Université Marie et Louis Pasteur
UFR Sciences et Techniques

Licence 3 Informatique – Année 2025-2026

RAPPORT DE PROJET DE CONCEPTION OBJET

Module Conception Objet

Modélisation d'un jeu de plateau

Auteurs :
Elouan BOITEUX
Aymeric MARIAUX
Killian MATHIAS

Professeur :
M. DARTOIS Luc (Professeur de Conception Objet)

Table des matières

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

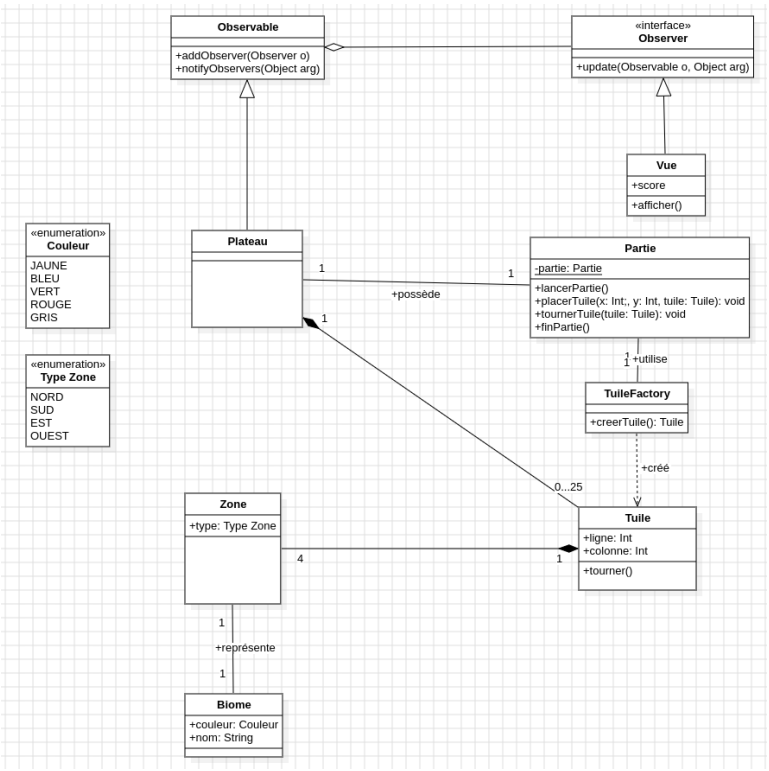

FIGURE 1 – Diagramme de classe

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, 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. Étant 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, ce 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 d'objet

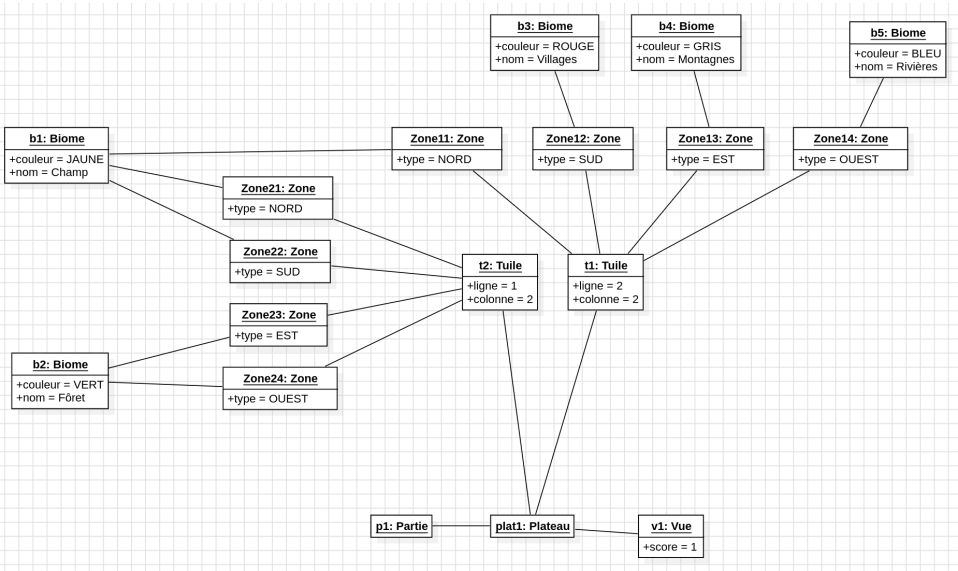

FIGURE 2 – Diagramme d'objet

L'objectif du diagramme d'objet est de modéliser un exemple concret de notre jeu. Nous avons choisi de montrer le comportement de notre jeu en montrant le comportement d'une partie avec 2 tuiles sur le plateau. Elle-même composée de 4 Zones chacune.

