagramme ci dessus que l'ensemble des éléments graphiques sont rassemblé dans des classes principales, qui elles mêmes sont rassemblées dans une seule classe principale, qui

à pour but de construire l'IHM correctement.



FIGURE 17 – IHM du jeu

3.5.1 La Map

L'affichage de la map est effectué dans la classe "GUIGraphicsMap". L'affichage est géré par la méthode "paintComponent", dans laquelle la liste de chaque type d'infrastructure de la Map est parcourue. Pour chaque infrastructure, la méthode redimensionne et positionne l'image correspondant à l'infrastructure via la classe "MapManager". Ensuite la méthode parcourt la liste des personnages, et de la même manière, via la classe MapManager, redimensionne et positionne l'image du personnage au bonne endroit. Si l'utilisateur à sélectionné un personnage dans la liste des personnages, la méthode ajoutera un cercle rouge derrière le personnage en question.

3.5.2 Informations des personnages

La classe chargée de construire la partie de l'interface affichant la liste des personnages et les informations des personnages (plus d'informations dans le manuel d'utilisation) est la classe "GUIInfoPart". Nous pouvons voir que la classe "GUIGraphicsList", chargée d'afficher la liste des personnages, et la classe "GUICharacterInfo", chargée d'afficher les informations d'un personnage, sont relié à la classe "GUIInfoPart".

La classe "GUICharacterList", affiche la liste des personnages. Pour se faire, elle parcourt la liste des personnages, et pour chaque personnage, elle :

- Affiche son nom
- Affiche l'image correspondant à un personnage
- Affiche les 3 jauge : Emotion, Money et Family

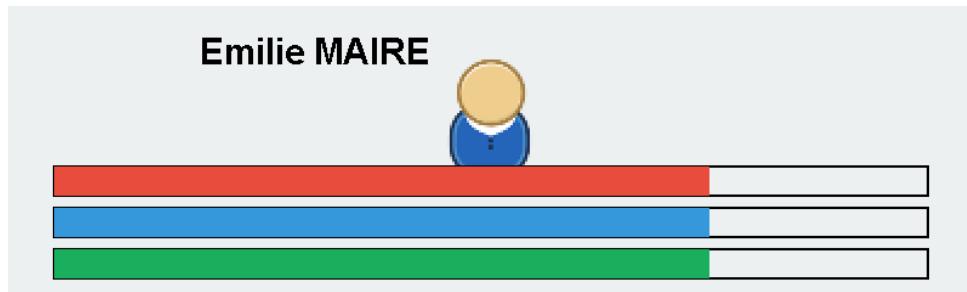


FIGURE 18 – Exemple d'affichage d'un personnage dans la liste

La classe "GUICharacterInfo" reçoit les données d'un des personnages que l'utilisateur a sélectionné, et en extrait les informations suivantes afin de les afficher :

- Son nom
- Son age
- Sa maison
- Son lieu de travail
- L'image correspondant à un personnage
- Les 3 jauge : Emotion, Money et Family
- Sa routine
- Des boutons affichant les graphiques liés à ce personnage
- Des boutons permettant l'ajout et la suppression d'action dans la routine

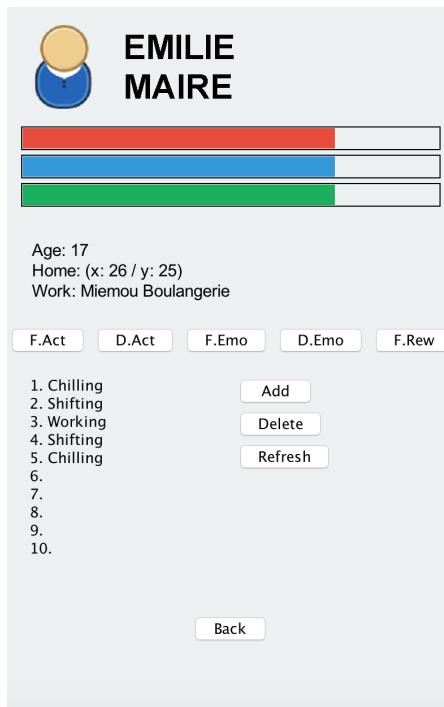


FIGURE 19 – Exemple d'affichage des informations d'un personnage

3.5.3 Information des infrastructures

La classe "GUIInfrastructureInfo" s'occupe d'afficher les informations de l'infrastructure sélectionnée au préalable par l'utilisateur (cf : manuel d'utilisation). Elle affiche les informations suivantes :

- Type de l'infrastructure
- Le nom de l'infrastructure
- L'horaire d'ouverture
- L'horaire de fermeture

3.5.4 Informations de l'Horloge

La classe "GUIClockInfo" s'occupe d'afficher sur l'IHM l'état actuel de l'horloge.

