Rapport Lumber

Le Guilly Erwann, Viala Alexandre, Lignon Thomas, Mann William

Sommaire

Introduction

Au cours de notre UV AP4B, nous avons été amené à créer un Idle game de gestion d'une entreprise de travail du bois. Nous devons réaliser ce jeu en langage Java à l'aide des bibliothèques graphiques de notre choix. L'aspect graphique n'est qu'une préoccupation secondaire dans le contexte de ce projet. En effet, l'objectif principal du projet est de suivre une démarche de génie logiciel en réalisant des schémas UML afin de nous aider à concevoir l'application principale. Dans ce rapport, nous avons réalisé tous les graphiques nécessaires à la conception d'un diagramme de classes.

Cahier des charges:

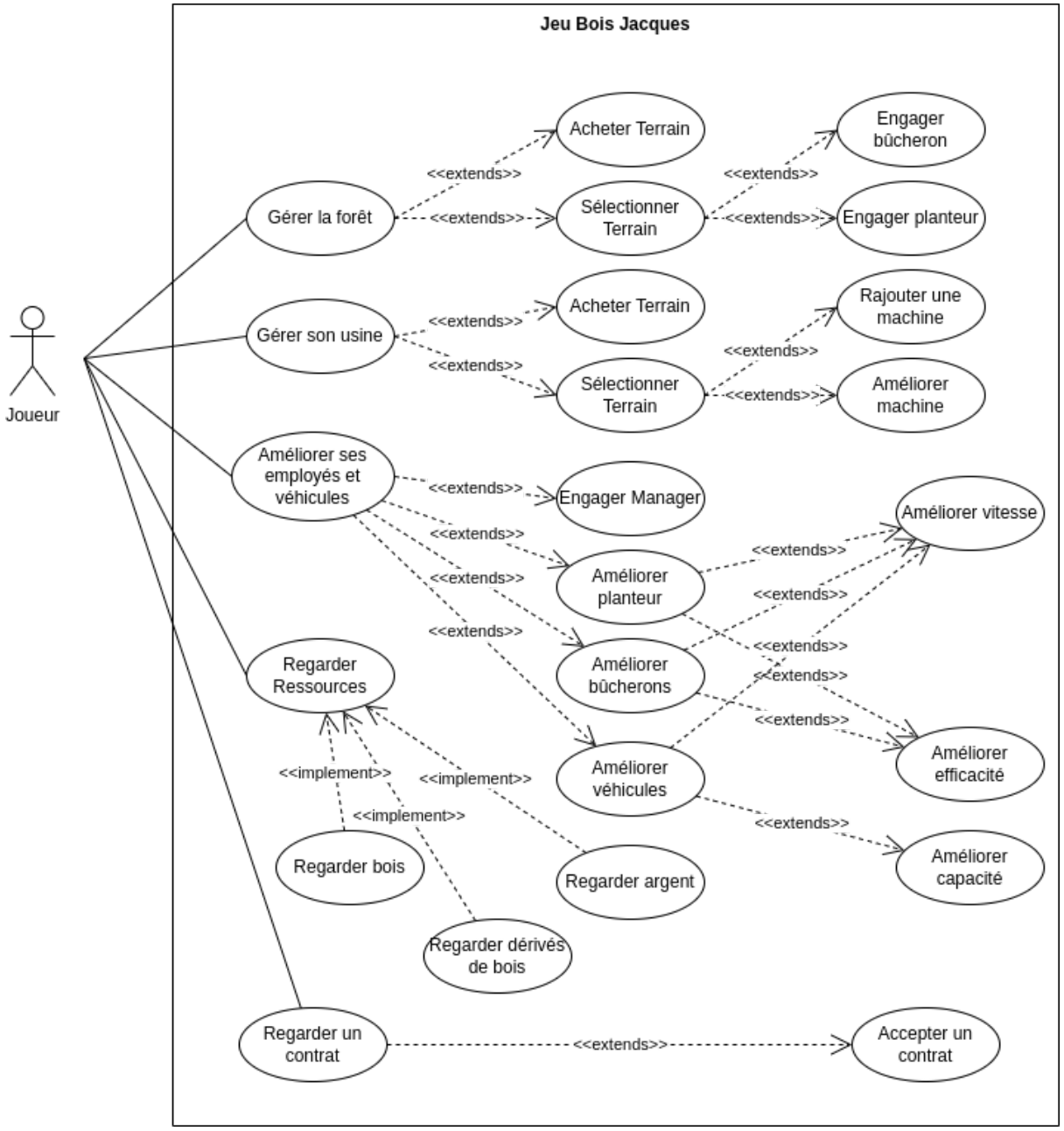
Objectif final:
- Modélisation UML du projet
- Implémentation d'une version fonctionnelle du jeu en JAVA

