Université de Cergy-Pontoise

RAPPORT

pour le projet Génie Logiciel
Licence d'Informatique deuxième année

sur le sujet

Mettez le Sujet de Votre Projet Ici

rédigé par

Les auteurs ici

Mai 2015

Table des matières

Table des figures

Liste des tableaux

Remerciements

Les auteurs du projet voudraient remercier...

1 Introduction

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

Fonctionnalités Fonctionnalités du programme : Le joueur aura plusieurs actions possibles afn d'infuer sur l'évolution de la ville. Tout d'abord il pourra agir sur le temps en l ?accélérant ou en le mettant en pause. Ensuite il pourra accéder à des informations sur les personnages comme : leurs informations de bases, leur historique ou leurs objectifs directs (exemple : ce personnage se rend à la piscine). Ces informations pourront être changées par l'utilisateur et il pourra ainsi renommer un personnage, le faire déménager ou le faire rentrer chez lui par exemple. De même pour les bâtiments, l'utilisateur pourra accéder à ses informations et les modifer. Il pourra donc par exemple modifer les horaires d'ouverture d'un lieu ou faire varier son nombre d'utilisateur maximum. Le joueur pourra donc modifer à sa guise ses informations et voir ce que ces modifcations apportent de bon ou de mauvais sur à population de la ville.

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

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 personnages et sera la principale source de baisse d'émotion
- Les Divertissement permettent aux personnages lorsqu'ils ne dorment pas de regagner de l'émotion

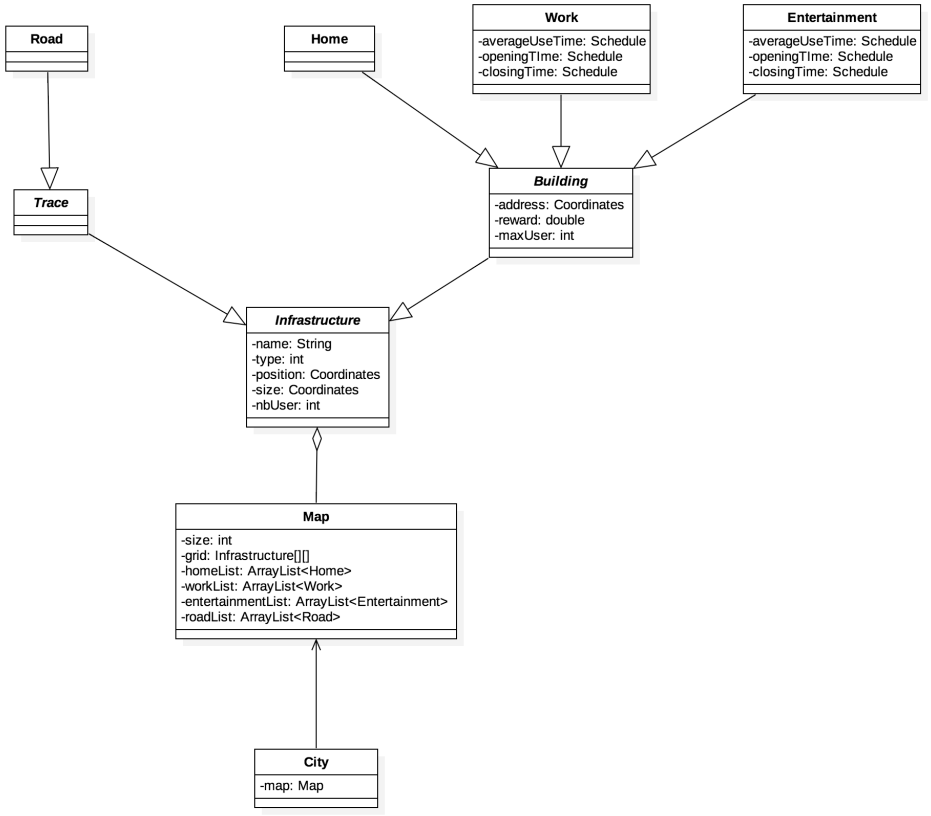

FIGURE 1 – Diagramme UML des composantes de la ville

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'entré dans le bâtiments 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 personnages d'utiliser un bâtiments lorsque celui ci à 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.

