Rapport AP4B

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Diagrammes UML

Rapport – Diagrammes UML - SimPower

ALBUZLU Gökdeniz
LECAS Nicolas
VAN AKEN Brice
DENIS EVAN

Pôle Energie et Informatique

UV AP4B : Modélisation UML et mise en pratique avec le langage Java

Responsable : Franck GECHTER

Table des matières

Introduction

Lors du développement d'un programme, il est toujours nécessaire de passer par la phase UML, phase permettant de concevoir nos idées concernant la manière d'implémenter le code. C'est en quelque sorte le lien entre idée et code. Pour rappel, UML signifie en anglais « Unified Modeling Language », traduction de Langage de modélisation unifié. On s'intéresse donc dans ce rapport aux modélisation UML de notre projet.

Notre projet consiste en la réalisation d'un jeu tournant sous java ressemblant fortement au jeu SimCity. En effet, le joueur doit gérer une ville en construisant des bâtiments de plusieurs types tout en satisfaisant les habitants.

La phase UML est déterminante quant à l'aspect du code que nous devrons fournir pour achever un tel programme. C'est en quelque sorte notre ligne directrice.

Dans ce rapport, nous allons tout d'abord étudier les classes plateau et case qui constituent le noyau du jeu, puis les constructions et enfin nous parlerons des diagrammes de cas d'utilisation et de séquence.

Préambule

Afin de comprendre le fonctionnement de notre programme, nous allons décrire brièvement son fonctionnement.
Nous avons choisi tout d'abord de créer une classe « ville » dans laquelle on stockera toutes les ressources (satisfaction, argent, fer …) sous forme d'attributs statiques, modifiables par toutes les classes. Ensuite, le nom de la classe « Plateau » définit parfaitement son rôle : il contient un tableau deux dimensions d'éléments de type case. De plus on y retrouve toutes les vérifications nécessaires à la construction d'un élément sur une case afin de respecter les règles du jeu.

De même pour la classe « Case », c'est elle qui va supporter réellement les futures constructions. Les cases sont comme indiqué ci-dessus présentes dans notre tableau

Enfin, la classe « Construction » va caractériser chaque type de construction, à savoir leur coût en ressource par exemple qui dépendra du type de construction, c'est-à-dire de ses classes filles.

I/ Diagramme de classe : le noyau du jeu

1. Ville

La classe ville a été créée afin de contenir l'ensemble des ressources de l'utilisateur nécessaire au bon fonctionnement du jeu.

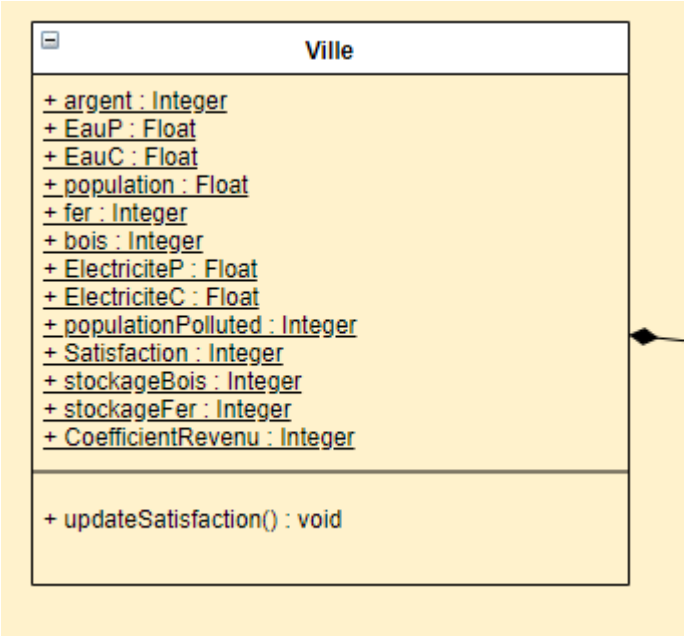

Diagramme de la classe « ville »

Elle permet aussi une mise à jour de la satisfaction, selon 3 paramètres :

-L'eau

-L'électricité

-La pollution

Chacun compte comme un tiers de la satisfaction globale et sont calculé selon le ratio entre consommation et production pour l'eau et l'électricité.

Pour la pollution, elle est calculée selon le ratio entre les personnes habitant sur une case polluée et la population globale.

Les attributs sont Static afin de pouvoir les utiliser sans avoir besoin d'instancier l'objet Ville. Ainsi nous pouvons avoir accès et modifier ces attributs depuis les autres classes.

1. Plateau

La classe Plateau est la classe la plus importante et imposante de l'ensemble du projet. En effet elle regroupe les méthodes pour construire les bâtiments, les détruire, faire des vérifications…
Elle contient tableau, qui est le tableau à deux dimensions des cases du plateau de jeux.