Fonctionnalités:
- Gérer les forêts, c'est-à-dire couper les arbres et augmenter son terrain, mais aussi replanter les arbres.
- Agrandir son usine pour pouvoir produire plus vite et diversifier les produits créés.
- Gérer ses employés et l'équipement pour qu'ils soient le plus efficaces possible.
- Engager un responsable marketing pour attirer le plus de clients possible.

Dans notre cas, le jeu se divise en 2 écrans, l'écran Usine et l'écran Forêt, ces derniers nous permettent de gérer respectivement le travail et la découpe du bois. Sur ces 2 écrans un menu d'amélioration des employés et des véhicules sera toujours disponible. Le joueur aura également à sa gauche un menu qui lui montre les ressources qu'il possède actuellement comme par exemple la quantité de bois, de planches ainsi que l'argent en sa possession. Il y a également un menu qui montre les commandes disponibles et un autre qui nous permet de gérer notre manager.

Sur l'écran Usine on peut voir les terrains (plots) que l'on peut acheter, chaque terrain est composé de 4 emplacements pour construire ses machines. On peut également acheter plus de véhicules pour l'usine. L'écran Forêt est semblable à l'écran Usine, on a la possibilité d'acheter un terrain pour produire plus de bois, lorsqu'un terrain est acheté, un bûcheron et un planteur sont automatiquement ajoutés sur ce terrain. On a également la possibilité d'embaucher des planteurs et/ou bûcherons pour entretenir les terrains.

