

Projet Technologie Objet - Groupe MN 2

Rapport : City-Builder

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

25 mai 2022

Table des matières

Introduction	2
1 Principales Fonctionnalités	3
2 Découpage en paquetage	5
3 Diagramme de Classe	8
4 Les différents choix de conception et réalisation, les problèmes rencontrés et les solutions apportées	17
5 Organisation de l'équipe et mise en œuvre des méthodes agiles	18

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.

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.

Notre jeu s'appelle "Utopia".

1 Principales Fonctionnalités

1.1 Répartition des différentes fonctionnalités

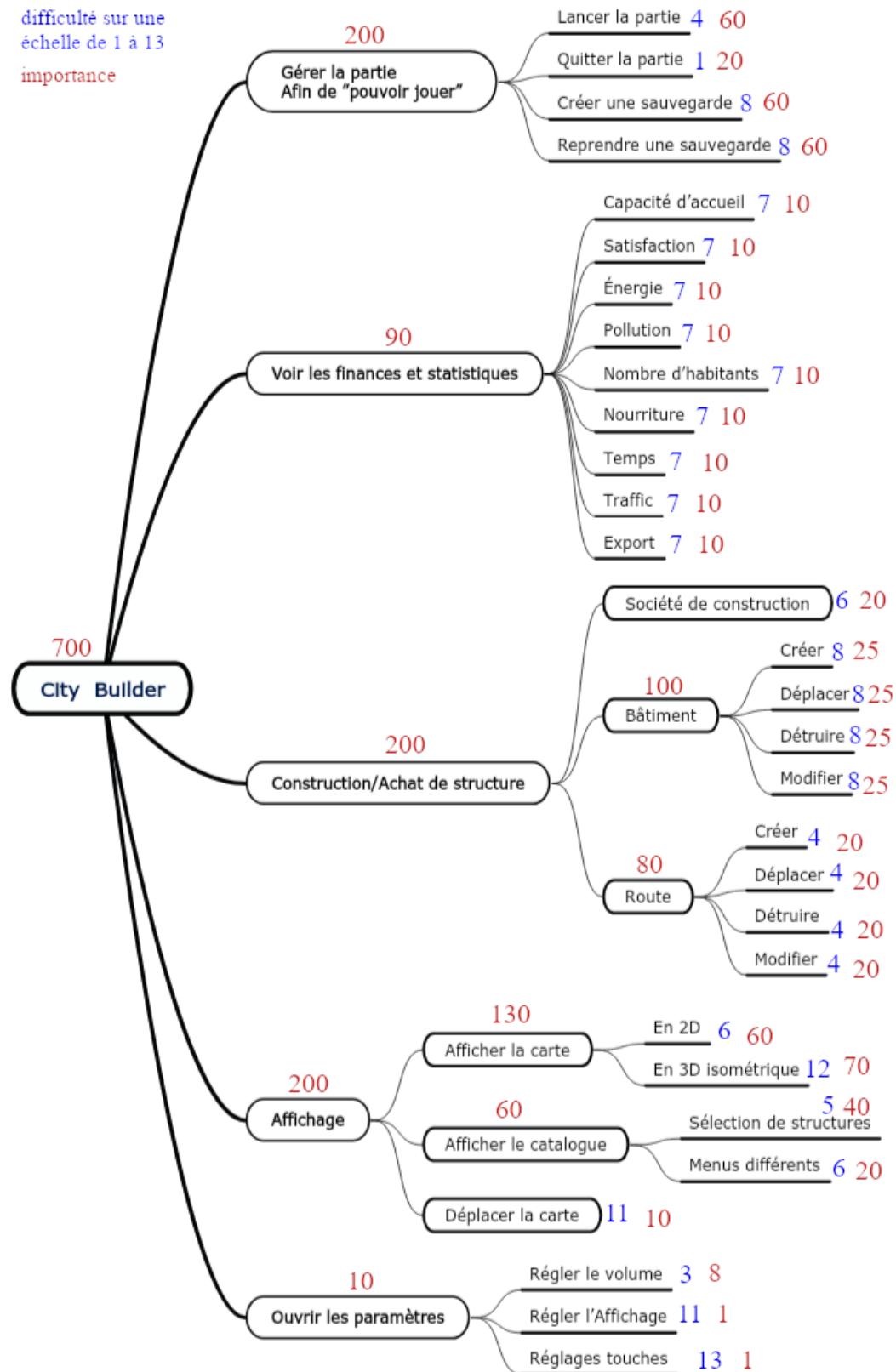


FIGURE 1 – Répartition des fonctionnalités principales

1.2 Réalisation des fonctionnalités

User-stories	État de réalisation	Itération
Lancer la partie	Fonctionnel	2
Quitter la partie	Fonctionnel	1
Créer une sauvegarde	Fonctionnel	2
Reprendre une sauvegarde	Fonctionnel	Back-end : 2 et front-end : 3
Ajouté en plus		
Renommer une sauvegarde	Fonctionnel	Back-end : 2 et front-end : 3
Supprimer une sauvegarde	Fonctionnel	Back-end : 2 et front-end : 3
Mettre une partie en pause	Fonctionnel	2
Reprendre une partie suite à une pause	Fonctionnel	2

TABLE 1 – Avancement de l'épic « Gérer la partie »

User-stories	État de réalisation	Itération
Capacité d'accueil	Fonctionnel en back-end	2
Satisfaction	Commencé	3
Énergie	Commencé en back-end	3
Pollution	Non implémenté	
Nombre d'habitants	Fonctionnel pour une version 1	Back-end : 2 et front-end : 3
Nourriture	Non implémenté	
Temps	Fonctionnel	3
Traffic	Non implémenté	
Export	Non implémenté	
Ajouté en plus		
Finances	Fonctionnel	Back-end : 2 et front-end : 3

TABLE 2 – Avancement de l'épic « Voir les finances et statistiques »