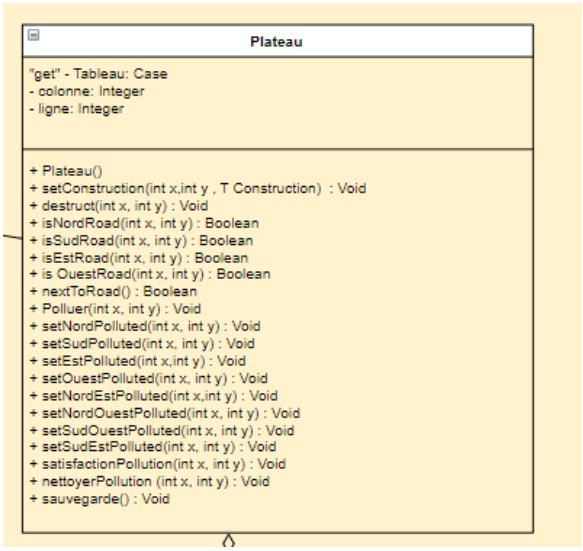

Diagramme de la classe « plateau »

La méthode Construire prend en paramètre les coordonnées ainsi qu'un type générique (suivant le bâtiment).

La méthode destroy va réinitialiser la construction sur la case afin de la remettre dans son état d'origine. Les quatre is (Nord, Est, Ouest, Sud) Road, permettent de retourner true s'il y a une route dans la direction indiquée sinon elles renvoient false. Les résultats sont rassemblés dans nextToRoad afin de savoir si la case est à côté d'une route.
Polluer quant à elle permet de polluer la case ainsi que les cases toute autour, comme par exemple sur la création d'une centrale nucléaire.
Nous trouvons toutes ensuite les méthodes pour toutes les directions pour vérifier si les cases autour sont pollués.
satisfactionPollution, permet de mettre à jour le nombre de personnes vivant sur une case polluée.
Pour terminer, nettoyerPollution permet à l'utilisateur de dépolluer les cases contaminées.

Il y a une composition entre Ville et Plateau car le plateau est un élément indissociable de la ville, car nous utilisons dans plateau une méthode de Ville.

2. Case

Chaque case est un élément du tableau contenu dans plateau, et chacune d'entre elles va contenir certains attributs la définissant. Nous pouvons donc dire que le plateau est composé de 1..* cases.

Nous utilisons donc une relation d'agrégation entre les deux.

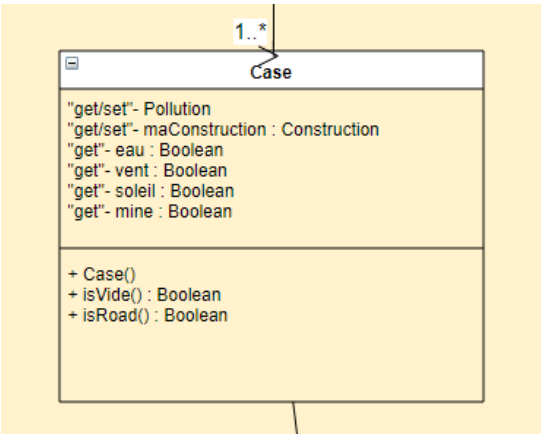

Diagramme de la classe « Case »

Case quant à elle est composé des attributs booléen eau, vent, soleil, mine, pollution qui sont les ressources qui devront être utilisées par l'utilisateur du jeu. Les attributs eau, vent, soleil et mine sont générés aléatoirement à l'initialisation du jeu lors de la création.

Nous avons aussi les méthodes get pour les obtenir et set pour celles qui pourront potentiellement être changées durant l'exécution du jeu.

L'attribut le plus important de case est maConstruction. De type construction, cet attribut représente la future construction présente sur la case. L'association de l'attribut à la construction que l'utilisateur demande à construire se faisant dans setMaconstruction. En effet setMaConstruction prend en paramètre un type générique et suivant le type de construction (habitation / producteur d'eau ou d'électricité) il va mettre à jour les informations de production, consommation et de population.

isRoad et isVide vont retourner vrai ou faux suivant si la case est vide ou si elle contient une route. Pollute et Unpolluted sont quant à eux les get et set pour la pollution.

