Projet Technologie Objet - Groupe MN 2

Rapport n°1 : City-Builder

François Lauriol, Priscilia Gonthier, Nicolas Catoni, Alice Devilder,
Clément Delmaire-Sizes, Eya Amrach, Yael Gras

9 Avril 2022

Table des matières

Introduction

L'objectif de notre projet est de créer un "City-Builder". Ce sera un jeu de construction dans lequel le joueur pourra choisir l'emplacement des bâtiments et les plus importantes caractéristiques de gestion de ville, comme les salaires, les taxes et les priorités de travail. Il pourra ajouter des rues à la ville, des bâtiments, des commerces, des parcs et des monuments. Chaque élément de la ville coûtera une somme d'argent spécifique. Par l'ajout de commerces, des taxes gagnées pourront être utilisées pour garantir l'élargissement de la ville. La ville se développera en conséquence des choix réalisés.

Une carte graphique d'une forêt sera le point de départ, on pourra ajouter des rues à la ville, des bâtiments, des commerces, des parcs et des monuments. Chaque élément de la ville coûtera une somme d'argent spécifique. Par l'ajout de commerces, des taxes gagnées pourront être utilisées pour garantir l'élargissement de la ville.

1 Principales Fonctionnalités

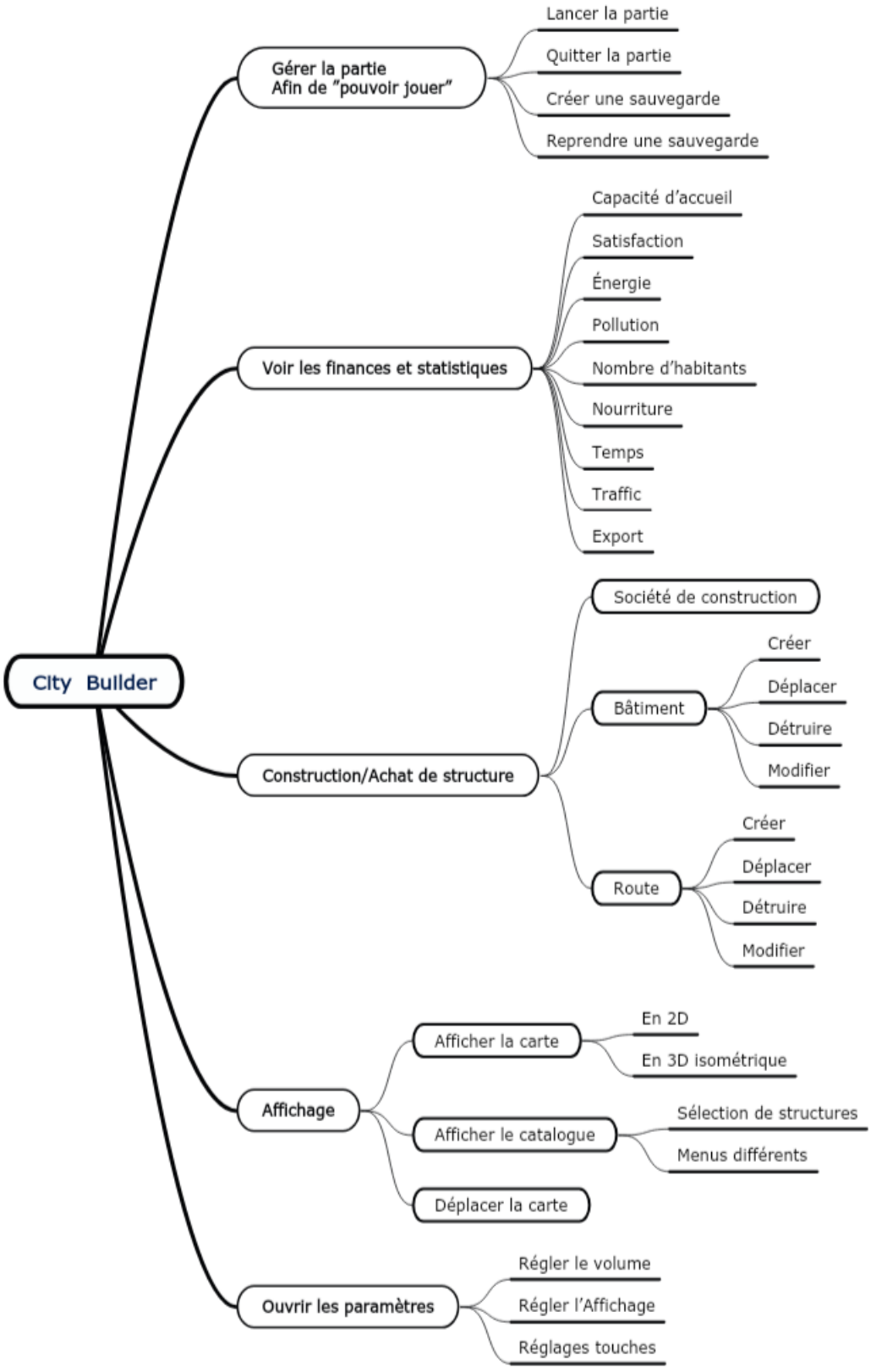

FIGURE 1 – Fonctionnalités principales

2 Découpage en paquetage

2.1 Le package "Batiments"

Le package "Batiments" contient les classes abstraites "Commerce", "Habitation", "Hopital" et "Industrie" qui sont des sous-classes de la classe abstraite "Batiment".

Chaque bâtiment est créé sur une carte et à un emplacement précis. Nous avons pour cela défini une classe "Carte" et une classe "Parcelle" dans la couche supérieure, c'est-à-dire dans le package "src". De plus, un bâtiment a une attractivité (cf fonctionnalité) et un niveau différents compris entre 1 et 3, ainsi qu'un temps et un coût de construction.

Plus précisément, dans la classe "Batiment" sont définies les méthodes d'accès aux attributs de la classe ainsi qu'aux propriétés liées à l'emplacement du bâtiment.

La classe abstraite "Commerce" est une sous-classe de "Batiment". Elle possède en plus, les attributs liées aux emplois tels que le nombre courant d'employés, la capacité maximale d'employés et la valorisation de l'emploi (cf fonctionnalités). Un commerce a également un nom afin de différencier leur rôle et leur attribuer des caractéristiques propres à leur fonction. Par exemple, un commerce "alimentaire" n'aura pas le même quota d'employés qu'un commerce "loisir". Le niveau du commerce va également définir certains attributs. D'où la présence d'une méthode setCommerceNiveau(int niveau) qui est appelée dans le 2ème constructeur de la classe.