3.6 Logging : Log4J

Afin de suivre la bonne création de la carte de jeu et de la création de la population nous avons suivi les étapes clefs ces créations par un système de logging. Les principales étapes qui sont suivies sont :

- La lecture dans les fichiers CSV (pour les batiments et les personnages)
- Les bon positionnement des batiments dans la carte
- La bonne initialisation des routines

Pour ce faire nous utilisons l'outils de logging Log4J.

3.7 Phase de test

Pour garantire un parfait fonctionnement de notre logiciel nous avons mis en place une phase de test composée de 4 tests différents.

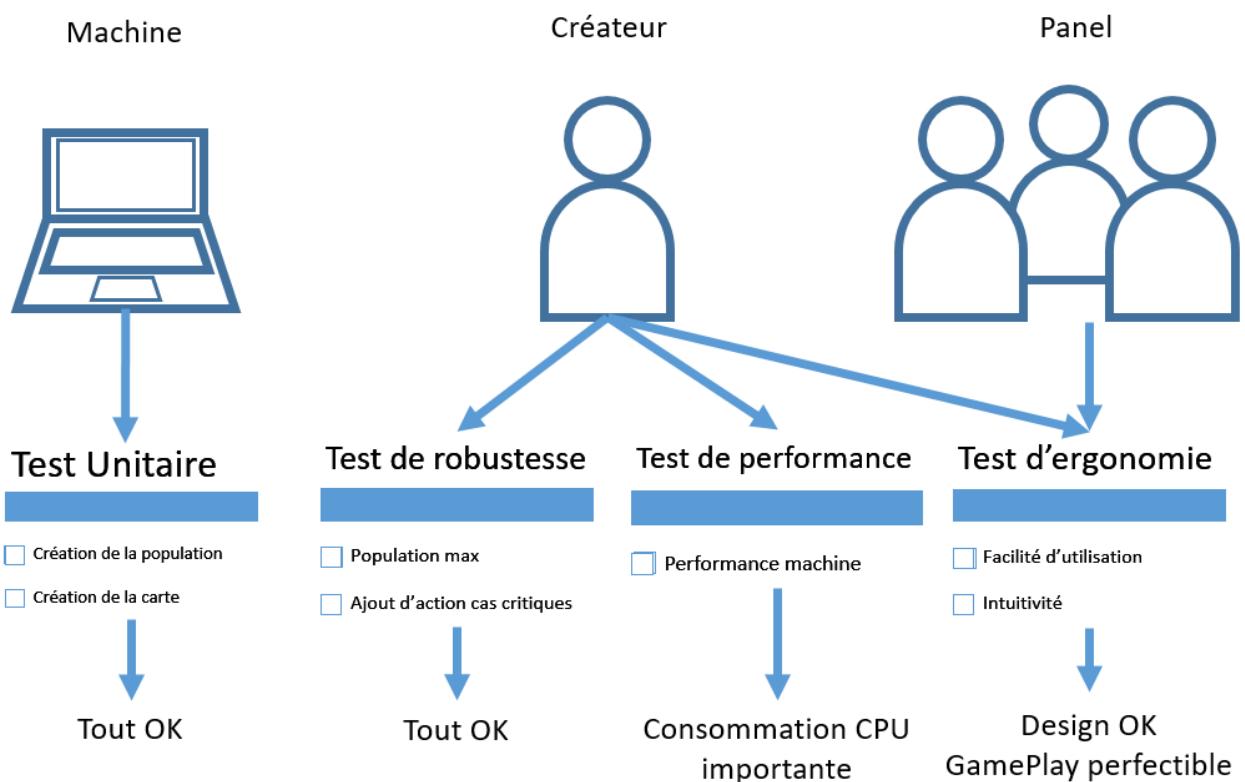


FIGURE 20 – Fonctionnement de notre phase de test

Test unitaire Grâce à l'outils JUnit nous réalisons des tests unitaires pour garantir :

- Création de la population (lecture CSV, tirage aléatoire, initialisation de la routine)
- Test de l'algorithme A* (simulation de plusieur trajet et vérification de l'existance du chemin résultat)

Test de robustesse Nous avons également réalisé des test de cas critiques (augmentation de la population au maximum, ajout d'actions au moment critique ...) Nous n'avons relevé aucun problème à ce niveau.

