

# **SAÉ 2.01**

Développement d'une application

---

# **SAÉ 2.02**

Exploration algorithmique

29.03.2022 - 03.06.2022

Maxime Capel & Bayram Gokcen



Introduction	3
1. Présentation de la SAE	3
2. Étape 1 : Mise en place du matériel	4
3. Étape 2 : Mise en place du plateau de jeu	7
4. Étape 3 : Les joueurs et l'arbitre dans la place	10
5. Étape 4 : Interaction joueur & arbitre : jouer une tuile	12
6. Étape 5 : Rôle de l'arbitre	15
7. Étape 6 : Interaction joueur & arbitre : acheter une action supplémentaire	
18	
8. Étape 7 : Interaction joueur & arbitre : échanger toutes les tuiles	20
10. A propos de la qualité ...	23
Qualité de code :	24
Qualité du développement	25

# Introduction

Dans le cadre de **SAEs**, nous avions pour objectif de **concevoir** un jeu de **Lattice**, sous le langage **Java**. Nous allons alors détailler la **réalisation** de cette **application**. Dans ce but, nous expliquerons dans un premier temps, les différentes étapes de **conception** et **d'implémentation**, puis dans un second temps, les détails relatifs à la **qualité de code**.

## 1. Présentation de la SAE

### Présentation du client et de son besoin

Cette SAÉ concerne le développement d'une forme simplifiée du jeu LATICE.

Le client à demandé que des simplifications suivantes soient apportées au jeu :

- le jeu ne pourra être joué que par **2 joueurs**

- le jeu ne comportera **pas de pierres**, mais les pierres seront remplacés par **des points**, à savoir :

- une demi-pierre fera gagner **1 point**

- une pierre de soleil fera gagner **2 points**

- le jeu n'aura **pas de limite de points** (pas de limite de pierre soleil par extension)

- l'achat **d'une action supplémentaire** coutera **2 points**

- l'action d'échanger **échangera directement tout le rack** (c'est-à-dire que toutes les tuiles du rack seront changés, il n'y aura pas de choix possible pour un échange : ce sera tout le rack ou rien)

- le jeu ne comportera **pas de tuile de vent**

... Ce qui implique de revoir la règle de fin de partie ...

Bien sûr, **un joueur peut gagner si son rack est vide et sa pioche est vide**, mais sans la tuile vent cela paraît plus difficile de terminer une partie, c'est pourquoi, le client a souhaité ajouter la règle suivante pour que la partie ait une durée de jeu raisonnable :

**Pour 2 joueurs, la partie se jouera en 10 cycles maximum (1 cycle = nombre de joueurs / tours).** Le gagnant est le joueur qui a posé le plus de tuiles.

### Présentation de la stack technique

L'application sera implémentée en **Java**, en utilisant **JavaFX** pour l'interface Homme-Machine conformément aux modules R2.01 - Développement orienté objet et R2.02 - Développement d'application avec IHM.

Des jeux d'essais devront également être mis en place pour tester que le comportement de l'application est bien conforme aux spécifications du cahier des charges ceux-ci seront faits avec le module **JUnit Test**.

Les diagrammes UML ont été générés par l'extension **PlantUML** et créés avec **StarUML**. Les tuiles ainsi que le fond du jeu ont été fait avec le logiciel **Figma**. La vidéo du menu est un montage réalisé avec le logiciel **iMovie**.

Nous avons utilisés **git** comme gestionnaire de version et nous avons poussé notre code sur GitUnilim en privé avec des droits pour les enseignants. Nous avons découvert **CodeTogether**, en effet cette extension nous a permis de faire du code à deux sur une même session afin de limiter les commit pour éviter de trop « polluer » le repository.

Voici le lien vers notre dépôt git pour avoir accès au projet : <https://git.unilim.fr/gokcen1/latice>

## Présentation de notre organisation du travail

Notre travail a été réparti sur les différentes SAÉ tout au long des séances. Cependant, le temps jouant contre nous, nous avons été contraint de travailler majoritairement sur notre temps libre quitte à sacrifier des nuits de sommeil.