Ensuite, la classe abstraite "Hopital" introduit de nouveaux attributs notamment associés aux patients qui circuleront entre la ville et ce bâtiment. En effet, nous avons créé des attributs correspondant à la capacité d'accueil des patients, le nombre de patients courant mais également le nombre d'ambulances total et le nombre d'ambulances courant, ayant un périmètre d'action défini et servant à transférer les patients vers l'hôpital.

Tout comme la classe "Commerce", le niveau de l'hôpital définit certains attributs. De plus, il y a l'implémentation des méthodes liées aux ambulances telles departAmbulance et retourAmbulance. La méthode définissant le départ d'un patient, c'est-à-dire son temps de guérison sera créée ultérieurement (lors qu'un échelle de temps sera mise en place).

La classe abstraite "Industrie" permet d'avoir des ouvriers qui pourront par la suite construire des bâtiments. Cette classe a une structure similaire à celle de la classe "Commerce".

Enfin la classe "Habitation" permet de créer des habitations avec un certain nombre d'habitant (caractérisé par l'attribut nbHabitant) et une capacité d'accueil définie. Une habitation possède également une capacité d'accueil différente en fonction du niveau (cf la fonction setHabitationNiveau(int niveau)).

Ce package "Batiments" sera complété lors des itérations qui suivent.

2.2 Le package "Menus"

Ce package contient tous les éléments nécessaires à la créations des menus principaux d'un jeu qui sont : le menu à l'ouverture du jeu, le menu paramètre, et le menu de lancement d'une partie. Il contient aussi une classe qui lance cet affichage pour pouvoir tester que chaque éléments fonctionne correctement.

Tous les menu héritent d'une classe abstraite Menu car ils sont tous des sous-type de JPanel qui implémentes la même interface (ie "ComponentListener").

Dans chaque menu, des Commandes sont ajoutées qui sont aussi des sous-types de JButton de manière à avoir une meilleure clarté dans le code et la structure de ce dernier. Les relations entre l'interface commandes et les commandes qui réalisent cette interface ont de grande chance de changer car cela n'est pour le moment pas une manière pertinente de faire ces boutons.

Pour finir, la partie graphique du menus n'est que provisoire ainsi le fond des menus et les placements ainsi que la forme des boutons vont certainement changer par la suite.