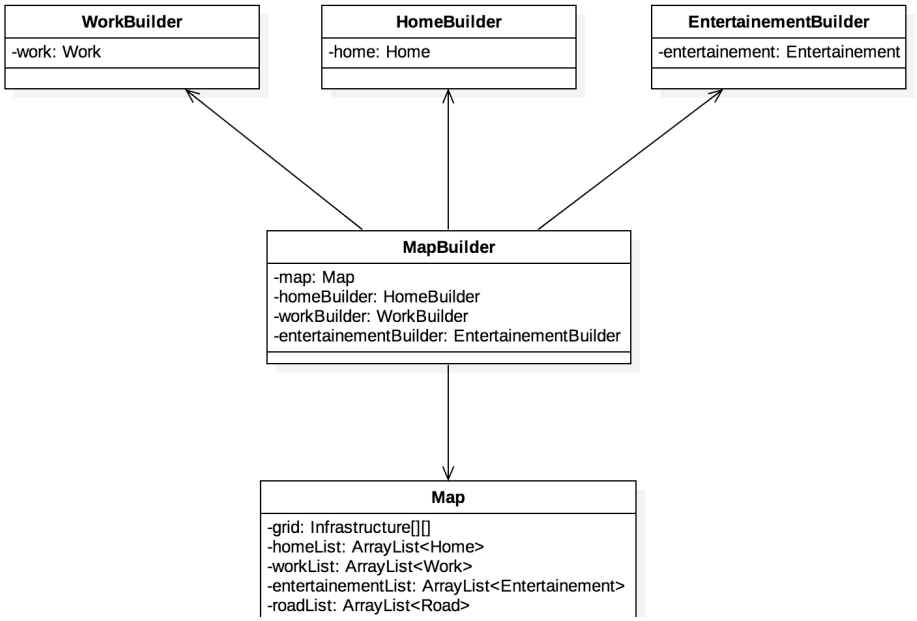

FIGURE 2 – Diagramme UML de la construction de la ville

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 depend 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'evolution 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 batiments de travail) et 60 en mode autonome (car 60 maisons) Si l'on décide d'aumenter 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 des ralentissement de l'interface graphique mais le moteur de jeu est parfaitement capable de faire tourné 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 age 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

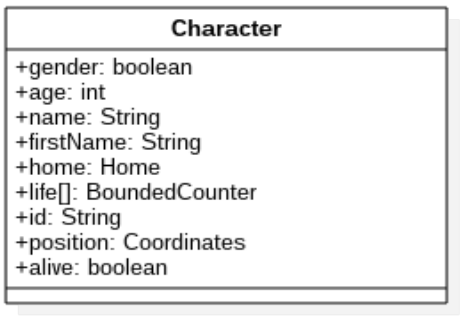

FIGURE 3 – Classe abstraite Character

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'age du personnage est simplement choisit 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 garantie 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 jauges contenant :
- une jauge d'émotion
- une jauge d'argent
- une jauge de fatigue

Ces jauges peuvent varier de 0 à 100. Ces jauges représentent les critères de vie du personnage. Elle est initialisé à 75 en début de partie mais elle variera en fonctions des actions des personnages. Si l'une des jauges 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 premiere, NCharacter, représente les personnages utilisés dans le mode de jeu normal. La seconde, QCharacter, représente les personnages utilisés dans le mode de jeu autonome.

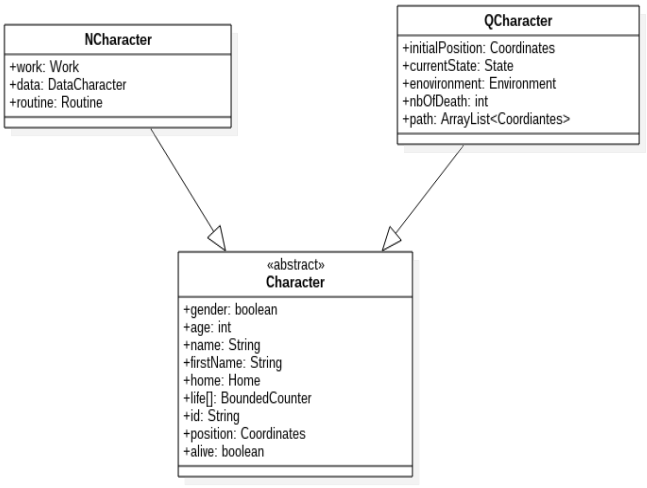

FIGURE 4 – Classe abstraite Character et ses classes filles

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 debut 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 réaparition pour leur permettre d'y retourner en cas de besoin de sommeil. Ils possèdent également un environement 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 etat 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 statisqtiques avec . Enfin, un QCharacter possède une liste de coordonnées qui lui permettra de retourner à sa maison si il à un besoin en sommeil.

