

# POOIG

---

DONI Diane

KAÏS Lamia

# Sommaire

## Introduction

## Objectifs

1. Vue graphique
2. Vue textuelle

## Caractéristiques

3. Classes et méthodes
4. Relation entre classes

## Grandes étapes

5. Modélisation
6. Gestion des difficultés

## Conclusion

## Introduction

Le sujet du projet étant la réalisation du jeu AZUL, la première étape a été d'essayer de se familiariser avec le jeu en y jouant grâce au lien fourni avec le sujet. Après avoir relativement compris le contexte, on a alors commencé à coder. Une devant réaliser la partie textuelle, et l'autre la partie graphique.

## Objectifs

### 1. La vue graphique

La modélisation de notre vue graphique a été inspirée par le jeu sur Rojo. Ainsi, en y jouant, nous avons eu des idées sur comment procéder. Les classes utilisées pour modéliser cette vue, sont les mêmes que celles utilisées pour la vue textuelle mais se terminant par vue. Le fonctionnement devrait toutefois être le même. Les options sont présentées aux joueurs et c'est à eux de suivre les instructions afin de jouer une partie.

### 2. Vue textuelle

Comme Azul est un jeu de société n'étant pas très adapté à une utilisation sur terminal, on a donc traduit comme on le pouvait la façon d'on on s'imaginait le fonctionnement du jeu. On s'est principalement servi des noms appliqués au jeu pour trouver les noms des classes tels que: Manche, Fabrique, Jeu, Défausse, Joueur etc. Ainsi, chaque classe avait des attributs permettant de les relier entre eux afin qu'il puisse y avoir une interaction.

## Caractéristiques

### 3. Classes et méthodes

Afin de ne pas avoir de fichiers trop lourds et dans le but d'avoir un code clair et concis, nous avons créé une vingtaine de classes dont celles incluant la vue. Les classes utilisées pour l'interface graphique interagissent avec celles utilisées pour le jeu sur terminal. Chaque classe permet d'initialiser une ou des instances qui seront utiles dans la réalisation du jeu. Par exemple, pour faire une partie, il faut d'abord qu'il y ait un nombre correct de joueur, et qu'ensuite en fonction du nombre de joueurs, on puisse trouver le nombre de fabriques nécessaires pour une partie spécifique. Ces fonctionnalités sont gérées par les classes Joueur, Manche et Fabrique. Cependant il peut arriver qu'en fonction du nombre de joueurs, certaines fonctionnalités peuvent changer et donc, la classe Fabrique devient une interface, qui est implémentée par CommonToAllPlayers dont le nom est assez explicite. Et cette classe étend TwoPlayers, ThreePlayers, FourPlayers. De plus, pour pouvoir faire une partie, il est nécessaire d'avoir des tuiles. Et ces dernières sont générées par la classe Tuiles qui peut apporter certaines modifications aux tuiles utilisées. Les classes CentreDeTable et Plateau modélisent le centre de table et les plateaux des joueurs. De plus, la plupart des attributs et méthodes de la classe CentreDeTable sont static parce qu'ils sont communs à tous les joueurs. Les noms des méthodes sont tout aussi explicites et sont soit en anglais soit en français. On a par exemple : addPlayers(), remplirFabrique(),commence(),nextPlayer() etc.

### 4. Relation entre classes

Comme indiqué précédemment, les classes sont liées entre elles. On sait que pour une manche, il est nécessaire d'avoir des tuiles, un centre de table, des fabriques, des plateaux et surtout des joueurs. Le jeu commence quand une manche est débutée. De ce fait, la méthode commence() est définie dans la classe Manche. Ensuite cette méthode en fait appel à d'autres en s'imbriquant de façon à avoir une suite logique. Cependant, pour qu'il n'y ait pas d'ambiguïté dans le lancement du jeu, une classe Jeu a été créée. Et c'est elle qui permet de savoir sur quel interface, le jeu va se dérouler. Si le nombre de joueurs est 2, on se servira plus de la classe TwoPlayers que s'il s'agissait de 3 ou 4 joueurs. Les erreurs sont gérées au mieux afin que le jeu soit le plus fluide possible, étant donné que sur le terminal, il faut une utilisation poussée de la classe Scanner.

## Grandes étapes

### 5. Modélisation


La modélisation textuelle est un peu différente de celle graphique du fait que la vue n'est pas dessinée. Seul l'état du jeu est donné avec des représentations visuelles des états des fabriques et du centre de table afin que le choix puisse se faire. Pour la vue graphique, on a pu prendre exemple sur le lien donné dans le sujet. On s'en est servi comme base pour implémenter l'interface graphique. Le diagramme de classes peut être linéairement donné comme suit:

- Manche
- Interface Fabrique
- CommonToAllPlayers implements Fabrique
- TwoPlayers extends CommonToAllPlayers
- ThreePlayers extends CommonToAllPlayers
- FourPlayers extends CommonToAllPlayers
- Plateau
- Plateau.Defausse
- Joueur
- Tuiles
- Sac extends Tuiles // qui contient l'ensemble des tuiles
- CentreDeTable
- Jeu

Et une fonction auxiliaire Objet étendant Tuiles qui confère ses caractéristiques à cette dernière. Ces classes servent pour la représentation textuelle. Celles allouées à la vue graphique ont relativement le même nom avec en plus "vue".

### 6. Gestion des difficultés

La modélisation semblait être la partie la plus difficile à effectuer. Cependant, au fur et à mesure que la forme du jeu apparaissait, l'utilisation des bons attributs



pour les bonnes fonctions permettaient de trouver des solutions. La plus grande difficulté rencontrée au niveau de la vue textuelle était l'affichage de l'état du plateau du joueur. Et aussi, le score. Certes, il fallait donner l'avancée du jeu au fur et à mesure mais ce n'est pas évident quand on a pas de défausse ni de planche de score. Et aucune indication n'est donné sur comment les points sont comptés. On a dû trouver notre propre modélisation. De plus les erreurs ont été gérées du mieux que nous pouvons mais certaines nous ont pris plus de temps que les autres. Notamment, les entrées invalides qui ne correspondent à aucun choix possibles

## **Conclusion**

En Définitive, nous avons fait en sorte d'avoir au moins une des vues possibles à défaut des 2. Les joueurs mêmes s'ils n'ont aucune notion du jeu, en le commençant peuvent le jouer et en comprendre le fonctionnement au fur et à mesure. Au cours des semaines, plusieurs modifications ont été apportées au code afin qu'il y ait le moins possible d'erreurs, que le jeu soit continue et qu'il termine.