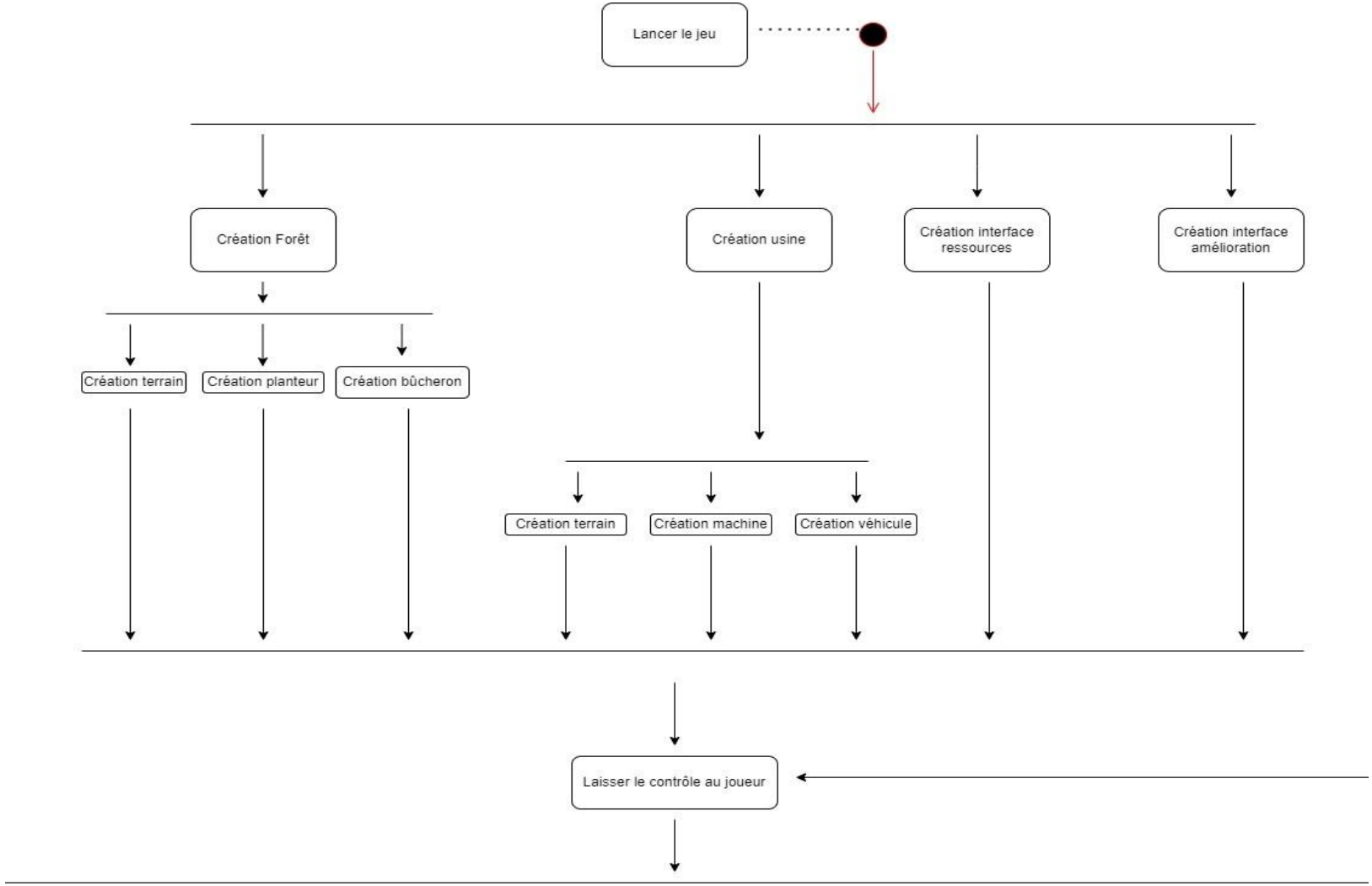
Diagramme de Cas d'Utilisation

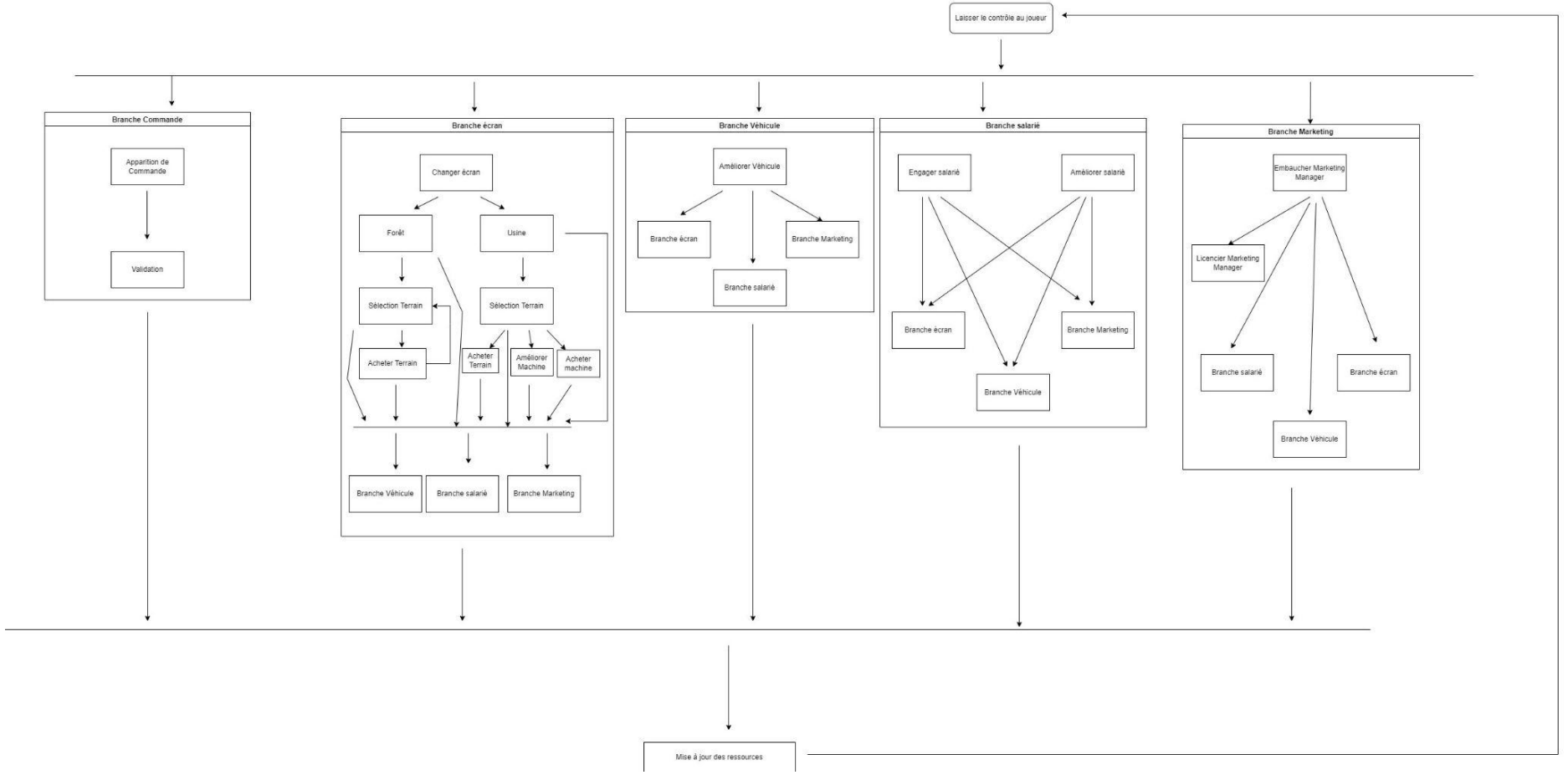


Le diagramme de cas d'utilisation ou diagramme "use case" permet de lister les différentes actions que l'utilisateur pourra réaliser. Il ne décrit cependant pas le retour de la machine, il nous permet simplement de connaître les différentes interactions qu'il faudra réaliser.

Dans notre cas, on a classifié les tâches des utilisateurs dans 6 catégories différentes. On remarque également qu'on pourra réutiliser du code pour plusieurs fonctionnalités, notamment l'amélioration de la vitesse ou de l'efficacité. On voit déjà apparaître certaines "vues" comme celle qui nous permettra de regarder les ressources de l'utilisateur.