II/ Diagramme de classe : Les constructions

1. Construction : Une classe mère nécessaire.

« Construction » est une classe qui va définir le prix des constructions et leur nom. Il s'agit d'une classe abstraite, c'est-à-dire qu'elle n'est pas instanciable. On se servira juste de ses méthodes et attributs dans les nombreuses classes filles qui hériteront de celle-ci. Elles correspondront aux types des bâtiments. On pourra donc réutiliser les attributs et méthodes de cette classe.
L'attribut « coutArgent » est un entier va définir le coût financier des bâtiments. De même pour « coutBois » et « coutFer » qui définiront respectivement le coût en bois et en fer d'un bâtiment.
« RessourcesEngoughtToBuild() » est une méthode de type void qui va vérifier si les conditions argent, fer, et bois sont satisfaites. En d'autres termes, on vérifie si le joueur possède suffisamment de ressources. (attributs statiques de « ville »)

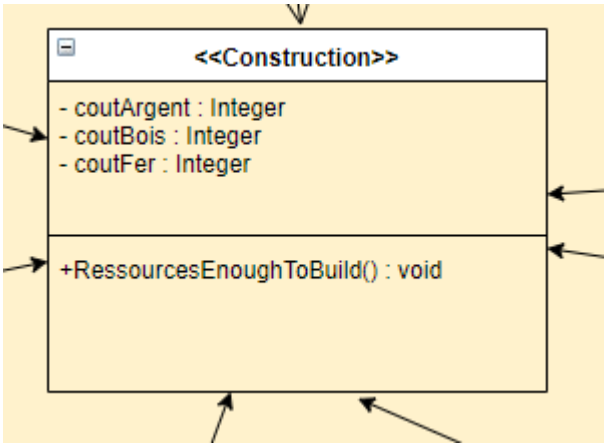

Diagramme de la classe « Construction »

2. Habitation : une classe fille de « Construction »

La classe « Habitation » est une classes abstraite fille de « Construction ». On retrouve dans celle-ci des getters et setters pour l'attribut population.
On retrouve en attribut des entiers : « consoEau » et « consoElec ». Ils vont définir la consommation des habitations (classes filles) héritant de « Habitation ».
« MajConsommation() » et « MajPopulation() » sont deux méthodes de type void. Elles vont mettre à jour les ressources globales du jeu, c'est-à-dire les attributs statiques de la classe « ville ».

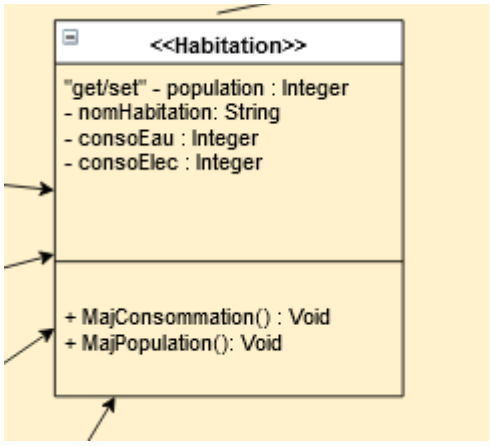

Diagramme de la classe « Habitation »

3. Les autres jumeaux de la classe « Habitation »

On retrouve ici les classes abstraites « Energie », « Usine », « Stockage » et « Service ». Ces classes fonctionnent d'une manière très similaire à la classe « Habitation » vu dans le point précèdent. Il parait redondant d'en expliquer le fonctionnement.

Quelques particularités cependant de la classe « Energie ». Elle ne possède que des getters et des setters. En effet, cette classe va récupérer la production et la pollution des différents types de bâtiments héritant de la classe « Energie ».

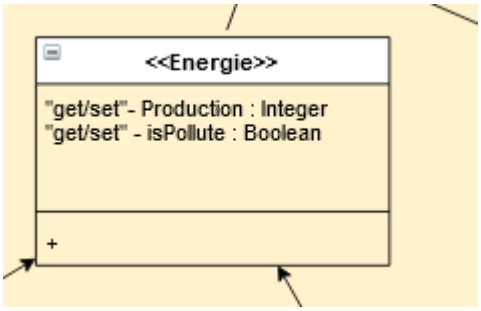

Diagramme de la classe « Energie »

La classe « Usine » quant à elle possède une méthode MajProductionUsine(int id) qui va mettre à jour (augmenter) les ressources dans les attributs de ville (exemple : le fer). Cette méthode est invoquée par la méthode Chrono(int id) qui va l'appeler à des intervalles réguliers. Le paramètre id est ici utilisé pour différencier les différents types d'usine (fer et bois)

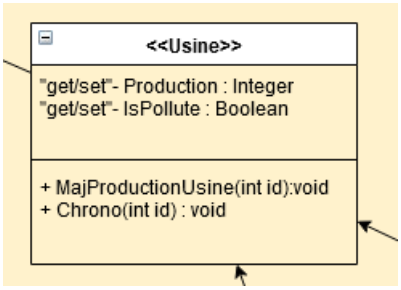

Diagramme de la classe « Usine »

La classe Stockage est la mère des deux classes StockageBois et StockageFer qui régissent la quantité de bois et de Fer dont l'utilisateur pourra disposer au maximum. Ces deux variables seront améliorables.

