

Rapport Projet Conception Objet

Elouan Boiteux
Aymeric Mariaux
Killian Mathias

9 décembre 2025

Table des matières

1	Introduction	3
2	Diagrammes UML	4
2.1	Diagramme de classe	4
2.1.1	Architecture MVC	4
2.1.2	Model	4
2.2	Diagramme de séquence	4
3	Patrons de conception	6
3.1	Patron Observateur	6
3.2	Patron Factory	6
3.3	Patron Singleton	7
4	Conclusion	7

1 Introduction

Ce projet consiste en la modélisation d'un jeu : DorfJavatik. Ce dernier est inspiré d'un jeu existant : DorfRomantik. Ce jeu se joue à un seul joueur et consiste en un plateau de 25 cases dans lequel on peut placer des tuiles. Le joueur peut tourner et placer une tuile à chaque tour sur une des cases libres du plateau. Chaque tuile possède 4 zones : Nord, Sud, Est et Ouest. Chaque zone peut être un biome parmi les suivants : Rivière et lac, forêt, champ, village et montagne. L'objectif est de maximiser le score en faisant correspondre les biomes des tuiles adjacentes (par exemple, connecter une forêt à une autre forêt). A noter qu'à chaque tour, la tuile tirée est générée aléatoirement.

2 Diagrammes UML

La première étape dans le développement d'un jeu est d'abord d'analyser et d'établir le cahier des charges. Pour ce faire nous utiliserons différents diagrammes UML, en commençant par le diagramme de classe.

2.1 Diagramme de classe

2.1.1 Architecture MVC

L'objectif de ce diagramme de classe est de mettre en évidence les différentes composantes de notre jeu. Dans un premier temps, nous avons décidé de suivre l'architecture MVC (Model View Controller) car il est explicité dans le sujet que l'affichage devait être en temps réel. Or, si l'affichage est explicitement cité alors nous avons pensé que suivre cette architecture en utilisant les Views serait pertinent.

Dans notre diagramme, la classe Vue jouera le rôle de View, la classe Partie jouera le rôle de Controller car c'est avant tout elle qui va gérer le comportement de notre jeu et enfin le reste jouera le rôle de Model car ce sont des structures de données qui seront utiles au bon fonctionnement de notre jeu sans gérer la logique derrière.

2.1.2 Model

Dans un diagramme de classe de classe, le choix des structures est essentiel au bon fonctionnement du produit final.

Dans un premier temps, pour notre tuile, nous avons trouvé pertinent de stocker les données dans différentes structures : Tuile, Zone, Biome et Couleur. Etant donné que la tuile possède 4 zones on a établi un lien 1 -> 4 de la tuile à la zone. Chaque zone possède un type parmi l'énumération Type Zone (Nord, Sud, Est et Ouest) ainsi qu'un biome associé. Un biome est soit : Rivière, Montagne, Village, Forêt et Champ. Pour ce qui est des couleurs des biomes, ça sera la vue qui gérera l'affichage de ces dernières.

Puis il faut aussi modéliser le plateau de jeu, qui peut être composé de 0 (au début de la partie) à 25 tuiles (à la fin de la partie).

2.2 Diagramme de séquence

L'objectif du diagramme de séquence est de modéliser le comportement et les interactions que l'on souhaite entre les différents acteurs. Nous avons choisi comme acteurs le joueur, la vue, une tuile, la partie, l'usine de tuiles et le plateau de jeu.

Le rôle du joueur dans ce diagramme est de montrer ses interactions avec la vue car c'est bel et bien lui qui va effectuer les choix de jeu.