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

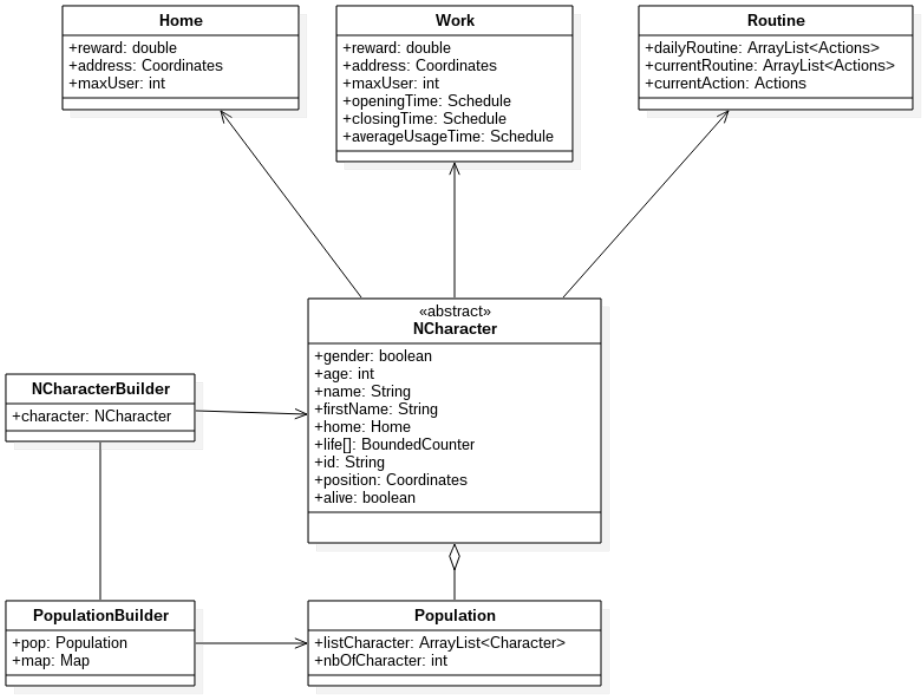

FIGURE 5 – Architecture de création des NCharacters et de la population

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 meme architecture pour créer la population de QCharacter dans le mode autonome.

3.2 Le moteur de jeu : mode normale

3.2.1 Gestion des jauges des personnages

Les jauges 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 jauges est totalement dynamique, elle se fera automatiquement en fonction des actions faites par le personnage. Certaines actions vont apporter un bonus différent sur les jauges du personnage alors que d'autres vnt apporter des malus. Pour certaines actions le bonus est constant (par exemple dormir apportera toujours +30 en fatigue) 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 fatigue 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	fatigue	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.