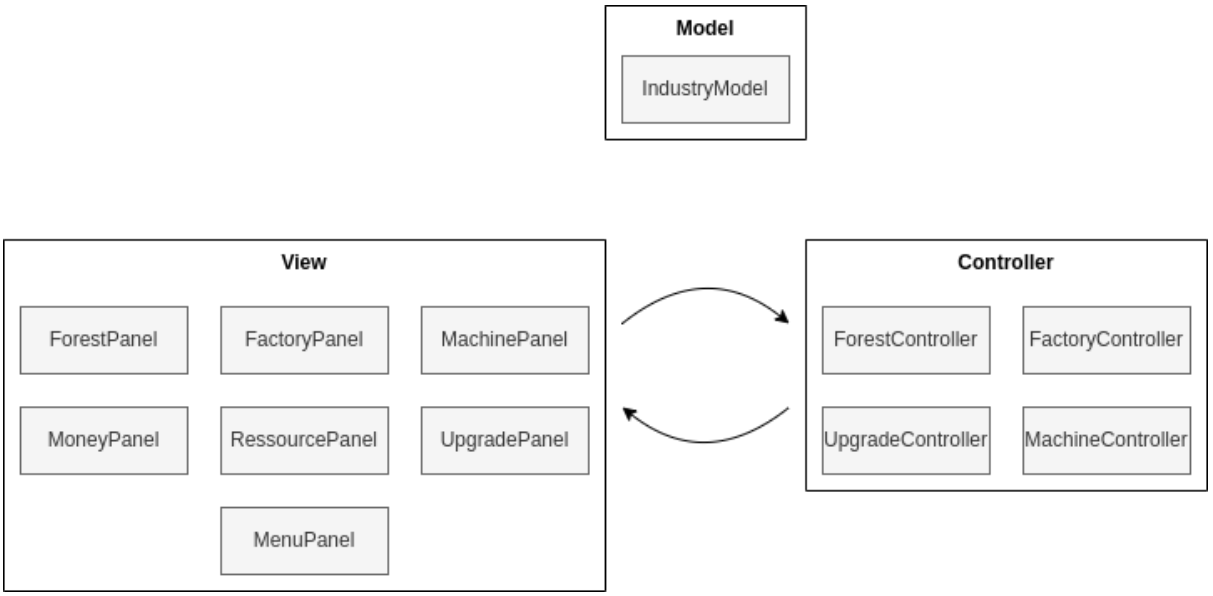
Diagramme d’Activités




Laisser le contrôle au joueur
Branche Commande
Apparition de Commande
Validation
Branche écran
Changer écran
Forêt
Usine
Sélection Terrain
Sélection Terrain
Acheter Terrain
Acheter Terrain
Améliorer Machine
Acheter machine
Branche Véhicule
Branche salarié
Branche Marketing
Branche Véhicule
Améliorer Véhicule
Branche écran
Branche Marketing
Branche salarié
Branche salarié
Engager salarié
Améliorer salarié
Branche écran
Branche Marketing
Branche Véhicule
Branche Marketing
Embaucher Marketing Manager
Licencier Marketing Manager
Branche salarié
Branche écran
Branche Véhicule
Mise à jour des ressources