Chaque Zone possède un biome, une couleur et un nom.

Dans notre exemple il y a 5 Biomes :
— Rivière : Bleu
— Montagne : Gris
— Village : Jaune
— Forêt : Vert
— Champ : Jaune

Les deux tuiles sont composées de différents biomes et sont placées de manière à ce que les biomes soient connectés. Ce qui signifie que le nord de la première tuile est connecté au sud de la seconde tuile, cela génère donc un score de 1 dans la vue.

2.3 Diagramme de séquence

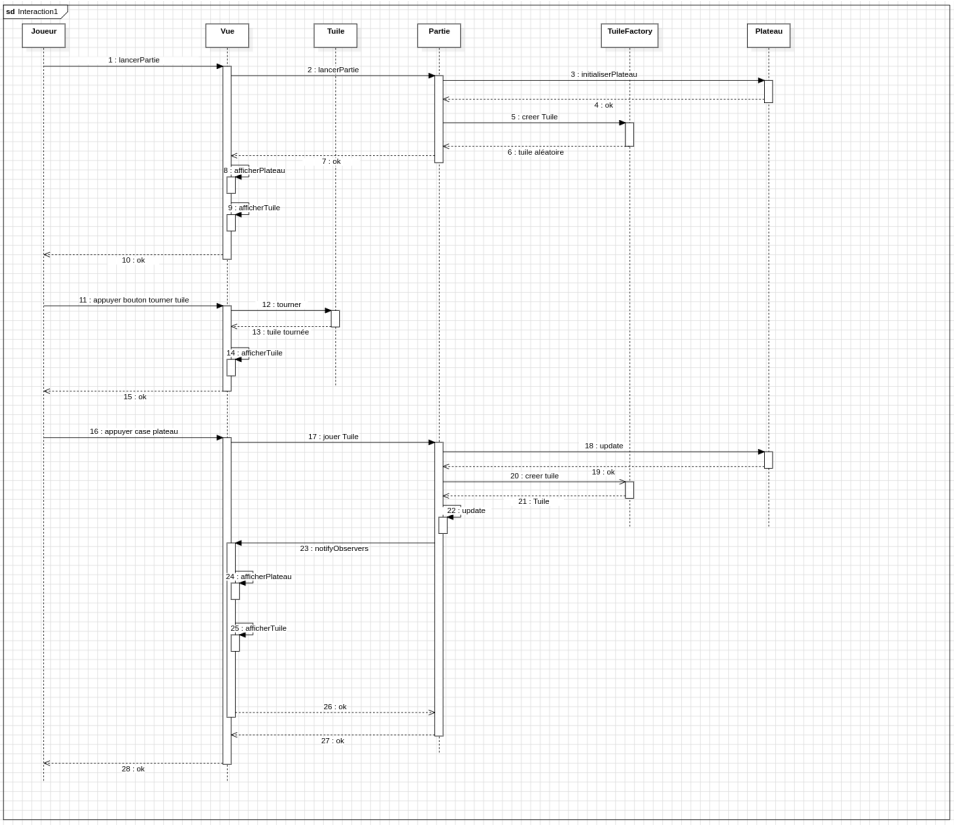

FIGURE 3 – Diagramme de séquence

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.
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 Observable, 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.