Test de performance Nous avons suivi l'évolution de la consommation CPU tout au long de projet. Ce suivi nous a permis de déceler un problème de rafraîchissement des images et de baisser la consommation de notre jeu. Cependant nous remarquons que notre jeu consomme beaucoup que ce soit en mode normale ou en mode autonome. Nous avons fait tourner le moteur indépendamment dans des conditions critique (population au maximum) et il s'avère que ce n'est pas le moteur qui cause cette forte consommation. Nous arrivons donc à la conclusion que le problème vient de l'interface graphique mais nous n'arrivons pas à identifier précisément le problème.

Test d'ergonomie Nous avons fait tester notre jeu à un panel (composé de quelques camarades de classe) pour obtenir des feedback sur le jeu. Tout les retours que nous avons collectés montre que le design de l'application est bien reçu par l'utilisateur. Cependant, certains retour on noté une fragilité du gamePlay de notre jeu.

3.8 Leaderboards

En fin de partie l'utilisateur à la possibilité d'enregistrer son score sur une plateforme web, et de consulter le classement des meilleurs score à Urban Life Simulator.

3.8.1 Le Site Web

Les classements aux différents niveaux de difficultés sont disponibles sur un site web hébergé, et accessible 24/24h.



FIGURE 21 – Les différentes pages accessibles du site web

Le site est composé de 5 pages :

- La page d'accueil
- Le classement du mode Easy
- Le classement du mode Normal
- Le classement du mode Hard
- Le classement du mode Pro

La page d'enregistrement des score est accessible seulement par un lien généré par le jeu en fin de partie, l'utilisateur pourra ensuite scanner un QR Code pour accéder à la page, ou cliquer sur un bouton qui le renverra sur la page si il ne possède pas de lecteur de code QR.

3.8.2 Principe du classement

Le principe du classement est de classer les meilleurs joueurs à Urban Life Simulator, voici les règles du classements de Urban Life Simulator :

- Les plus hauts scores se retrouveront en haut de la liste.
- Le classement ne retiens que les 10 meilleurs scores.
- Les scores en dessous du dizième meilleur score ne sont pas retenu, ou sont effacé lorsque le 10eme score est battu.
- En cas d'égalité de score dans le classement, le joueur ayant fait le score en premier occupe la place supérieur. Ainsi en cas d'égalisation d'un score, le joueur performant l'égalisation se verra attribué la place inférieur à tous les autres joueurs ayant déjà fait le même score auparavant.

3.8.3 Modélisation d'un classement

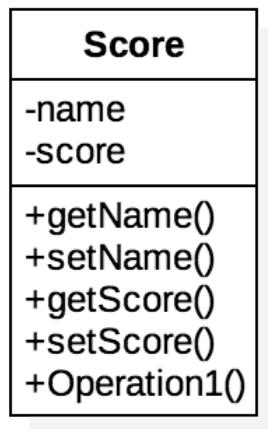


FIGURE 22 – Diagramme UML de la Classe Score en PHP

Un score est modélisé par une classe "Score" en PHP ayant pour attribue :

- Un nom
- Un score

La classe "Score" possède pour seules méthodes, les accesseurs et mutateurs des deux attribues précédants. Les 10 meilleurs scores sont ensuite ajouté dans un tableau. Le tableau est ensuite sérialisé dans un fichier sur le serveur.

3.8.4 Ajout d'un score dans un classement

L'ajout d'un score dans un classement est classique et se fait via un parcours du tableau contenant tous les scores par la fin, le parcours s'arrête quand le score du joueur est inférieur au prochain score. Lors du parcours tous les scores sont décalés vers le bas jusqu'à l'arrêt du parcours du tableau. Une fois que le parcours du tableau est arrêté, le score de l'utilisateur est inséré à la place correspondante.

4 Manuel Utilisateur

4.1 Déroulement d'une partie

Une partie commence à minuit le 1er janvier 2017.

4.1.1 Mode Normal

Au début d'une partie en mode normal, il est possible de choisir son niveau de difficulté entre :

- Niveau Easy : Un seul personnage sur la map
- Niveau Normal : Trois personnes sur la map
- Niveau Hard : Cinq personnes sur la map
- Niveau Pro : Quinze personnes sur la map (le maximum)

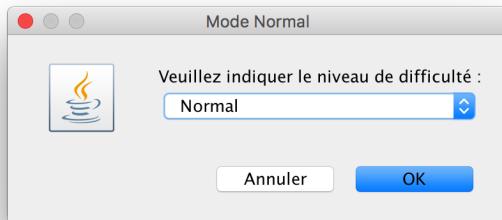


FIGURE 23 – Fenêtre de selection de niveau en mode normal

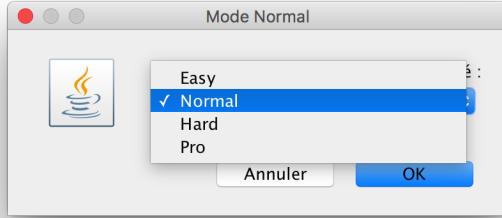


FIGURE 24 – Les différents niveaux en mode normal

L'objectif du joueur est de faire survivre sa population le plus longtemps possible en influent sur le comportement des personnages. Un personnage meurt lorsqu'une de ces jauge (Emotion, Money, Family) arrive à 0. La partie s'arrête quand il n'y a plus de personnages en vie.