Le diagramme d'activité décrit les différentes interfaces qui seront accessibles à l'utilisateur ainsi que la manière dont vont s'enchaîner les différents écrans. Cela nous permet de déterminer quels sont les processus parallèles que nous devrons mettre en place. En Java, cette parallélisation s'effectuera grâce aux Threads.
Dans notre diagramme, on peut voir les différentes tâches qu'il sera possible de réaliser pour l'utilisateur une fois la partie préférée. On se rend également compte de la manière dont nous devons initialiser nos interfaces grâce à la partie supérieure du diagramme. Enfin, afin de simplifier le diagramme, nous avons choisi de ne pas représenter des flèches entre chaque groupes d'états, nous avons préféré une vision simplifiée des groupes.

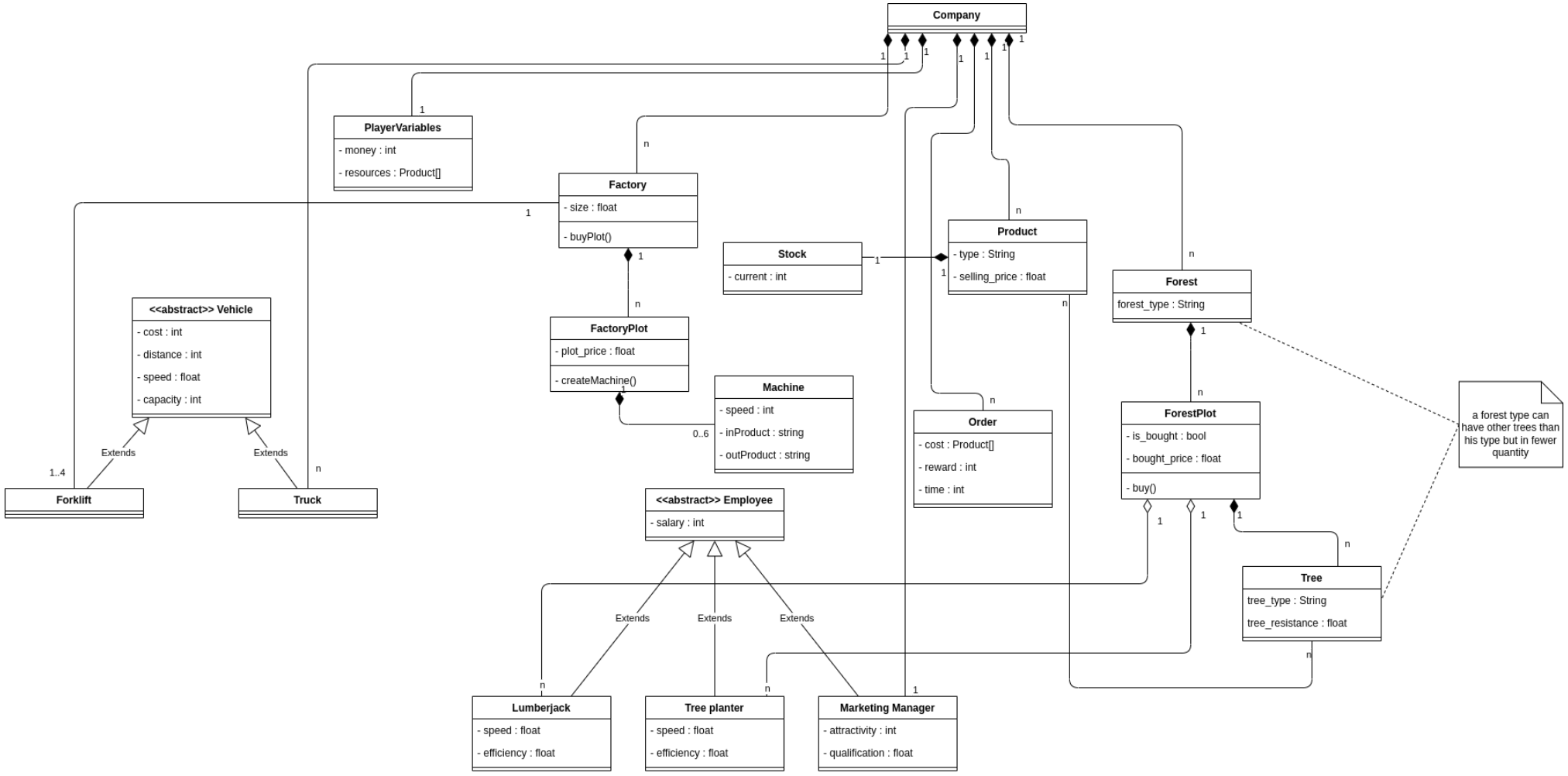
Architecture MVC



Afin de finaliser la démarche de création en suivant une méthodologie logicielle, nous avons été amené à utiliser une architecture Model View Controller (MVC). Nous avons réalisé un schéma nous permettant de déterminer les différents contrôles et vues sur notre application. On remarque cependant que notre application ne dispose que d'un seul modèle. Ce dernier va stocker l'ensemble des informations de l'application. Il disposera de la plupart des objets et c'est à partir de ce modèle que le contrôleur accédera aux données nécessaires au bon fonctionnement du jeu.

Diagramme de Classes

Enfin, nous avons réalisé un diagramme de classe UML afin d'avoir une idée la plus précise de ce que nous allions devoir concrètement concevoir. En effet, le modèle de classes représente l'ensemble des classes d'une application ainsi que les différents types de relations que l'on va retrouver. Ce diagramme ne permet pas toujours de savoir si l'on va choisir de représenter une relation par une liste, un tableau ou une simple imbrication de plusieurs objets différents dans un autre. Néanmoins, il nous permet déjà d'avoir une idée assez précise des classes qu'il sera nécessaire de créer, sans prendre en compte les spécificités du langage de programmation choisi.
Dans notre diagramme, on observe les premières notions d'héritage que nous utiliserons pour les véhicules et les employés afin de réutiliser du code. La plupart des classes seront contenues dans la classe Company. Ces éléments auront sûrement des composantes graphiques, mais nous n'avons pas encore choisi de les représenter sur ce diagramme. Nous ne connaissons pas encore la composition des éléments interactifs de chaque vue, mais elles contiendront une multitude d'éléments graphiques que nous n'avons pas encore déterminés. Nous avons néanmoins une vision générale de ce que nous voulons réaliser.