User-stories	État de réalisation	Itération
Société de construction	Non implémenté	
Créer un bâtiment	Fonctionnel	Back-end : 1 et front-end : 3
Déplacer un bâtiment	Fonctionnel en back-end	2
Détruire un bâtiment	Fonctionnel	Back-end : 1 et front-end : 2
Modifier un bâtiment	Non implémenté	
Créer une route	Fonctionnel	Back-end : 1 et front-end : 3
Déplacer une route	Non implémenté	
Détruire une route	Fonctionnel	Back-end : 1 et front-end : 3
Modifier une route	Non implémenté	

TABLE 3 – Avancement de l'épic « Construction/Achat de structure »

User-stories	État de réalisation	Itération
Afficher la carte en 2D	Fonctionnel pour une version 1	2 et 3
Afficher la carte en 3D isométrique	Non implémenté	
Afficher le catalogue et sélectionner des structures	Fonctionnel	3
Afficher le catalogue avec des menus différents	Fonctionnel pour une version 1	3
Déplacer la carte	Fonctionnel pour une version 1	2

TABLE 4 – Avancement de l'épic « Affichage »

User-stories	État de réalisation	Itération
Régler le volume	Fonctionnel	2
Régler l'affichage	Fonctionnel pour une version 1	1
Régler les touches	Non implémenté	

TABLE 5 – Avancement de l'épic « Ouvrir les paramètres »

2 Découpage en paquetage

2.1 Le package "batiments"

Le package "batiments" contient les classes "CasernePompier", "Commerce", "Ecole", "Habitation", "Hopital", "Hotel", "Industrie", "Mairie", "PoleEmploi", "Police" et "Usine" qui sont des sous-classes de la classe abstraite "Batiment" ainsi que la classe "Parcelle" qui permet de positionner les bâtiments sur la carte.

Chaque bâtiment est créé sur une carte et à un emplacement précis. Nous avons pour cela défini une classe "Carte" (situé dans la package "carte") et une classe "Parcelle". 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 "CasernePompier" défini, en plus, des attributs spécifiques, tels que le nombre de camion de pompier ou le nombre de pompier, qui permettent de calculer la capacité de la caserne pour traiter les feu dans la ville.

La classe "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.

Les écoles possèdent un nombre d'enseignant ainsi qu'un nombre d'étudiant. Elle existent de plus en plusieurs niveaux, chaque niveau correspondant à un niveau d'éducation.

Ensuite, la classe "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.

La classe "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".

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 setHabitation-Niveau(int niveau)).

La classe "Hotel" permet d'accueillir des touristes. Elle définit donc un nombre d'employés ainsi qu'un nombre de touristes. Elle diffère des habitation classiques par le calcul de son attractivité et la nature de ses résidents. Les mécaniques de tourisme n'étant toute fois pas réalisées, elle reste au stade embryonnaire.

La mairie constitue le coeur de la ville. Toute ville doit en avoir une, la classe n'a cependant pas de fonctionnalité pratique dans le code. En effet c'est un bâtiment obligatoire mais qui n'apporte pas directement de fonctionnalité spécifique.

La classe "Police" définit les attributs des postes de police notamment, le nombre de policiers ainsi que le nombre de voitures de police disponibles et leur rayon d'action. La présence de postes de police est indispensable pour réguler les troubles à l'ordre public survenant dans la ville et leur absence entraîne le mécontentement des habitants.

La classe "PoleEmploi" donne accès au joueur à un bâtiment lui procurant des informations sur l'état du marché du travail de sa ville. Elle recense donc en particulier le nombre d'employé et le compare au nombre d'habitants.

2.2 Le package "menus"

Ce package contient tous les éléments nécessaires à la création des menus principaux d'un jeu qui sont : le menu à l'ouverture du jeu, le menu paramètre, le menu pause, le menu de reprise d'une partie et le menu de lancement d'une partie.

Tous les menus héritent d'une classe abstraite Menu car ils sont tous des sous-types de JPanel qui implémentent la même interface (ie "ComponentListener"). De plus tous les menus à l'affichage mettent à jour la taille de la fenêtre pour être sur que tous les éléments graphiques soient bien positionnés.

Dans chaque menu, des boutons 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.

2.3 Le package "carte"

Ce package contient toutes les classes nécessaires à la carte. Tel que Case qui est la composante principale de la Carte comme celle-ci se compose d'une liste de Case. Elle contient aussi les exceptions liés à Case : CaseOccupeeException qui indique que la case contient déjà une structure

et CaseVideException qui indique que la case est vide. La classe Case contient des méthodes utilisées dans Carte, pour ajouter une structure, récupérer une structure, savoir si la case est occupée et la vider.

Carte contient quant à elle toutes les méthodes nécessaires pour construire une structure, détruire une structure, sélectionner une case, déplacer une structure et récupérer une case de la carte.

2.4 Le package "routes"

Ce package contient toutes les routes qui sont implémentés dans cette version du jeu. Il contient une classe générale dans laquelle se trouvent toutes les méthodes des routes, puisque pour chaque route les méthodes sont identiques. Ainsi toutes les routes hériteront de cette classe générale.

Le package contient aussi une énumération Cardinal qui permet de connaître les croisements de la route et pour des versions ultérieures du jeu permettra de régler les statistiques suivant si les routes sont bien reliées entre elles ou si les bâtiments aussi.

La classe conversionCardinal contient des méthodes qui permettent de convertir les cardinal en numéro de route et rotation de celle-ci et inversement. Cela permet de faciliter la sélection et l'affichage des images.

2.5 Le package "affichagePartie"

Ce package, contient les classes nécessaires à l'affichage de la partie en cours. Tel que la bande d'action, les statistiques et la carte, ainsi que toutes les classes nécessaires à la sélection des actions et des structures.

Ainsi, les classes StatFinances et StatHabitants gèrent respectivement l'affichage des statistiques liées aux finances et aux habitants. Les classes Affichage_annonces et Affichage_titre permettent quant à elle respectivement l'affichage des annonces de la ville et l'affichage du nom de la partie et de la date de celle-ci. Ces dernières classes sont toutes instanciées dans la classe BandeStatistiques.