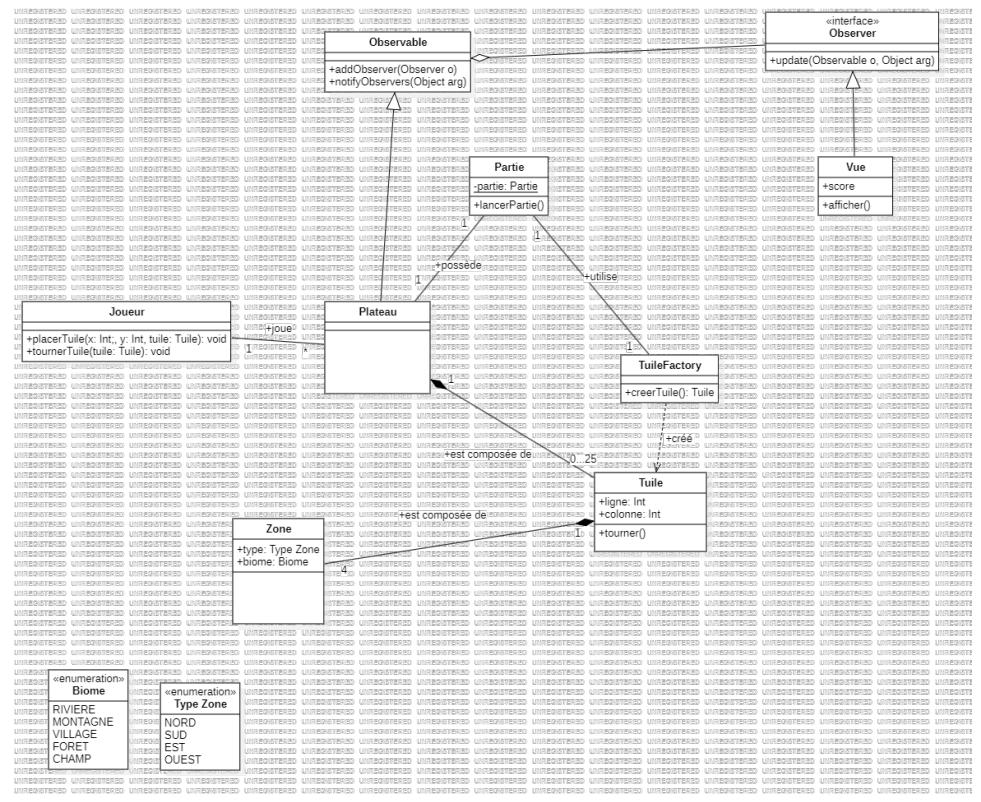


FIGURE 1 – Diagramme de classe

La vue va traiter les actions du joueur pour ensuite en faire une action concrète via la partie (le controller dans notre cas) ou la tuile, dans le cas où on souhaite la tourner.

En fonction du choix du joueur, la partie va effectuer différentes actions, comme lancer la partie, jouer une tuile et la placer sur le plateau.

En fonction de ces actions, la partie va faire appel à des classes telles que l'usine de tuiles pour générer une nouvelle tuile aléatoire, le plateau pour mettre à jour ce dernier.

Le plateau, comme montré dans le diagramme de classe est un élément Observateur, et va donc être observé, ici par la vue. Cela signifie donc que si le plateau change alors la vue sera notifiée et affichera les changements en conséquence.

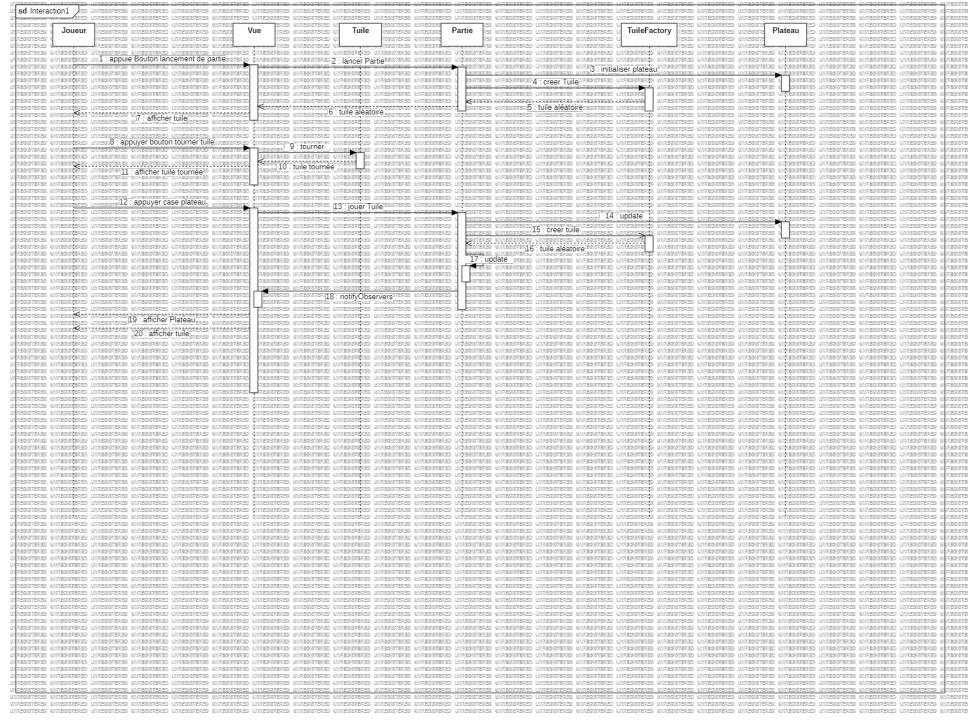


FIGURE 2 – Diagramme de séquence

3 Patrons de conception

Pour modéliser notre jeu DorfJavatik, nous avons utilisé trois patrons de conception différents : l’Observateur, la Factory et le Singleton.

3.1 Patron Observateur

Nous avons choisi de mettre en place le patron de conception Observateur pour que la Vue affiche en temps réel les données telles que le plateau ou le score du joueur. Pour ce faire, nous avons désigné la partie comme héritant de la classe Observable et la classe Vue héritant de Observer. Cela signifie qu'à chaque fois que la partie informe la vue d'un changement alors la vue est mise à jour.

3.2 Patron Factory

Nous avons choisi d'implémenter le patron de conception Factory, car nous voulions qu'à chaque tour une tuile soit générée aléatoirement. Le but était donc de déléguer la tâche de création aléatoire de ces tuiles à une entité différente de partie. Le patron de conception Factory correspondait

parfaitement à notre besoin car il délègue la création d'un élément d'une classe à une usine (factory). Ici notre usine est la classe Tuile Factory qui possède une méthode créerTuile() et qui crée une tuile aléatoire.

3.3 Patron Singleton

Enfin, nous avons décidé d'utiliser le patron de conception Singleton car nous voulions que la partie soit unique. En effet, étant donné que ce n'est pas un jeu en ligne et que c'est un jeu à un seul joueur, il serait inutile de créer plusieurs instances du jeu. Nous avons donc utilisé ce patron dans le but de limiter notre partie à être unique.

4 Conclusion

Votre conclusion ici.