4.1.2 Mode Autonome

En mode autonome le joueur ne peut influer sur le comportement des personnages. Le mode autonome est simulation de l'évolution d'une population dans un environnement urbain. L'utilisateur peut choisir le nombre de personnages présent dans la population lors de la simulation, au début de la partie. Le nombre de personnages en mode autonome peut aller de 1 à 60.

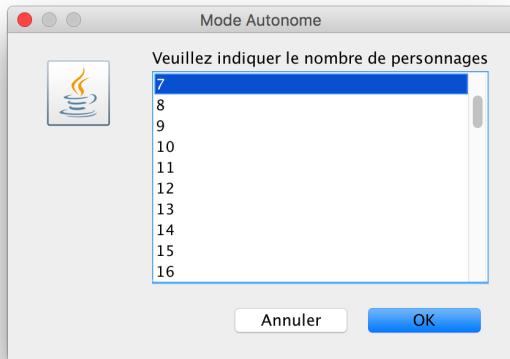


FIGURE 25 – La sélection du nombres de personnages en mode autonome

La partie, en mode autonome, ne se fini jamais. En effet lorsqu'une des jauge d'un personnage arrive à 0, ce personnage déménage et commence une nouvelle vie, jusqu'à arriver à vivre une vie stable.

4.2 Manuel de l'IHM

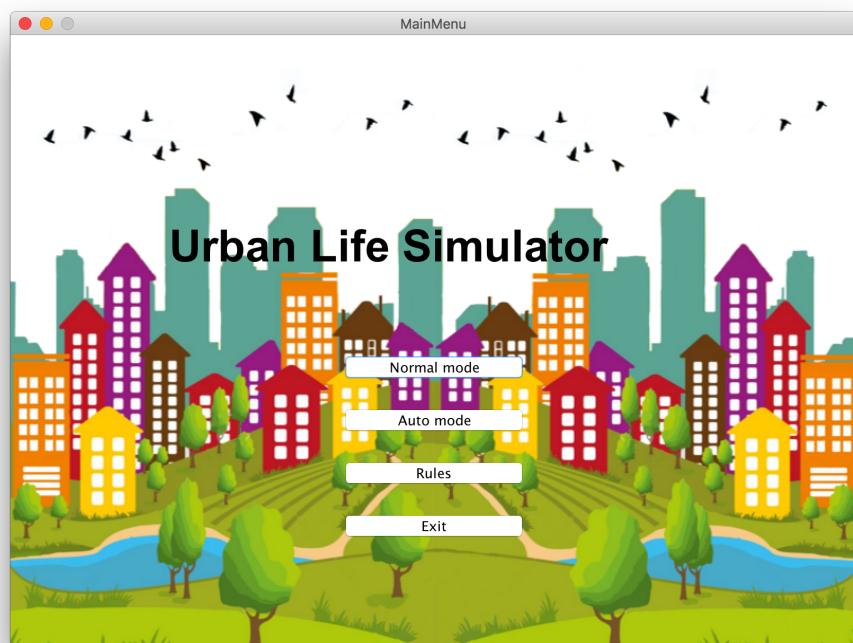


FIGURE 26 – Menu principal



FIGURE 27 – IHM du jeu

4.2.1 Information des Infrastructures

Type: Entertainment Name: Club de sport Opening: 6h30 Closing: 22h0

FIGURE 28 – Information des infrastructures

Pour afficher les informations d'une infrastructure, l'utilisateur peut double-cliquer sur n'importe qu'elle infrastructure sur la map.

4.2.2 Information de l'Horloge

Menu Play 00 : 17 - (01 / 01 / 2017) > >> >>>

FIGURE 29 – Information sur l'horloge

Cette partie, en plus de contenir les informations sur l'horloge rafraîchie automatiquement, contient 5 boutons : (Fig. 29)

- Bouton menu : permet de revenir au menu principal (Met fin à la partie)
- Bouton Play/Pause : Permet de stopper l'horloge et de la faire reprendre
- Bouton Speed : 3 niveau disponible permettant de régler la rapidité de l'horloge

4.2.3 La Map



FIGURE 30 – Affichage de la Map

La Map est un affichage de la situation de la population dans son environnement, à l'instant T, elle ne possède pas de bouton permettant de la contrôler.