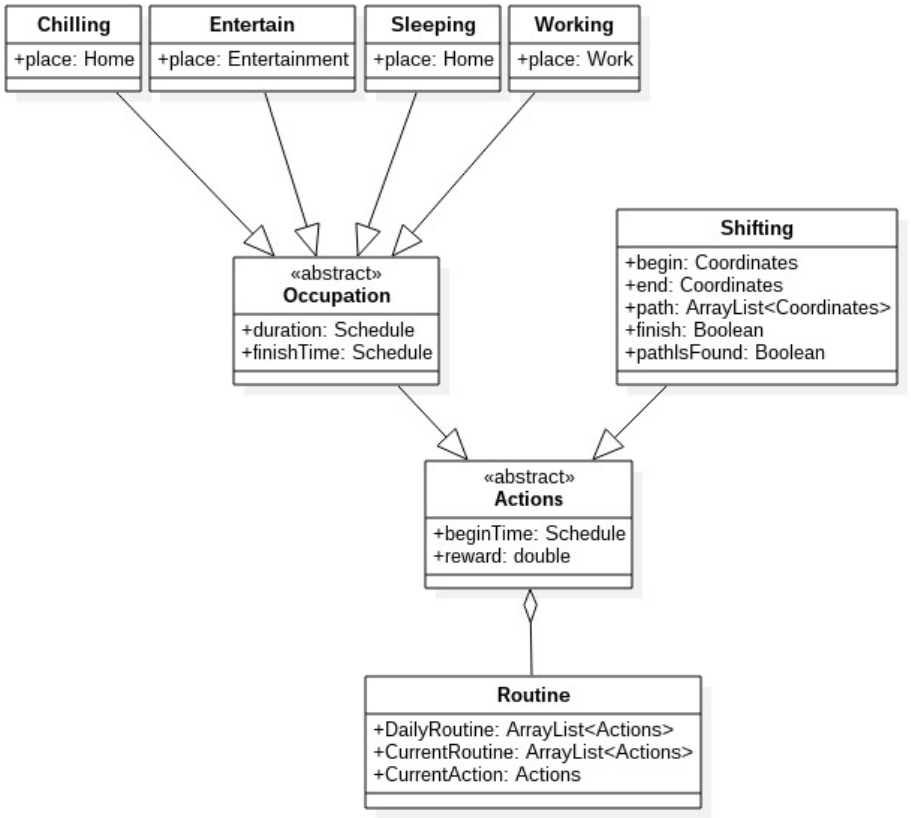

FIGURE 6 – Architecture de gestion des routines

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 a 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

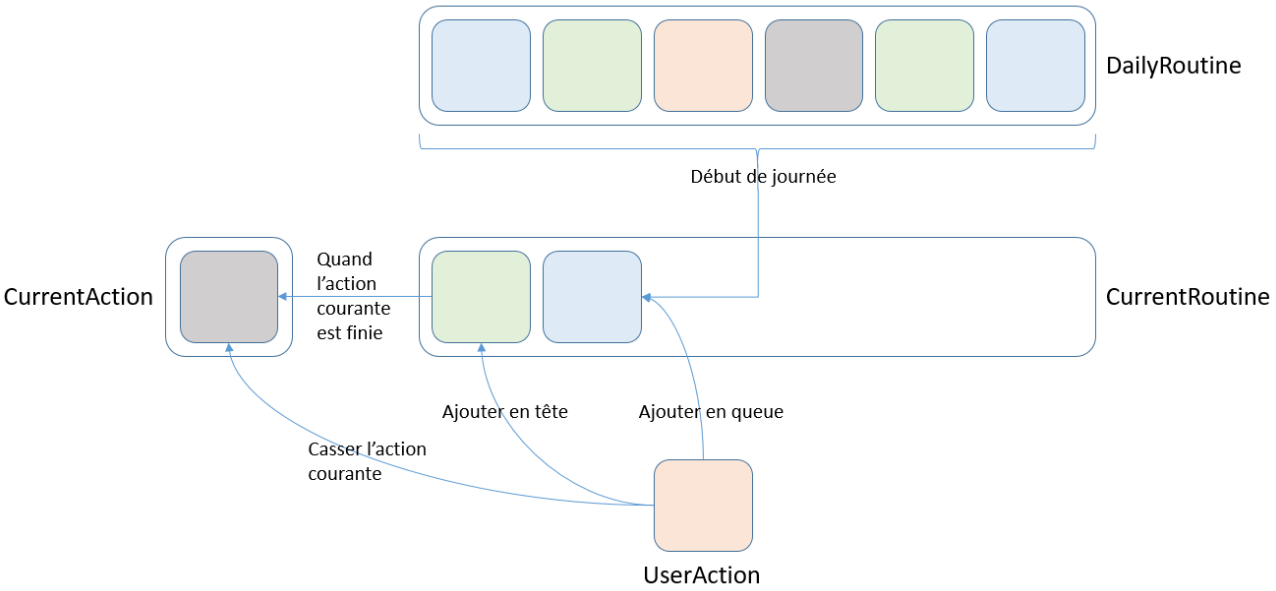

FIGURE 7 – Architecture de gestion des routines

La liste d'actions communes a chaque journées est stocké dans la liste DailyRoutine. A chaque début de journée, on ajoute toutes les actions de cette liste a celle de la journée courante, CurentRoutine. 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.

3.3 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 séléctionné 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.