La classe BandeAction gère l'affichage des différentes structures plaçable, ainsi que la sélection de ces différentes structures, et la classe SelectionAction permet de communiquer le choix de l'utilisateur entre les outils graphiques et la partie.

Le package contient de plus la classe AffichageCarte qui permet l'affichage de la carte, c'est à dire de chaque case et de la structure qu'elle contient (route ou bâtiment). Elle gère aussi la création et le placement des structures, et le mouvement de la carte.

Enfin, la classe AffichagePartie permet de contenir les instances des classes majeures liées à l'affichage de la partie : affichageCarte, BandeAction et BandeStatistiques et de les repartir dans la fenêtre.

2.6 Le package "sauvegarde"

Ce package contient une classe Sauvegarde ainsi que quelques tests. Cette classe contient 4 méthodes qui permettent de créer une sauvegarde, reprendre une sauvegarde, renommer une

sauvegarde et supprimer une sauvegarde.

La méthode creersauvegarde(PartieCourante partieCourante) va dans un dossier sauvegarde situé ou créé à l'endroit de l'exécution, enregistrer un fichier .city contenant la partie à enregistrer et ajouter dans le fichier Villes_Sauvegardees.save le nom de la partie. Ce fichier .save est utile pour l'affichage du menu de reprise de la partie. Le joueur n'a plus qu'à sélectionner le nom de la ville pour la lancer.

La méthode reprendreSauvegarde(string nomPartie), va renvoyer la partieCourante qui a le nom donné en paramètre située dans le fichier .city correspondant.

La méthode renommerVilleSauvegarde va quant à elle renommer à la fois le .city, le nom de la partie dans le fichier Villes_Sauvegardees.save et le nom de la partie dans la classe partieCourante en sauvegarde.

La méthode détruire va supprimer le fichier .city correspondant au nom de la partie et supprimer le nom de la partie dans la liste de ville dans Villes_Sauvegardees.save.

Pour permettre de réaliser ces méthodes, il a fallut que toutes les classes qui étaient sauvegardées (soit car elles sont des attributs de classes sauvegardées, soit car elles sont directement sauvegardées) implémentent l'interface Serializable.