4.2.4 Liste des personnages

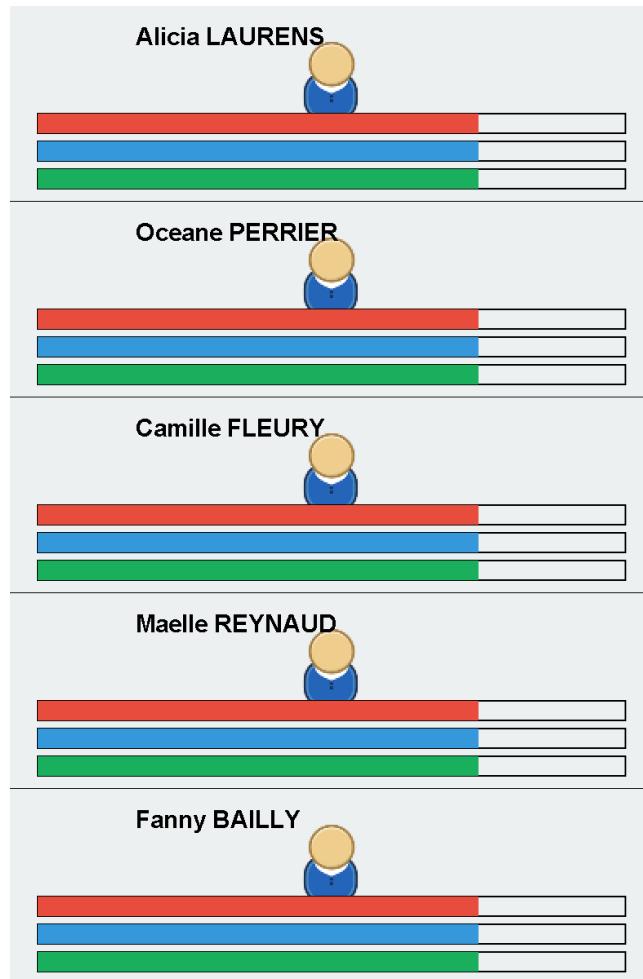


FIGURE 31 – Liste des personnages

Dans cette partie de l'interface, nous pouvons voir graphiquement l'état de chaque personnage de la population. Toutes les informations essentiels sont directement accessible visuellement, tel que :

- Le nom du personnage
- La jauge "Emotion" (en rouge)
- La jauge "Money" (en bleu)
- La jauge "Family" (en vert)



FIGURE 32 – Affichage de l'IHM avec les informations détaillé sur un personnage

Afin d'accéder à plus d'options et d'informations sur un personnage, il vous suffit de cliquer sur la case d'un personnage. Vous allez ensuite arriver sur l'interface de gestion d'un personnage qui remplace la liste des personnages.

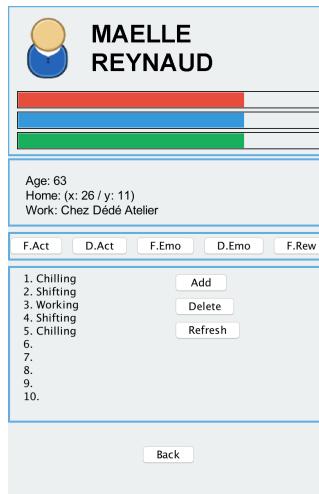


FIGURE 33 – Interface de gestion d'un personnage

L'interface de gestion d'un personnage est divisé en 4 parties : (Fig. 33)

- Les informations essentiels
- Les informations détaillées comprenant :
 - L'âge du personnage
 - Sa maison
 - Son lieu de travail
- Les boutons pour accéder aux statistiques
- La gestion de la routine comprenant :
 - La liste des actions dans la routine courante du personnage
 - Un bouton pour ajouter une action dans la routine

- Un bouton pour supprimer une action dans la routine du personnage
- Un bouton pour rafraîchir la liste des actions dans la routine courante du personnage
- Un bouton pour revenir à la liste des personnages

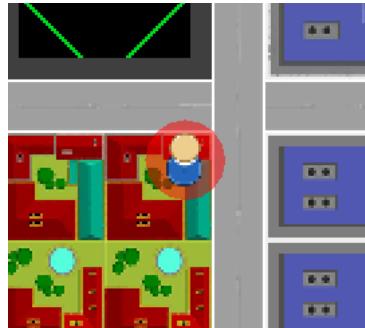


FIGURE 34 – Le personnage "Maelle Reynaud" démarqué par un cercle rouge

Lorsqu'un personnage est sélectionné dans la liste et que l'interface de gestion du personnage est activé, le personnage est démarqué sur la map pour pouvoir le suivre plus facilement. En effet, dans ce cas là, un cercle rouge est présent derrière l'image représentant le joueur, pour le repérer plus facilement.(Fig. 34)