3 Diagramme de Classe

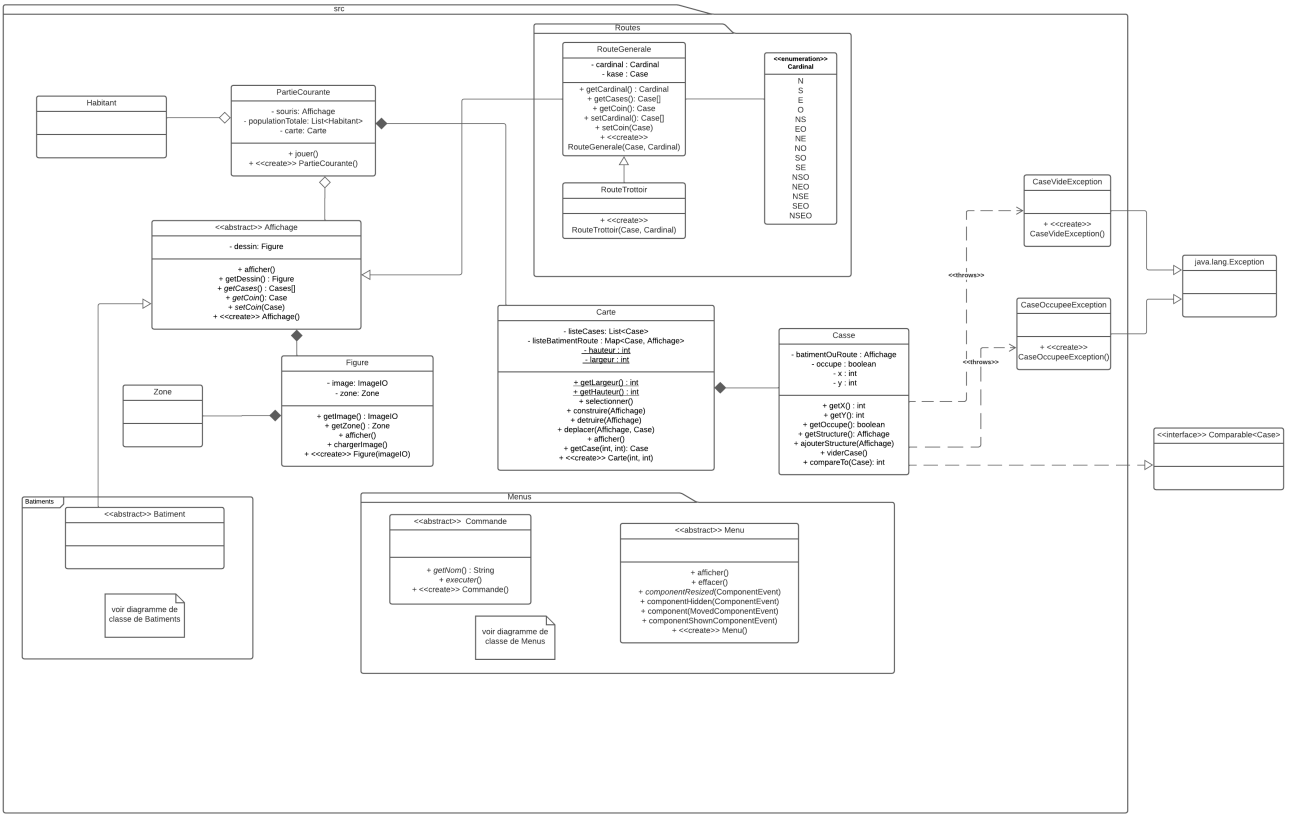

FIGURE 2 – Diagramme UML du City Builder

FIGURE 3 – Diagramme UML du package Batiments

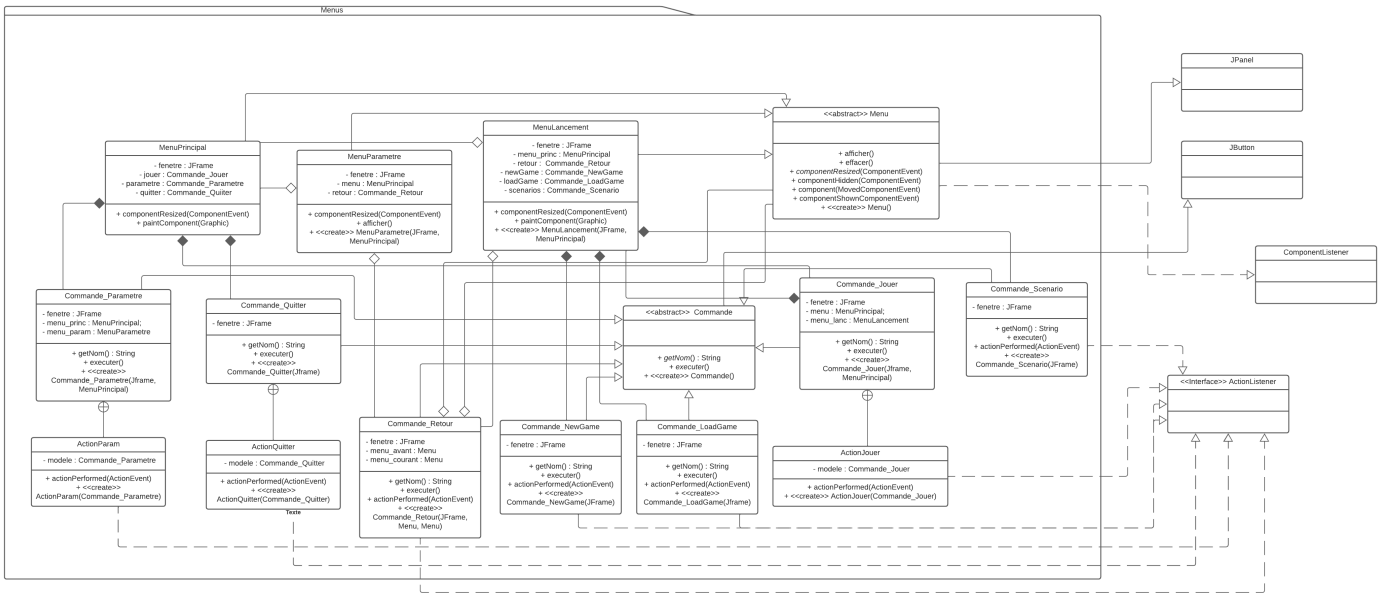

FIGURE 4 – Diagramme UML du package Menus

4 Les différents choix de conception et réalisation, les problèmes rencontrés et les solutions apportées

4.1 Création des bâtiments

Nous avons opter pour une classe abstraite pour créer la classe principale "Batiment". La prise en compte de tous les paramètres et scénarios possible était un peu compliqué. Nous avons donc choisi de faire des sous-classes de "Batiment".

4.2 Affichage des menus

Nous avons eu certains problème sur l'affichage des menus dû au choix d'utiliser des JPanel pour chacun. Cela créait un soucis avec l'utilisation de layout car chaque JPanel devait se superposer pour que quand il y en a un qui s'affiche, il soit bien sur l'entièreté de l'écran. De plus, certaines spécificités de swing ont causé quelques problèmes dans le passage d'un menu à un autre mais nous avons réussi à régler le problème.

4.3 La carte

Nous avons fait le choix de centraliser les informations dans une classe Carte. C'est elle qui contiendra les différents bâtiments et routes construits dans lors d'un jeu. C'est par cette classe que l'on passera pour construire, détruire, déplacer, afficher quelque chose.

Nous avons fait le choix de créer une classe Case. Et la carte est une liste de ces cases dans lesquelles se trouvent les bâtiments et routes. Nous avons aussi décider de créer une liste des bâtiments et routes afin de simplifier le parcours de la carte lors de l'affichage de celle-ci.

4.4 Création des routes

Nous avons fait le choix d'une classe route générale qui contient toutes les méthodes, et chaque route différente étendra cette classe pour pouvoir faciliter l'affichage.