3 Diagramme de Classe

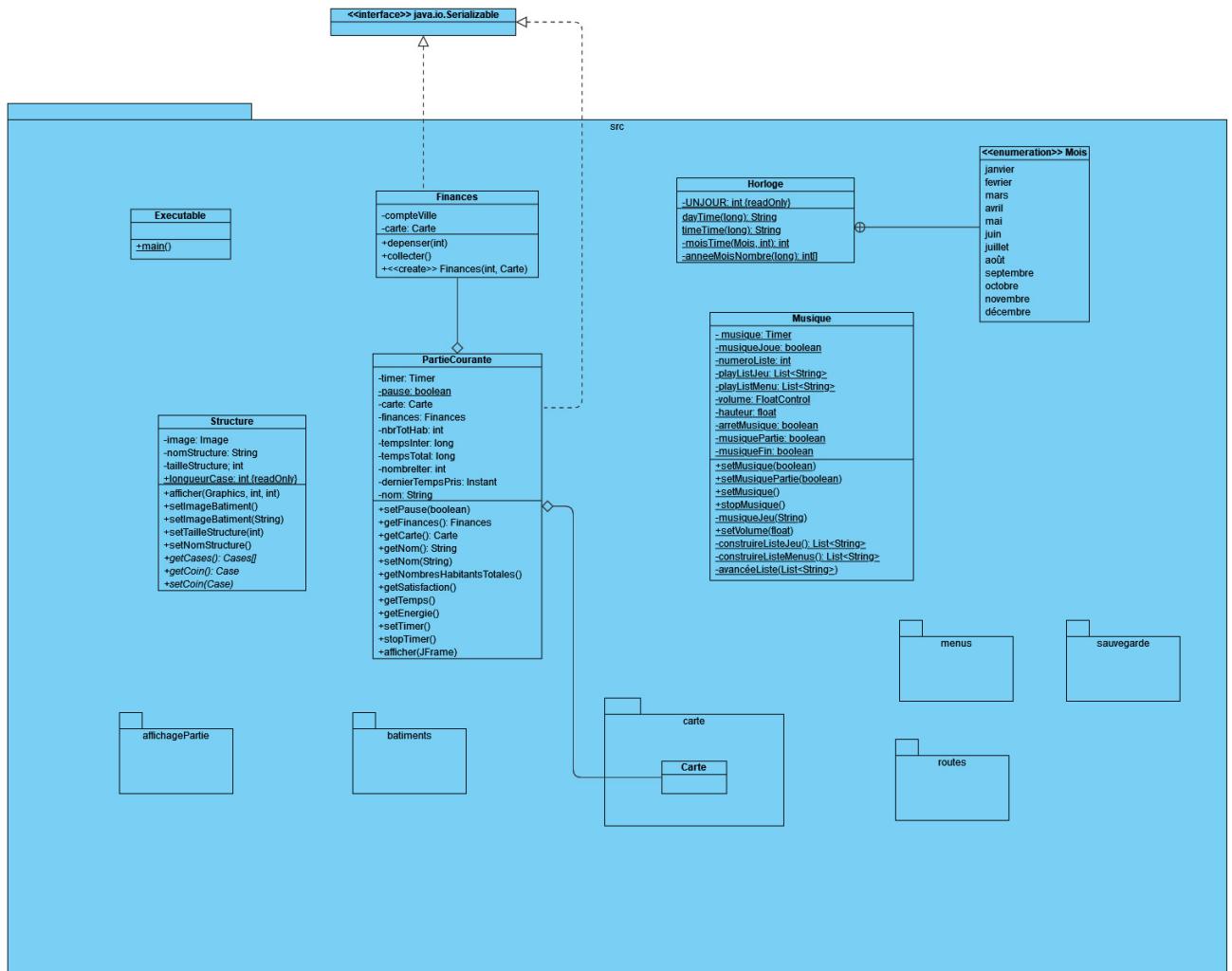


FIGURE 2 – Diagramme UML du City Builder

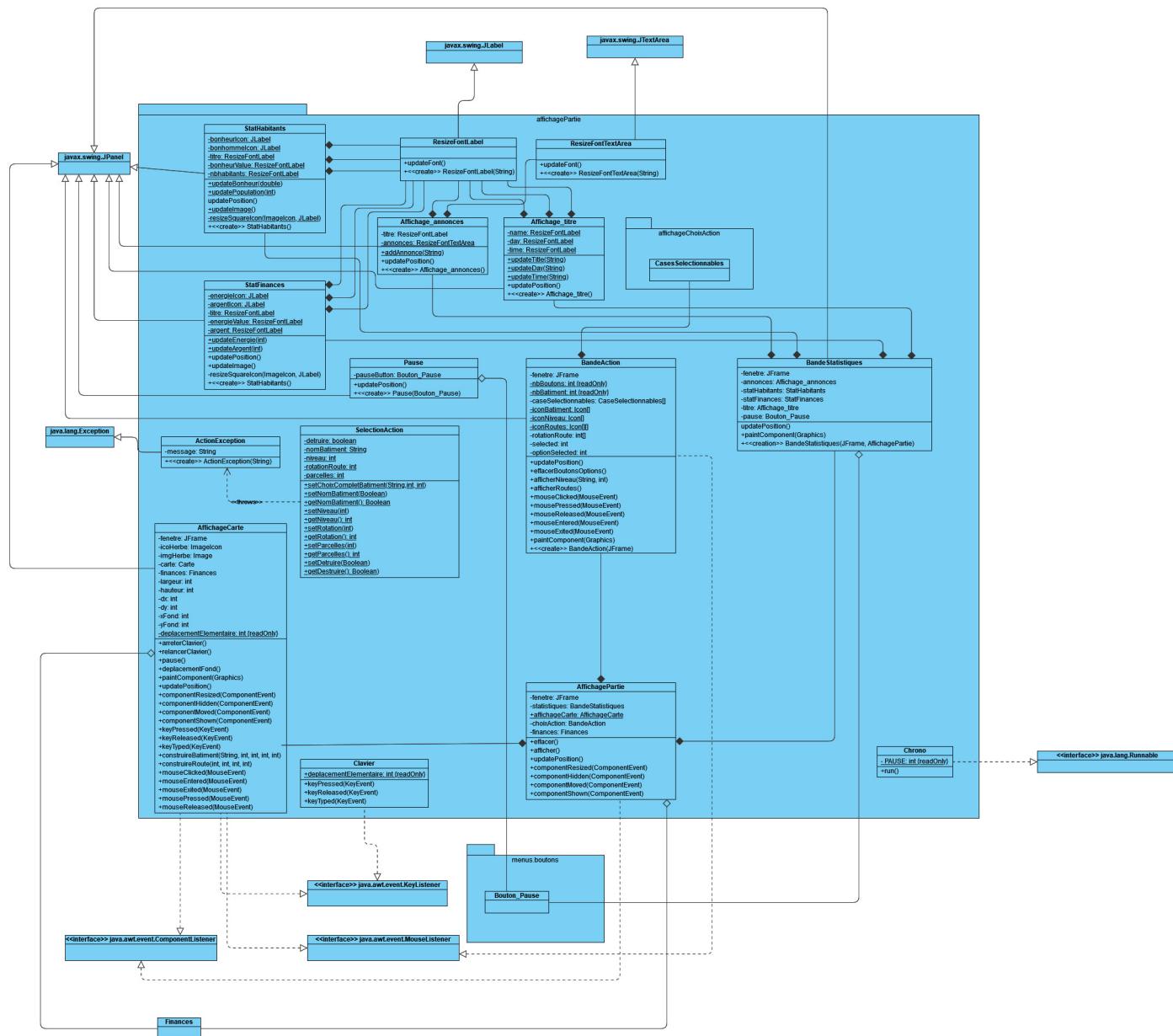