4.2.5 Les statistiques

L'utilisateur peut accéder à 5 graphiques (généraient grâce à la bibliothèque JFreeChart) représentant l'évolution du personnage sélectionné, au cours du temps, en cliquant sur l'un des 5 boutons présent dans l'interface de gestion de personnages. Parmi les graphiques nous pouvons trouver, par ordre d'apparition :

- La répartition des actions du personnage
- La répartition des actions du personnage sur la journée en cours et celle de la veille
- L'évolution des jauge Emotion, Money et Family du personnage
- L'évolution des jauge Emotion, Money et Family du personnage sur la journée en cours et celle de la veille
- La répartition des récompense (récompense positive, négative et nulle) du personnage

4.2.6 La routine

Ajout d'action dans la routine



FIGURE 35 – Fenêtre d'ajout d'action d'un personnage

Pour accéder à l'interface d'ajout d'action dans la routine, l'utilisateur doit cliquer sur le bouton "Add" présent dans l'interface de gestion d'un personnage.

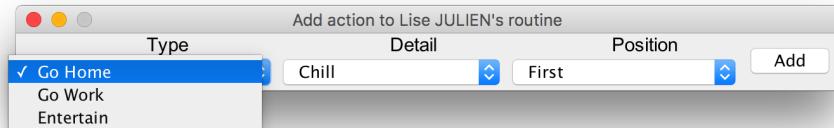


FIGURE 36 – Choix d'action disponible

L'utilisateur à le choix entre 3 type d'actions :(Fig. 36)

- Go Home
- Go Work
- Entertain



FIGURE 37 – Choix de detail lors du choix d'une action de type Go Home

Lorsque l'utilisateur choisit une action de type "Go Home", il peut spécifier si le personnage doit dormir, ou juste se reposer.(Fig. 37)

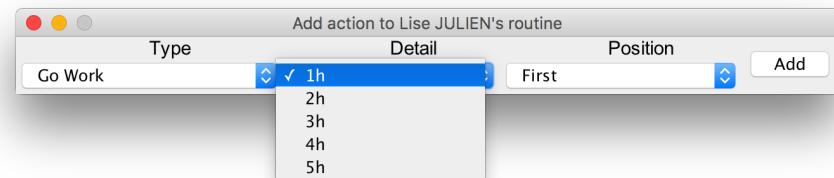


FIGURE 38 – Choix de detail lors du choix d'une action de type Go Work

Lorsque l'utilisateur choisit une action de type "Go Work", il peut spécifier combien de temps le personnage doit travailler.(Fig. 39)

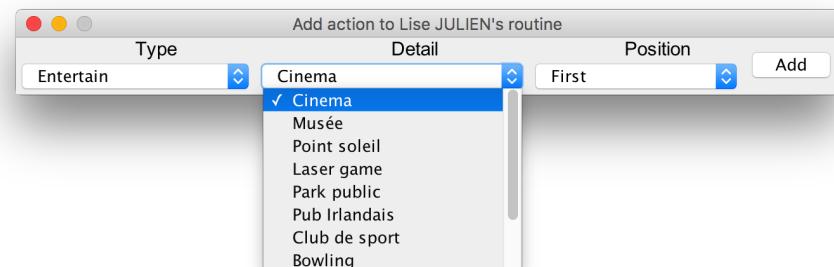


FIGURE 39 – Choix de detail lors du choix d'une action de type Entertain

Lorsque l'utilisateur choisit une action de type "Entertain", il peut spécifier où le personnage ira se divertir.(Fig. ??)



FIGURE 40 – Choix de la position de l'ajout de l'action

L'utilisateur à le choix, lors de l'ajout d'une action, de la position de l'action dans la routine courante (En premier ou en dernier)(Fig. 40)

Pour valider l'ajout, l'utilisateur doit appuyer sur le bouton "Add". Lors de l'ajout d'une action, les actions de déplacement sont automatiquement pris en compte.

Suppression d'action dans la routine

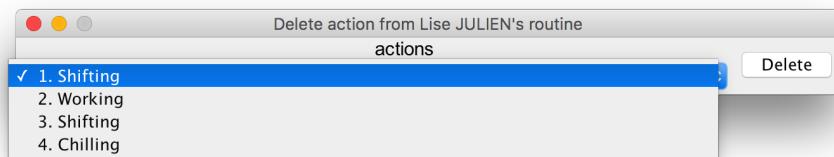


FIGURE 41 – Fenêtre de suppression d'action d'un personnage

Pour accéder à l'interface de suppression d'action dans la routine, l'utilisateur doit cliquer sur le bouton "Delete" présent dans l'interface de gestion d'un personnage.