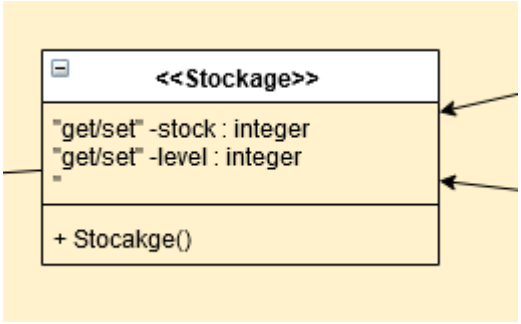

Diagramme de la classe « Stockage »

La classe service possède une méthode spéciale MajCoeffRevenu() qui va augmenter l'argent au niveau de (attribut de la ville). En effet le nombre d'habitants n'est pas l'unique source de revenu de la ville, le nombre de bâtiments de service influe également sur la richesse. Cette fonction fonctionne augmente l'argent de la Ville, ceci grâce à l'attribut protected « coefficientRevenu », qui sera initialisé par chaque classe fille, donc bâtiment de service.

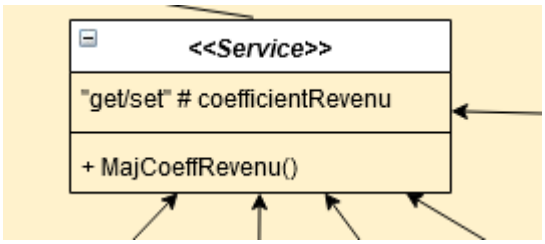

Diagramme de la classe « Service »

III/ Actions utilisateur : Cas d'utilisation et diagrammes de séquence

1. Diagramme de cas d'utilisation

Lors de l'arrivée sur le monde le joueur pourra créer une nouvelle partie, puis si le joueur a déjà joué auparavant, il pourra charger une ancienne sauvegarde pour continuer à jouer.

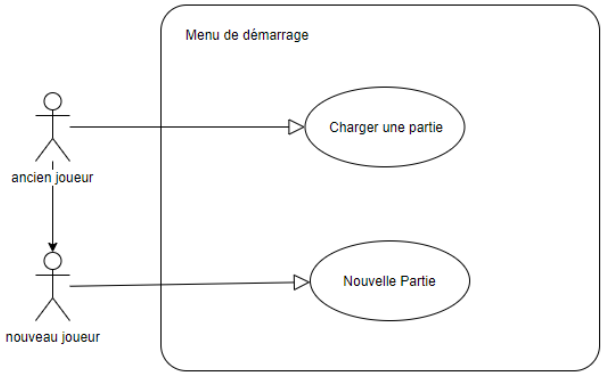

Diagramme de cas d'utilisation du menu de démarrage

Une fois arrivé sur le monde, le joueur aura devant les yeux différentes possibilités.

Dans le menu de droite, il aura accès au menu de construction, qui, est le plus imposant en matière de fonctionnalités et de variétés d'actions.

En effet, il aura accès à quatre menus correspondants chacun à un type du bâtiment (Energie, Habitation, Usines, Services) ainsi que la possibilité de construire une route ou de détruire des bâtiments.

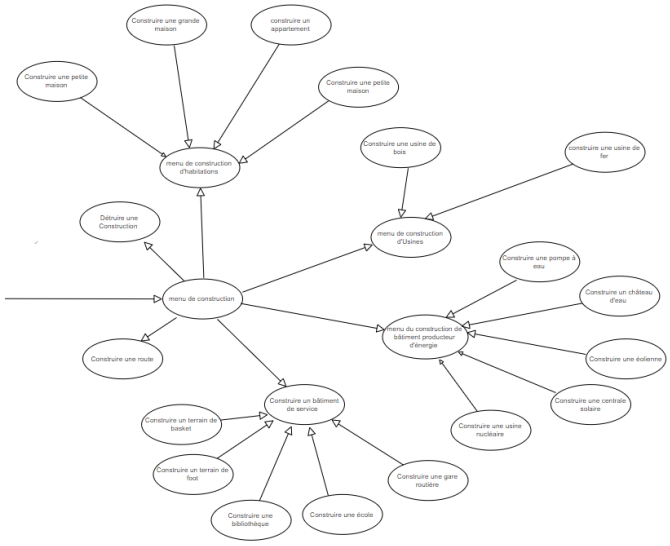

Diagramme de cas d'utilisation du menu de construction

Dans le menu du bas, il peut avoir accès aux différentes cartes qui lui permettent de voir où sont placés les cases contenant les ressources et la pollution.

Il peut aussi choisir de mettre les usines en fonctionnement.