FIGURE 3 – Diagramme UML du package AffichagePartie

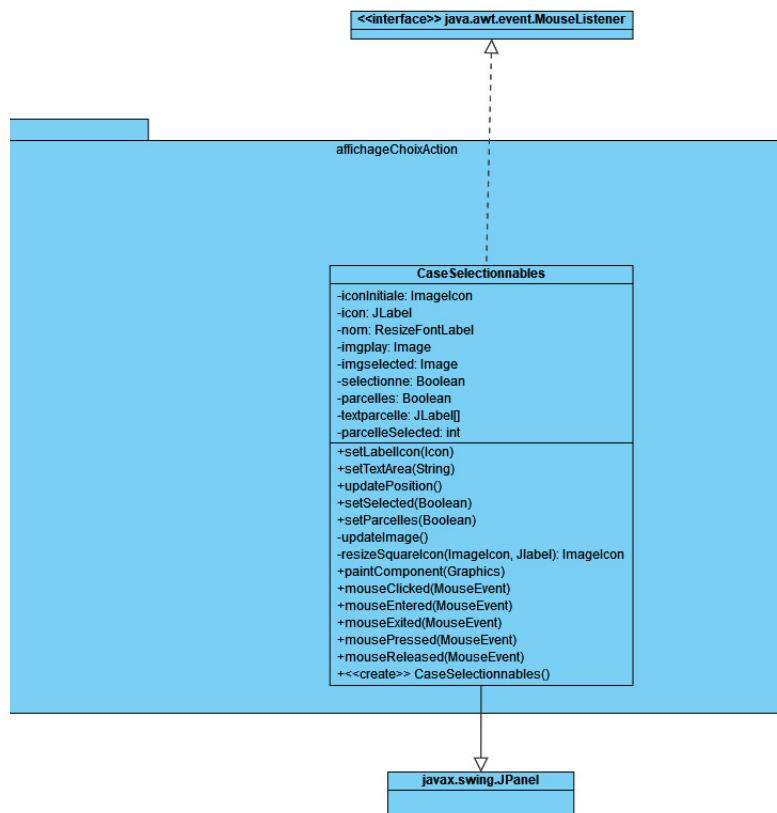
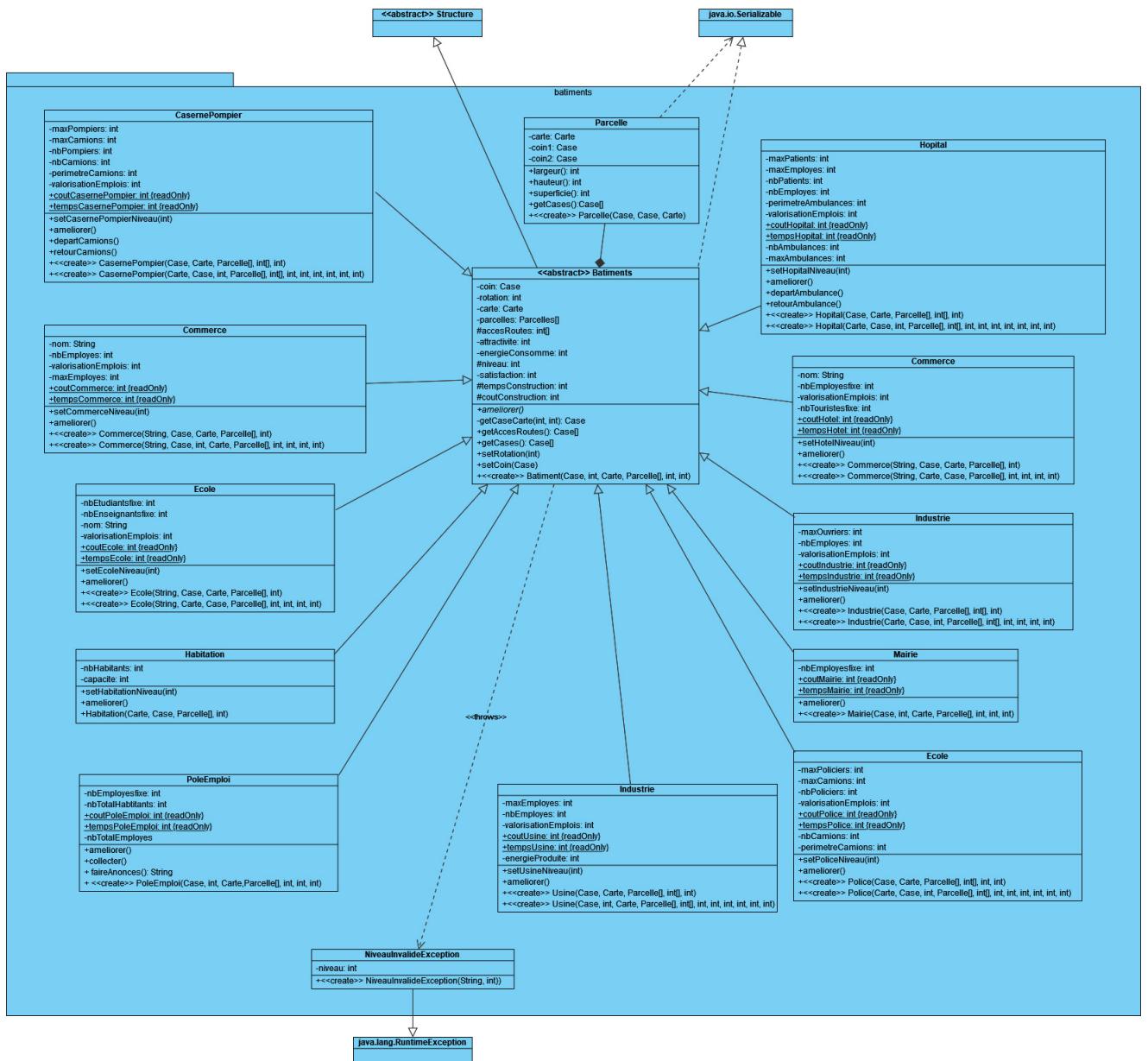