L'interface de suppression d'action est composé de la liste d'action dans la routine courante du personnage sélectionné, par ordre d'execution, ainsi l'action "1" est la prochaine à être executé. L'utilisateur peut donc choisir précisement quelle action il souhaite supprimer dans la routine du personnage.(Fig. 41) Pour valider la suppression l'utilisateur doit appuyer sur le bouton "Delete".

4.3 Après la partie

Une fois la partie terminé (en mode normale), un score de l'utilisateur est calculer, et le programme propose à l'utilisateur d'enregistrer son score et de consulter le classement des meilleurs joueurs à Urban Life Simulator. Son score n'apparait dans le classement que si il fait partie des 10 meilleurs joueurs dans son niveau de difficulté.



FIGURE 42 – Fenêtre de fin de partie avec un exemple de QR Code

Pour enregistrer son score l'utilisateur peut scanner le QR Code unique que le jeu lui a généré (Il peut aussi directement cliquer sur le bouton en dessous du QR Code si il ne possède pas de lecteur de QR Code). (Fig. 42) Ce QR Code le redirige directement vers la plateforme de Leaderboard en ligne.

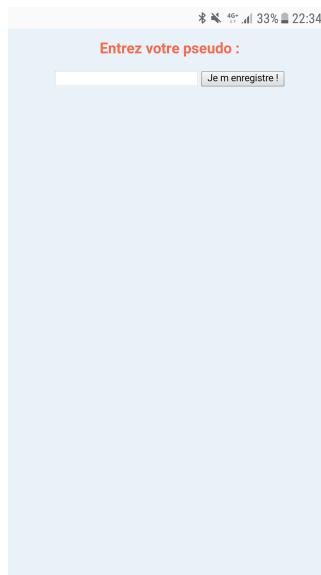


FIGURE 43 – Fenêtre d'enregistrement du score

La page d'enregistrement est composé d'un champ pour entrer son pseudonyme et d'un bouton pour valider. (Fig. 43) Pour l'exemple nous allons mettre "Projet Urbain" comme pseudonyme. Une fois valider le site web enregistre le score dans le classement correspondant au niveau de difficulté auquel l'utilisateur a joué, et redirige automatiquement vers la page du classement.



FIGURE 44 – Classement du niveau Pro

Sur cette page (Fig. 44) l'utilisateur peut se comparer à d'autres personnes ayant joué au jeu sur le même niveau de difficulté que le sien. Nous pouvons voir que lors de la partie notre joueur "Projet Urbain", a fait un score de 16 en niveau "Pro", et est donc le premier du classement au moment où il a joué.

5 Déroulement du projet

Dans cette section, nous décrivons comment la réalisation du projet s'est déroulée au sein de l'équipe de projet. La répartition des tâches, la synchronisation du travail et l'utilisation du temps seront abordées.

5.1 Répartition des tâches

Matthieu Vilain Quentin Gerard	
Reflexion autour du sujet	
Classes de données partie population	Classes de données partie ville
Moteur Mode normale/Autonome	IHM
Test unitaires	Logging
Prototypage algorithmes	Création du site
QRCode génération de l'URL	fonctions PHP
Rapport	
Présentation	

TABLE 3 – Répartition des tâches

5.2 Synchronisation du travail

Pour la synchronisation du travail nous avons utilisé l'outil de versionnage Git couplé à la plate-forme GitHub.

Nous avons voulu l'utiliser comme il peu être utilisé en entreprise : avec un environnement de production, un environnement de test et un environnement client (où le produit est dans une version fonctionnel). Pour cela nous avons utilisé le système de branche de Git selon le schéma suivant :