Company
PlayerVariables
- money : int
- resources : Product[]
Factory
- size : float
- buyPlot()
Stock
- current : int
Product
- type : String
- selling_price : float
Forest
forest_type : String
<<abstract>> Vehicle
- cost : int
- distance : int
- speed : float
- capacity : int
FactoryPlot
- plot_price : float
- createMachine()
Machine
- speed : int
- inProduct : string
- outProduct : string
Order
- cost : Product[]
- reward : int
- time : int
ForestPlot
- is_bought : bool
- bought_price : float
- buy()
a forest type can have other trees than his type but in fewer quantity
Forklift
Truck
<<abstract>> Employee
- salary : int
Tree
tree_type : String
tree_resistance : float
Lumberjack
- speed : float
- efficiency : float
Tree planter
- speed : float
- efficiency : float
Marketing Manager
- attractivity : int
- qualification : float
Extends
Extends
Extends
Extends
Extends
1
n
0..6
1..4

Conclusion

Avec la réalisation des diagrammes présentés ci-dessus, nous avons pu déterminer quelles fonctionnalités implémenter en priorité, afin de ne pas perdre de temps, et pour éventuellement disposer du temps nécessaire à l'ajout des fonctionnalités les plus superflues à la fin du projet.
L'analyse du déroulement de la partie nous permettra de diviser le projet efficacement en sous-tâches. On pourra ainsi savoir quelle classe sera chargée d'en créer d'autres en suivant une hiérarchie de classes.

Grâce aux diagrammes, nous disposons d'une direction à suivre pour nous guider dans la réalisation du projet, en conséquence d'une meilleure visualisation des attentes.
Cette structure n'est pas définitive et peut toujours être modifiée dans le contexte du développement en Java, avec la découverte de nouvelles contraintes. En effet, les diagrammes réalisés sont normalement les mêmes quel que soit le langage de programmation. Il ne permet donc pas de prendre en compte les spécificités du langage Java et de la bibliothèque graphique que nous utiliserons.

Néanmoins, nous avons pu déterminer comment intégrer les différents atouts fournis par Java au sein du projet, comme l'utilisation des Threads, et nous avons ainsi établi une structure qui nous semble appropriée pour la réalisation de la tâche donnée.

Cette étape aura servi à nous familiariser avec la démarche de génie logiciel à travers sa pratique dans un projet concret, ce qui s'appliquera également à nos futurs projets.