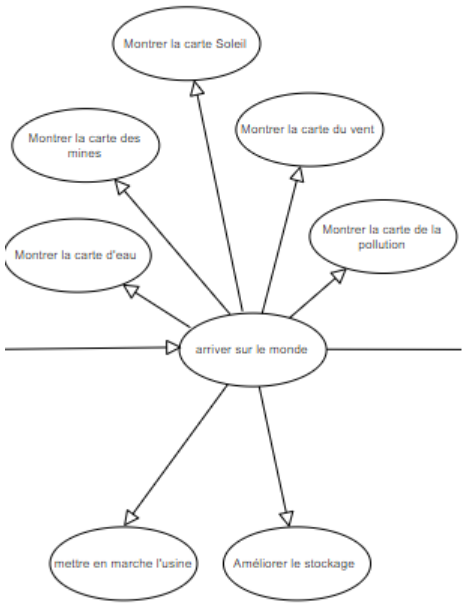

Diagramme de cas d'utilisation du menu des cartes

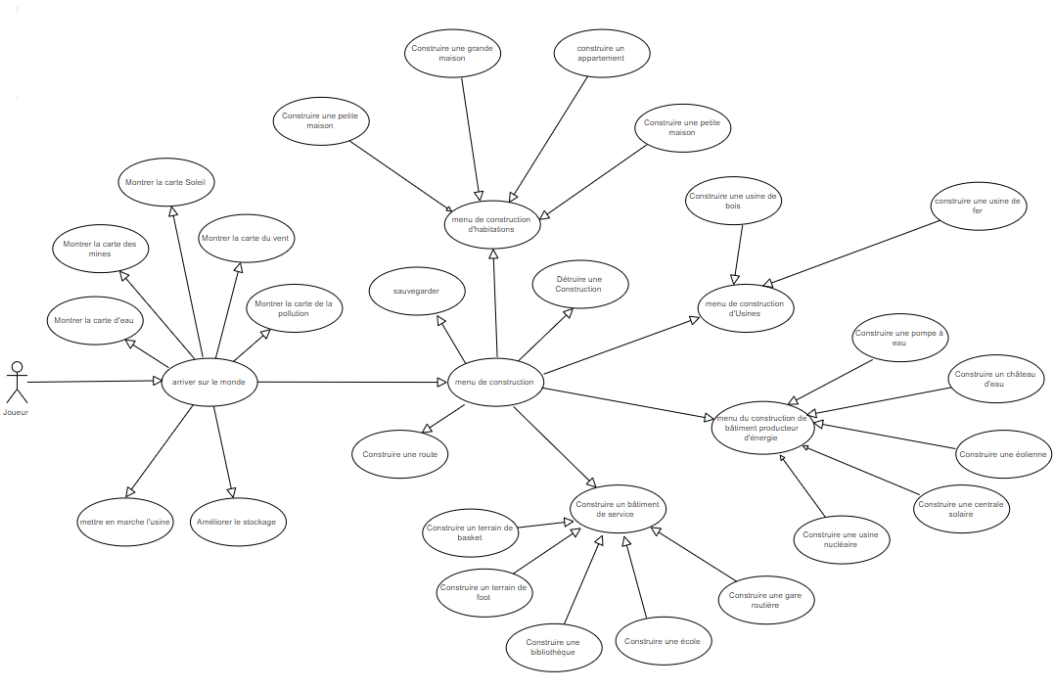

Diagramme de cas d'utilisation complet du jeu

2. Diagrammes de séquences : Actions principales

Les deux actions principales de l'utilisateur dans le jeu sont la construction et destruction de bâtiments. Ce sont donc nos diagrammes de séquences les plus importants.

2.1 Construction

Lorsque l'utilisateur pose un bâtiment, on doit vérifier tout d'abord si une case est vide. On appelle pour cela une méthode retournant un booléen pour vérifier cette dernière. On récupère d'abord sur quelle case l'utilisateur a cliqué. Si la case était pleine, alors on retourne un message d'erreur, sinon on vérifie que le bâtiment est à proximité d'une route, puis on vérifie si les ressources sont satisfaisantes avant de mettre à jour les ressources, les habitants, la satisfaction…
Enfin, si les ressources sont insuffisantes, on renvoie un message d'erreur.