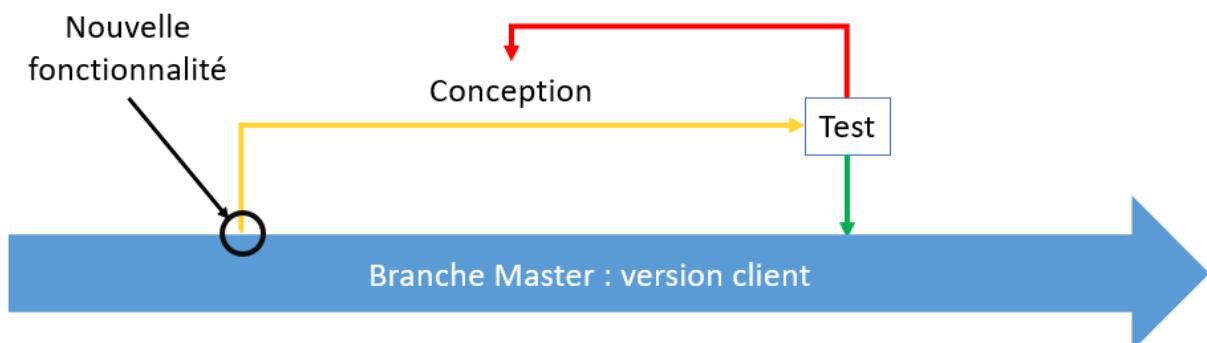


FIGURE 45 – Utilisation de Git

5.3 Utilisation du temps

Pour la gestion du temps, nous avons rapidement développé toutes les classes de données, les fonctionnalités principales et une IHM simplifié. Cela nous a permis d'avoir rapidement une base de test solide pour développer les fonctionnalités plus complexes et une meilleure IHM. Ensuite nous avons gardé un rythme constant pour ne pas prendre de retard.

Jan 8, 2017 – May 20, 2017

Contributions: **Commits** ▾

Contributions to master, excluding merge commits



FIGURE 46 – Répartition des commits Git au cours du temps

6 Conclusion

6.1 Problèmes Rencontrés

Lors de la réalisation de notre projet nous avons rencontré quelques problèmes comme :

- La gestion des conflits sur Git
- Implementation du design pattern Observer afin de synchronisé des éléments avec l'horloge

6.2 Améliorations Possible

Voici une liste non exhaustive d'améliorations possible du jeu :

- Un Mode de jeu mélangeant personnages jouables et personnages autonomes
- Influence de l'émotion des personnages autour d'un autre personnage sur son émotion
- Gestion des embouteillages
- Optimisation de la consommation du processeur
- Modification des propriété d'une infrastructure
- Modification des propriété d'un personnage
- Création manuelle d'un personnage avec des propriétés personnalisés

6.3 Ressenti sur le projet

Ce projet nous a permis de développer nos compétences en programmation orienté objet, mais aussi sur la gestion d'un projet long. Nous voyons maintenant comment organiser le développement d'une application et comment gérer les différentes tâches. Ce projet nous a aussi permis de travailler sur des notions qui nous intéressent.

Table des figures

1	Diagramme UML des composantes de la ville	5
2	Diagramme UML de la construction de la ville	6
3	Classe abstraite Character	7
4	Classe abstraite Character et ses classes filles	8
5	Architecture de création des NCharacters et de la population	9
6	Architecture de gestion des routines	10
7	Architecture de gestion des routines	11
8	A* : calcul de la pertinence d'un noeud	12
9	Statistiques pour chaque personnage	13
10	Classe DataCharacter	13
11	Equation de Q-Learning	14
12	Les étapes du Q-Learning	14
13	Un algorithme possible de Q-Learning	15
14	Prototype pour le Q-Learning	15
15	Prototype pour A*	16
16	Diagramme UML de l'IHM	16
17	IHM du jeu	17
18	Exemple d'affichage d'un personnage dans la liste	18
19	Exemple d'affichage des informations d'un personnage	18
20	Fonctionnement de notre phase de test	19
21	Les différentes pages accessible du site web	20
22	Diagramme UML de la Classe Score en PHP	21
23	Fenêtre de selection de niveau en mode normal	22
24	Les differents niveau en mode normal	22
25	La sélection du nombres de personnages en mode autonome	23
26	Menu principal	23
27	IHM du jeu	24
28	Information des infrastructures	24
29	Information sur l'horloge	24
30	Affichage de la Map	25
31	Liste des personnages	26
32	Affichage de l'IHM avec les informations détaillé sur un personnage	27
33	Interface de gestion d'un personnage	27
34	Le personnage "Maelle Reynaud" démarqué par un cercle rouge	28
35	Fenêtre d'ajout d'action d'un personnage	28
36	Choix d'action disponible	29
37	Choix de detail lors du choix d'une action de type Go Home	29
38	Choix de detail lors du choix d'une action de type Go Work	29
39	Choix de detail lors du choix d'une action de type Entertain	29
40	Choix de la position de l'ajout de l'action	30
41	Fenêtre de suppression d'action d'un personnage	30
42	Fenêtre de fin de partie avec un exemple de QR Code	31
43	Fenêtre d'enregistrement du score	31
44	Classement du niveau Pro	32
45	Utilisation de Git	33
46	Répartition des commits Git au cours du temps	34

Liste des tableaux

1	Taille de la population	7
2	Répartition des rewards	10

3	Répartition des tâches	33
---	----------------------------------	----

Références

- [1] L. M. Haas, E. T. Lin, and M. A. Roth. Data integration through database federation. *IBM Syst. J.*, 41(4) :578–596, 2002.