Nous avons mené le développement du jeu LATICE jusqu'à l'étape 7 c'est-à-dire qu'il est fonctionnel, que l'on peut jouer une partie tout en aillant la gestion des scores.

## 2. Étape 1 : Mise en place du matériel

### Description de l'étape

Pour cette étape nous avons dû mettre en place le début, la base du jeu "Latice" c'est à dire :

- regrouper toutes les tuiles du jeu Latice et les mélanger.
- Pouvoir distribuer les deux piles de tuiles équitablement et aléatoirement aux deux joueurs dans deux pioches.
- Pouvoir tirer au sort de 5 tuiles à partir d'une pioche et installer sur un rack

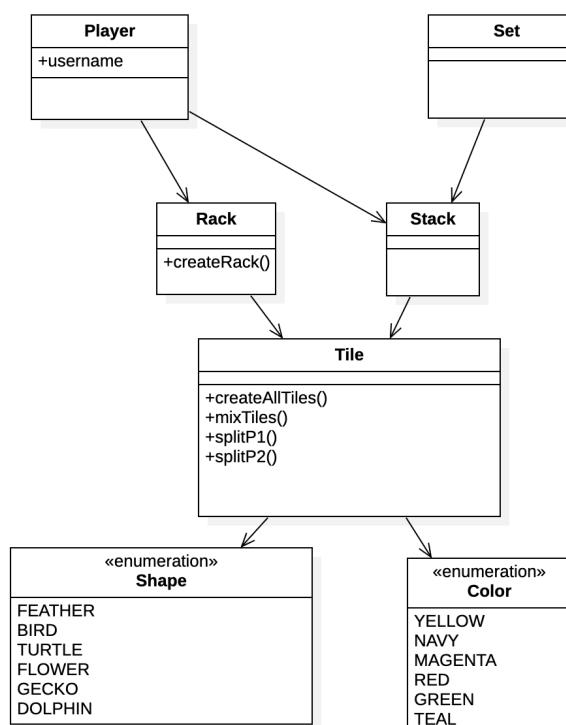
Il s'agit d'actions racines, qui seront utilisées pour mettre en jeu notre programme et qui nécessiteront les premières et les plus importantes classes de notre jeu.

Il faut donc beaucoup réfléchir à la conception car il s'agit des fondements de notre programme. Se tromper, dès le début, pourrait être regrettable.

Cette étape nous a pris environ 4 heures.

## Conception

### DIAGRAMME DE CLASSE



Nous sommes passés par plusieurs diagramme avant d'arriver à cette version qui est celle qui nous semble la meilleure par rapport au sujet.

## **PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION**

Pour la structure de base, nous avons beaucoup hésité.

En effet, nous ne savions pas quelles classes étaient essentielles et nous ne savions pas où mettre les méthodes, nous avons donc essayé plusieurs conceptions, avec par exemple une classe qui regroupait toutes les tuiles du jeu. Cependant, cette classe nous semblait inutile.

Puis, nous avons pensé à une classe joueur qui aurait tous les attributs et toutes les méthodes, seulement, cela semblait encore trop tôt pour crée une classe Joueur qui de plus aurait des méthodes inappropriées.

De plus, étant donné que ni moi, ni mon co-équipier n'avons jamais réalisé de projet de développement (orienté objet), nous avons du mal à commencer, en effet il s'agit d'une étape majeure et c'est pourquoi nous avons passé énormément de temps à réfléchir.

## **Implémentation**

### **COMMENT AVEZ-VOUS MENER LA PHASE D'IMPLÉMENTATION ?**

Afin d'implémenter au mieux le début nous avons travaillé en binôme.

En effet étant donné qu'il s'agissait du début, nous avons préféré travailler ensemble, l'un faisait une fonctionnalité, l'autre travaillait et vérifiait avec lui.

Comme ça, nous sommes partis sur les mêmes bases, si l'un se posait une question, il pouvait alors la poser à l'autre et comprendre.

En somme, nous ne nous sommes pas données de rôles directement dès le début, nous avons préférés travailler en pair-programming.