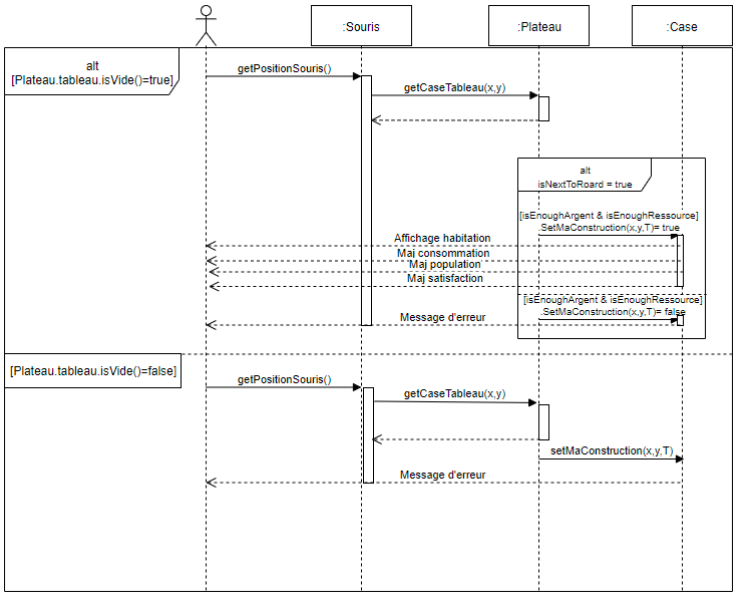

Diagramme de construction d'un bâtiment de type habitation

Une variante de ce diagramme a été créé pour les bâtiments d'énergie, on met à jour en plus la production d'électricité dans ces derniers. D'autres diagrammes pourront voir le jour si le programme évolue lors de sa version finale. (Nouveaux bâtiments etc)

2.2 Destruction

La destruction des bâtiments ressemble à la construction. Ici, on inverse la condition, et on vérifie si les cases du tableau sont pleines. On récupère d'abord la position de la souris, puis la case du tableau associé. Si la case est vide, alors on invoque une méthode de destruction qui entraine la suppression de la valeur contenue dans la case du tableau, entrainant la suppression de l'affichage du bâtiment, une mise à jour des ressources, de la population et de la satisfaction. Si la case était vide, alors on retourne un message d'erreur.

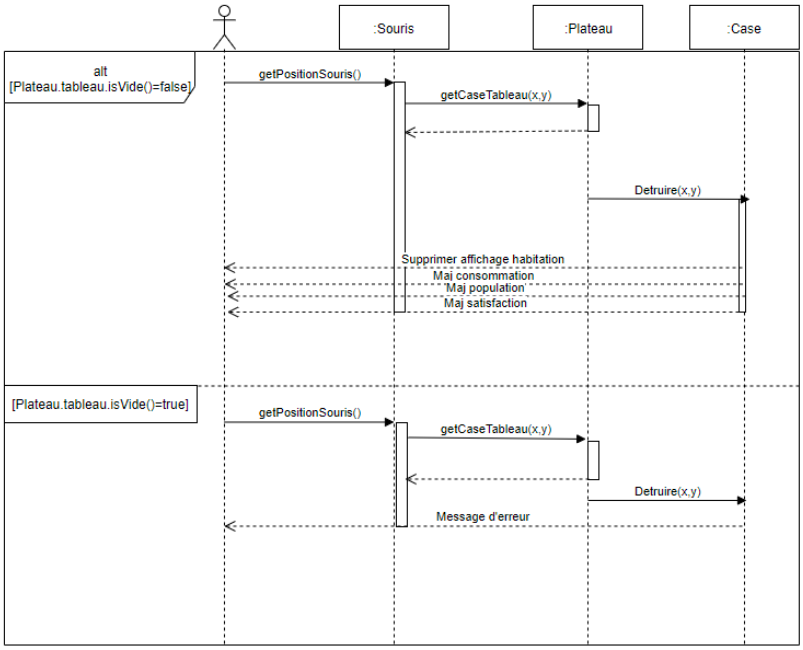

Diagramme de destruction d'un bâtiment de type habitation

Encore une fois, une variante de ce diagramme a été créé pour les bâtiments de type énergie. On mettra à jour dans ceux-ci la production d'électricité.

3. Diagrammes de séquences : Actions secondaires

On va retrouver dans cette partie toutes les actions liées au boutons disponibles dans le jeu. Ces boutons servent notamment à la construction et la destruction des bâtiments.

3.1 Bouton Construction

Ici on va lancer la construction d'un bâtiment (comme vu précédemment) lorsque l'utilisateur cliquera sur le bouton.
On vérifie que l'utilisateur clique sur le bouton, puis on invoque la méthode de construction d'un bâtiment, dès lors il faudra suivre l'autre diagramme UML vu dans la partie 2

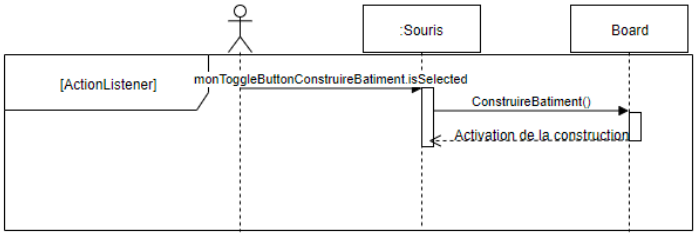

Diagramme du bouton de construction

3.2 Bouton Destruction

De même que pour la construction, on lance la destruction lorsque qu'on clique sur le bouton.
On vérifie que l'utilisateur clique sur le bouton, puis on exécute la méthode de destruction d'un bâtiment