2.4 Diagramme de cas d'utilisation

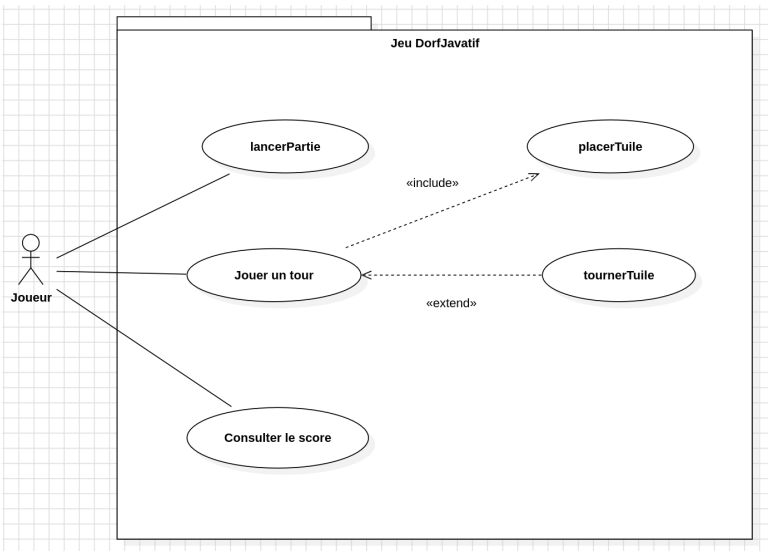

FIGURE 4 – Diagramme de cas d'utilisation

L'objectif du diagramme de cas d'utilisation est de montrer les différents cas d'utilisation de notre jeu. Nous avons donc modélisé les différentes actions que le joueur peut effectuer lors de la partie.
Le joueur est l'acteur principal et peut interagir avec le système `Jeu` via plu-

sieurs fonctionnalités :

— **Lancer la partie** : Initialise le plateau de jeu et la pioche de tuile.
— **Jouer un tour** : Permet d'obtenir la tuile suivante à placer.
— **Placer la tuile** : Permet de placer une tuile sur le plateau. (`include` de jouer un tour)
— **Tourner la tuile** : Change l'orientation de la tuile actuelle pour optimiser les connexions de biomes. (`extend` de jouer un tour)
— **Consulter le score** : Permet de voir le score actuel.

2.5 Diagramme d'état-transition

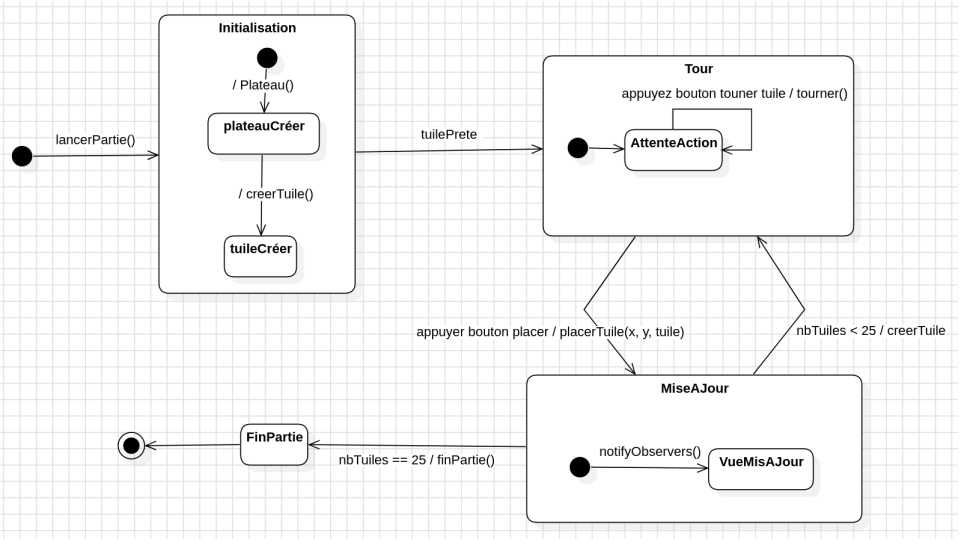

FIGURE 5 – Diagramme d'état-transition

L'objectif du diagramme d'état-transition est de montrer les différents états que peut prendre notre jeu au fur et à mesure du déroulement de la partie. Nous avons donc modélisé les différentes étapes de la partie de l'initialisation au fonctionnement de la partie.

Les états sont les suivants :

— **Initialisation** : L'état composite initial de la partie, où le joueur peut lancer la partie.
— **PlateauCréer** : L'état après que le plateau ai été créé.
— **TuileCréer** : L'état après que la tuile ai été créée.
— **Tour** : L'état composite où le joueur peut jouer un tour.
— **Attente Action** : L'état où le joueur peut placer une tuile, la tourner ou consulter le score.

— **Mise à jour** : L'état composite où la vue est mise a jour après avoir placer une tuile pour que le joueur puisse consulter le score.
— **Fin de partie** : L'état final de la partie, où le joueur peut consulter le score.

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 notifie 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

La modélisation de notre jeu DorfJavatik a été réalisée avec succès. Nous avons utilisé des patrons de conception appropriés pour assurer la cohérence et la lisibilité de notre code. Nous avons également utilisé des diagrammes pour visualiser les relations entre les classes et les objets de notre système.