FIGURE 4 – Diagramme UML du package AffichageChoixAction



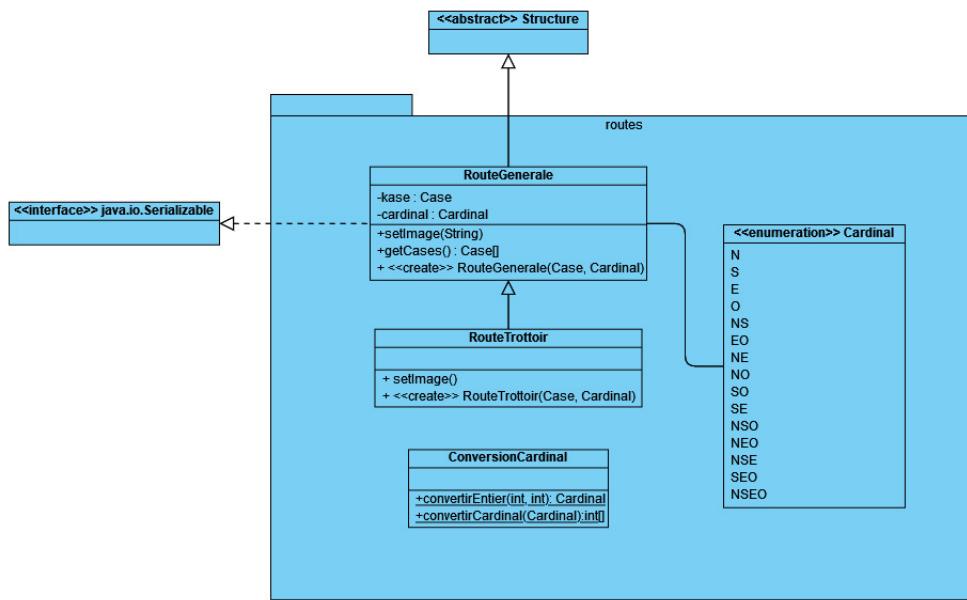


FIGURE 6 – Diagramme UML du package Routes

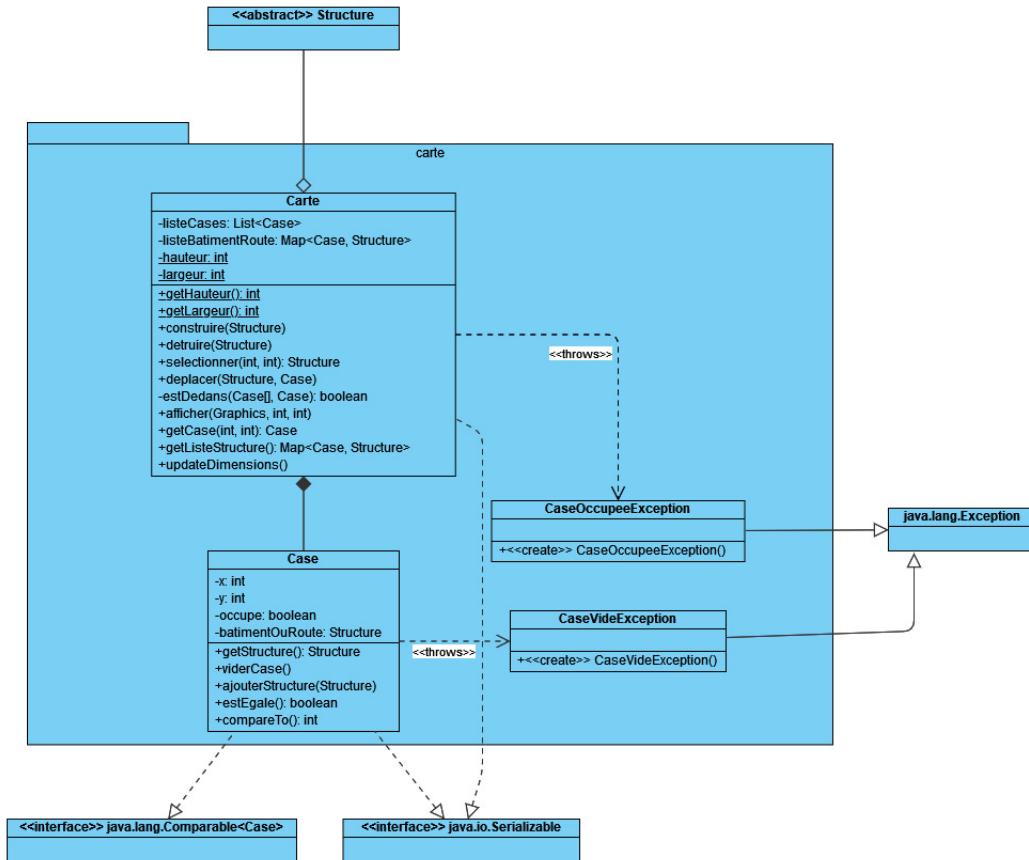


FIGURE 7 – Diagramme UML du package Carte

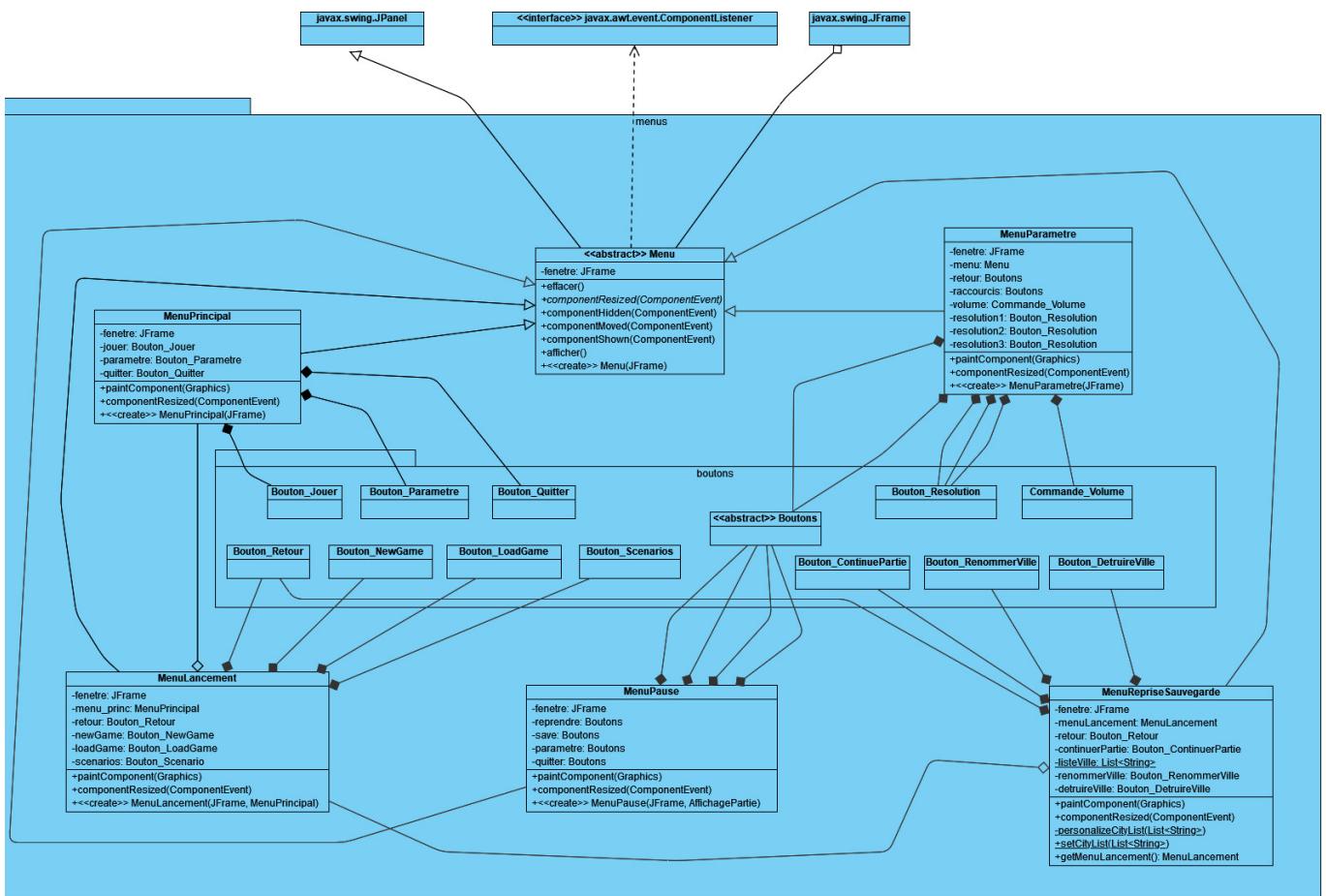


FIGURE 8 – Diagramme UML du package Menus

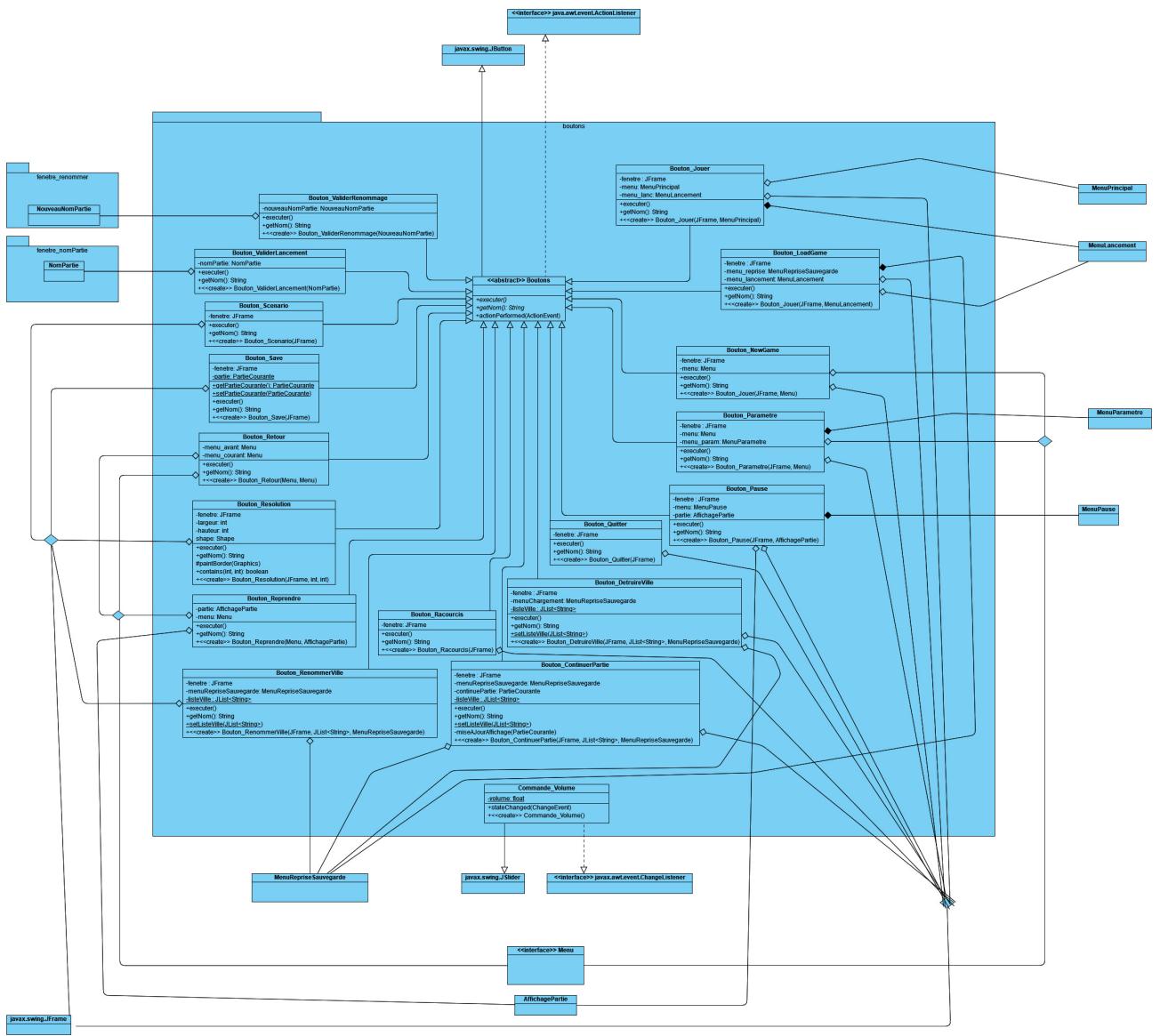


FIGURE 9 – Diagramme UML du package Boutons

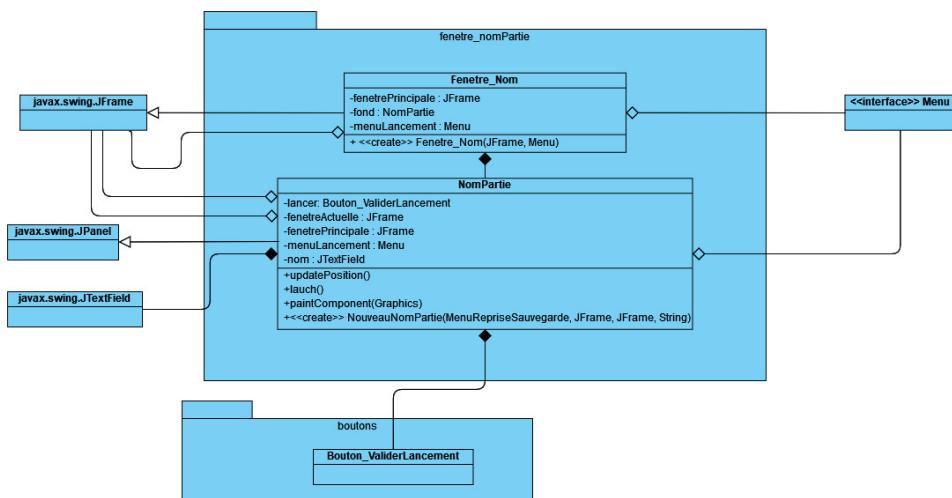


FIGURE 10 – Diagramme UML du package Fenetre_nomPartie

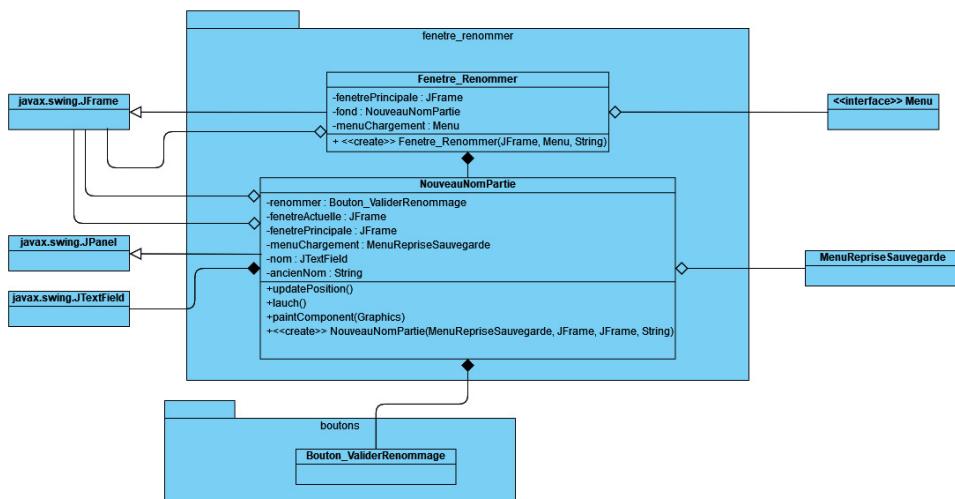


FIGURE 11 – Diagramme UML du package Fenetre_Renommer

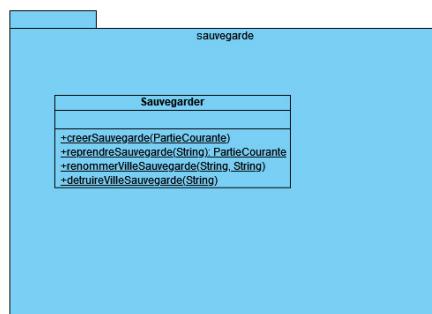


FIGURE 12 – Diagramme UML du package Sauvegarde

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

4.1 Cration des batiments

Nous avons opte pour une classe abstraite pour crer la classe principale "Batiment". La prise en compte de tous les paramtres et scenarios possibles tait un peu compliqu. Nous avons donc choisi de faire des sous-classes de "Batiment".

4.2 Interface utilisateur

L'interface utilisateur est gree dans des classes bien distinctes que le fonctionnement du jeu en lui mme de manire  ce qu'une classe d'affichage soit un composant graphique en lui mme. Ainsi certaines mthodes sont des mthodes de classes de manire  ce que dans le modification en backend puisse modifier l'affichage sans devoir avoir accs  l'lement prcis qu'il doit modifier. Plusieurs problmes se sont rencontr sur la superposition d'lments en Swing qui ont t rgler par le fonctionnement de : 1 vue = 1 JPanel et que chaque page du jeu soit un JPanel qui fait la taille de la fentre (en particulier pour tous les menus).