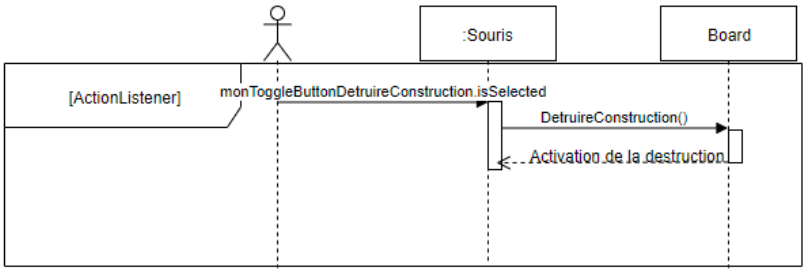

Diagramme du bouton de destruction

3.3 Bouton Sauvegarde

On exécute la sauvegarde lorsqu'on clique sur le bouton. On vérifie que l'utilisateur clique sur le bouton, puis on exécute la méthode de sauvegarde situé dans Plateau qui écrit dans un fichier .txt localisé dans src/simcity.

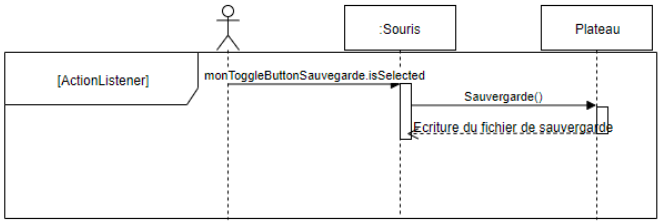

Diagramme du bouton Sauvegarder

3.4 Bouton de chargement

Ici on lance le chargement lorsqu'on clique sur le bouton. On vérifie que l'utilisateur clique sur le bouton, puis on exécute le constructeur de Board, qui appellera le constructeur lui aussi le constructeur de Plateau. Il en résulte de la lecture du fichier de sauvegarde .txt.

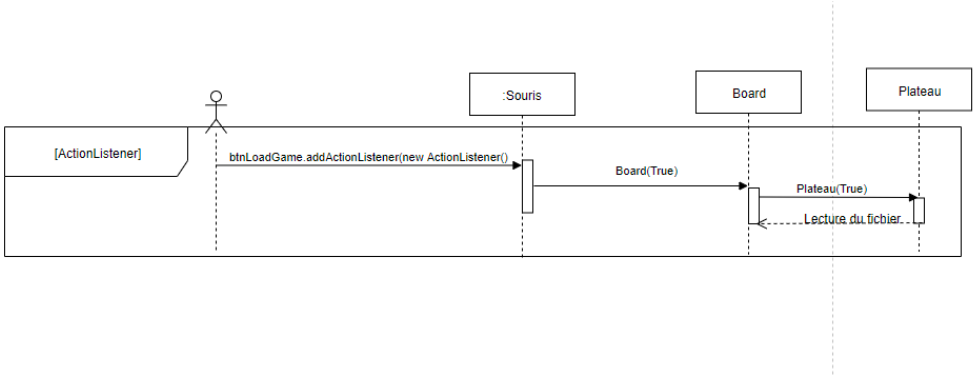

Diagramme du bouton LoadGame

3.5 Bouton d'amélioration du stockage (bois et fer)

On exécute l'amélioration du stockage lorsque l'utilisateur clique sur le bouton associé. On vérifie qu'il clique dessus, puis on exécute la méthode AmeliorerStockage situé dans Board, qui va appeler par la suite la méthode amelioration contenu dans Stockage. Cette même méthode appellera upgrade de l'objet (stockage bois / fer). Elle entrainera finalement l'execution de MajRessources() qui mettra à jour les ressources de stockage disponible dans la ville (static int).

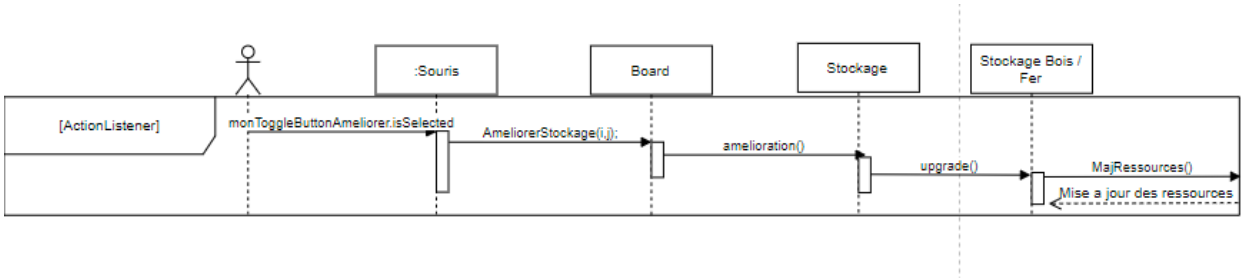

Diagramme du bouton Améliorer

3.6 Bouton d'affichage des règles

On vérifie que l'utilisateur clique sur le bouton, puis on exécute le fichier html des règles dans le navigateur par défaut de l'ordinateur.