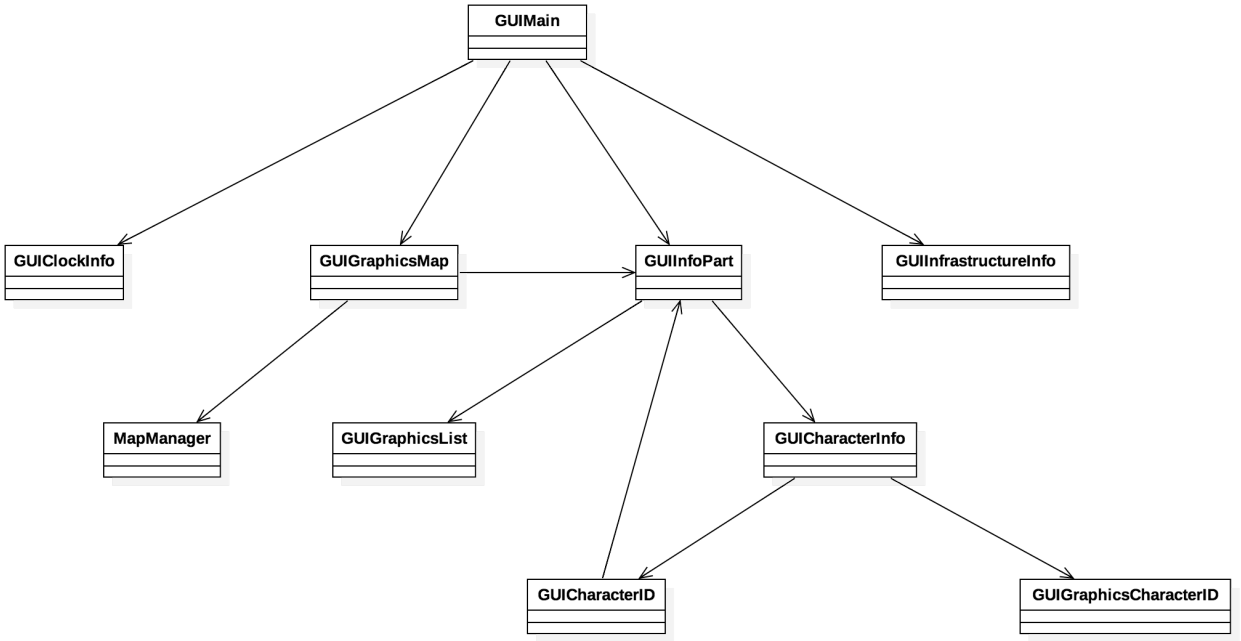

FIGURE 8 – Diagramme UML de l'IHM

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 9 – IHM du jeu

3.3.1 La Map

L'affichage de la map est effectué dans la classe "GUIGraphicsMap". L'affichage est géré par la methode "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 parcours 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.

3.3.2 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 personnages, sont relié à la classe "GUIInfoPart".

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

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

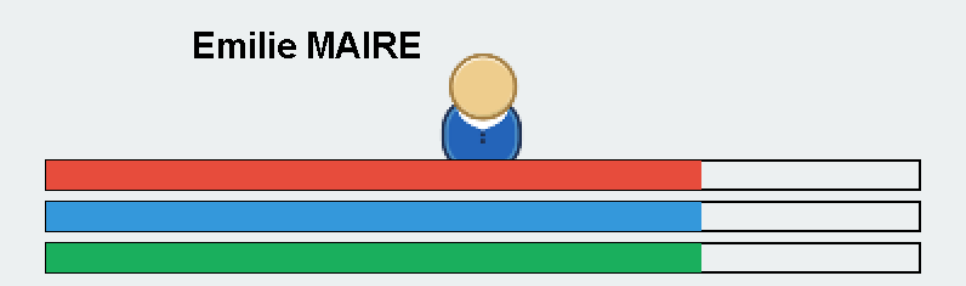

FIGURE 10 – Exemple d'affichage d'un personnage dans la liste

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

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

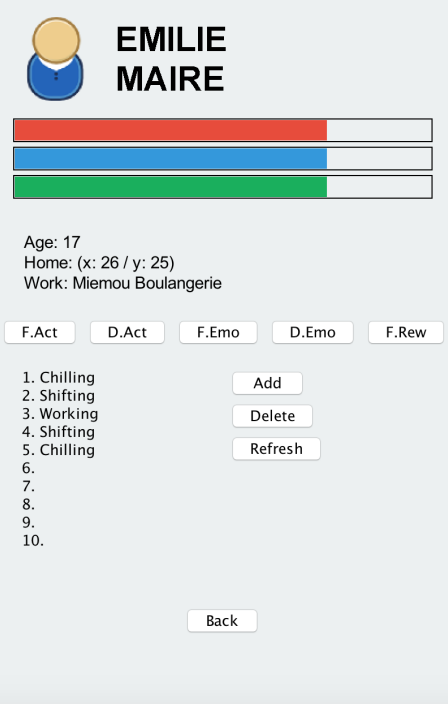

FIGURE 11 – Exemple d'affichage des informations d'un personnage

4 Manuel Utilisateur

Cette section est dédiée au manuel utilisateur.

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.

6 Conclusion

Dans cette section, nous résumons la réalisation du projet et nous présentons également les extensions et améliorations possibles du projet.

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.