4.3 La carte

Nous avons fait le choix de centraliser les informations dans une classe Carte. C'est elle qui contiendra les diffrents batiments et routes construits lors d'un jeu. C'est par cette classe que l'on passera pour construire, dtruire, dplacer, afficher quelque chose.

Nous avons fait le choix de crer une classe Case. Et la carte est une liste de ces cases dans lesquelles se trouvent les batiments et routes. Nous avons aussi dcid de crer une liste des batiments et routes afin de simplifier le parcours de la carte lors de l'affichage de celle-ci.

4.4 Cration des routes

Nous avons fait le choix d'une classe route gnrale qui contient toutes les mthodes, et chaque route diffrente tendra cette classe pour pouvoir faciliter l'affichage.

4.5 Affichage de la carte

Le problme principal que nous avons rencontr quant  cet partie du projet est li au KeyboardListener utilis pour dplacer la carte. En effet, selon la manire dont tait lance la partie, celui-ci ne s'activait pas tout le temps. Ce problme a t rgler en ajoutant des options au JPanel (associ  l'affichage de la carte) qui taient actives quel que soit la manire dont on lancait la partie.

4.6 Sauvegarde des fichiers

Nous avons eut des problme lors de la sauvegarde et la reprise de la partie. Certains attributs de PartieCourante comme la carte se sauvegardaient mal. Pour rgler ce problme nous avons crer des fonctions qui mettent  jour ou recrent ces attributs lors de la sauvegarde.

4.7 Problèmes au niveau de la compilation

Tous les membres du groupes n'utilisaient pas le même IDE pour coder ainsi la compilation posaient problème, en particulier pour la gestion des images et des package. Nous avons donc décider d'écrire un fichier qui permets de compiler, nettoyer, exécuter et créer les sources (.jar) du projet de manière à ce que tous le monde compile le projet de la même manière sans obliger un IDE en particulier à tout le monde.

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 s'occuper de 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 régulièrement 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 était loin d'être créée à l'itération 1 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.

Afin de répartir les tâches entre les différents membres du groupe, nous avons donner des points d'efforts, qui correspondent à des difficultés estimées ainsi que des points d'importances à chacune des user-stories. Nous nous sommes aperçus avec l'aide de notre intervenant que nous nous étions trompés au niveau de la répartition des points d'importances. En effet nous étions parti d'un point de vu développeur, alors que c'était le point de vu utilisateur qui devait être utilisé dans cette tâche. Nous avons donc rectifié l'attribution des points en nous mettant à la place d'un utilisateur du jeu.