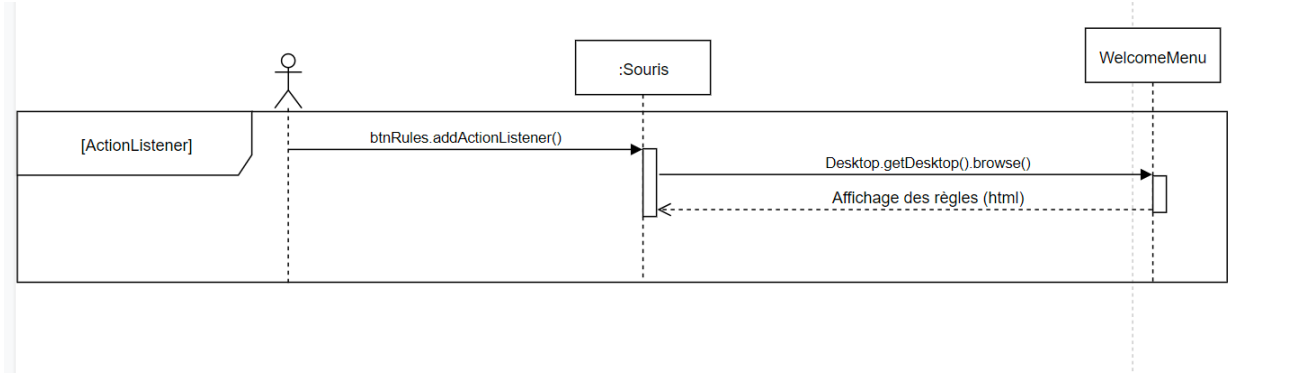

Diagramme du bouton Rules

Conclusion

Ainsi, nous avons étudié les points forts des diagrammes UML de notre projet. Ceux-ci vont définir notre code et diriger nos travaux dans la bonne voie.
C'est la réunion des 3 types de diagrammes qui permet une vision globale du code. Le diagramme de séquence permet de visualiser graphiquement les conséquences de l'action de l'utilisateur sur le programme. Les diagrammes de classe montrent directement les attributs, les méthodes et le type d'héritage de chaque classe.
Enfin, le diagramme de cas d'utilisation permet d'appréhender globalement le déroulement du SimPower à travers toutes les actions possibles de l'utilisateur.
Néanmoins il faut garder à l'esprit que ces diagrammes peuvent évoluer lors de la phase de développement du code. Il est possible que nous fassions face à des problèmes nécessitant une modification des différents diagrammes.

Liste des diagrammes :

Diagramme de Classe :

https://gitmind.com/app/flowchart/af17617300

Diagramme de cas d'utilisation du menu :

https://gitmind.com/app/flowchart/54f7096717

Diagramme de cas d'utilisation du jeu :

https://gitmind.com/app/flowchart/cb97097252

Diagramme de séquence de l'affichage de l'aide des boutons :
https://gitmind.com/app/flowchart/36f7064498

Diagramme de séquence du bouton destruction :

https://gitmind.com/app/flowchart/1f37064896

Diagramme de séquence du bouton construction :

https://gitmind.com/app/flowchart/6956523003

Diagramme de séquence du bouton LoadGame

https://gitmind.com/app/flowchart/b7f7617943

Diagramme de séquence du bouton Sauvegarde

https://gitmind.com/app/flowchart/95b7617823

Diagramme de séquence du bouton Améliorer

https://gitmind.com/app/flowchart/3497618302

Diagramme de séquence du bouton Rules

https://gitmind.com/app/flowchart/8567619007

Mots clefs

Rapport UML, Java, AP4B, Diagrammes,

ALBUZLU Gökdeniz **VAN AKEN Brice**

LECAS Nicolas **DENIS Evan**

Rapport AP4B – Diagrammes UML - SimPower

Résumé

Les diagrammes UML sont toujours nécessaires lors de la réalisation d'un programme, surtout lorsque qu'il nécessite la mise en place de nombreuses classes. Dans ce rapport, nous rendons compte des choix fais pour chaque type de diagramme. Ceux-ci étant cruciaux lors de la phase de développement, constituant une réelle ligne directrice pour notre projet.