4.5 Problèmes au niveau de la compilation

Nous avons des problèmes au niveau de la compilation des bâtiments et des routes car ils ne trouvent pas la classes Case et Carte (même après les imports). Nous n'avons pas encore réussit à résoudre ce problème, ces classes ne sont donc pas présentes dans la source en .jar du svn.

5 Organisation de l'équipe et mise en œuvre des méthodes agiles

Nous avons tout d'abord lors d'une réunion, dans l'objectif d'être agile, listé les fonctionnalités essentielles du jeu afin de nous répartir les différentes implémentations. Nous en avons conclu qu'une personne devait gérer les menus, une autre les routes et la carte, 2 personnes devaient d'occuper des l'affichage et des figures, 2 personnes devaient s'occuper des bâtiments et une personne du diagramme UML et de l'architecture générale. Nous avons attribué des rôles à chacun.

Nous avons ensuite réalisé d'autres réunions afin de regarder l'avancement et les difficultés de chacun et de pouvoir s'entraider.

Dans l'objectif des méthodes agiles, et plus particulièrement le fait de montrer que notre projet avance, nous avons fait le choix d'avoir une personne qui débute l'interface utilisateur pour le lancement d'une partie pendant que le reste de l'équipe développe la partie en elle-même. Cela permet d'avancer rapidement sur le jeu en lui-même mais aussi de pouvoir montrer au client que l'on avance. En effet, la partie fonctionnelle du jeu est loin d'être créée car il y a beaucoup de petite chose à faire avant de pouvoir mettre en relation tous les objet créés. Ainsi avoir un menu déjà fonctionnel permet au client d'avoir un premier aperçu comme un utilisateur qui découvrira le jeu et de discuter des choix graphiques que nous devons faire pour les différents éléments du City-Builder.