Cependant, dès que les bases étaient posées, nous avons décidé de nous répartir des rôles. L'un a fini la première étape, le deuxième a commencé la deuxième étape. En effet, puisque que nous n'avancions pas assez avancé par rapport à notre date limite mais aussi par rapport aux autres groupes, nous avons décidé de scinder les rôles.

Nous avons effectué environs 30 commits pour mener à bien l'implémentation de cette étape. En effet, nous avons mis énormément de temps à commencer, mais de nombreux commits sont inutiles et auraient pu être évités. De plus, comme nous l'avons dis plus tôt, nous avons décidé de scinder le groupe en 2.

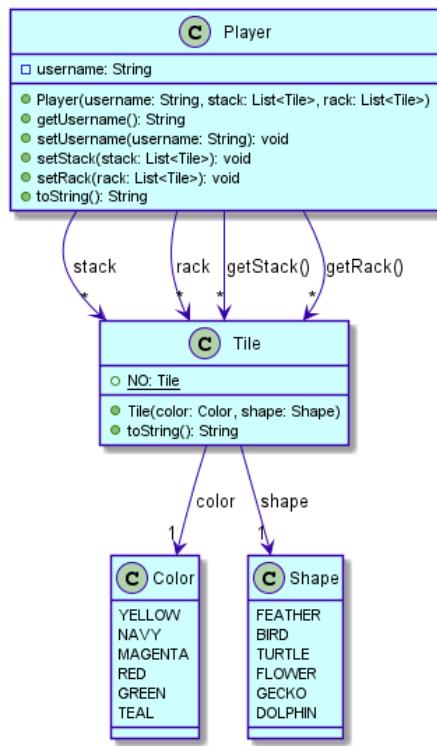
De ce fait, le début de l'étape 2 est déjà réalisé et certains commits sont plus liés à l'étape 2 qu'à l'étape 1.

### **INTERFACE HOMME MACHINE**

Au début, le choix d'une interface "homme-machine" en console a été réalisé. En effet, cela ne sert à rien d'utiliser une réelle interface homme-machine pour afficher simplement des tuiles. Cela nous a permis d'afficher toute les tuiles créées, de les séparer équitablement entre les joueurs pour enfin créer un rack de 5 tuiles par joueur.

Étant donné que nous n'avons pas utilisé de réelle interface homme-machine, cette étape a été assez simple, il nous suffisait d'utiliser des `System.out.println( )`.

### **RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET**



## **PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION**

Il n'y a pas eu de réels problèmes ou de difficultés rencontrés durant la phase d'implémentation, nous avons juste réalisé une phase de refactoring en fin d'étape pour que tout colle avec le diagramme voulu et pour éviter la répétition.

En effet, nous avions passé beaucoup de temps sur la phase de conception, c'est pourquoi nous savions déjà ce qu'il fallait faire, tout le temps "perdu" en conception nous a permis d'implémenter de façon efficace. Comme quoi, il ne s'agissait pas réellement de temps perdu, mais seulement d'une étape obligatoire pour mieux réussir.

Même si la conception de base a changé au fur et à mesure, nous avons réussi à nous adapter.

# Jeux d'essais

## Démarche réflexive

Comme nous en sommes à l'étape 1, nous avons implémentés une conception simple du jeu avec uniquement le joueur avec un pseudo et une / des tuiles(s) avec une forme et une couleur. On peut le remarquer grâce aux diagrammes UML qui nous ont permis d'élaborer ces conceptions simples. Cette étape a été simple grâce à la méthode algorithmique c'est-à-dire que les éléments ont été découpés en éléments simples. Des tests ont permis de vérifier les potentiels erreurs et de revenir sur ce qu'il n'allait pas comme par exemple la relation entre tile et player qui avait un petit problème.

L'avancée du projet est suivie par le client grâce au dépôt git tout en aillant des algorithmes simples.

L'affichage en console a été un choix évident car nous n'avons pas encore vu ceci en javaFX, de plus, la console permet une bonne visibilité pour un début avec des méthodes simples.

## 3. Étape 2 : Mise en place du plateau de jeu

### Description de l'étape

A la fin de cette étape nous devons disposer d'un plateau de jeu vide (prêt à recevoir des tuiles) avec des cases soleils et la case lune.

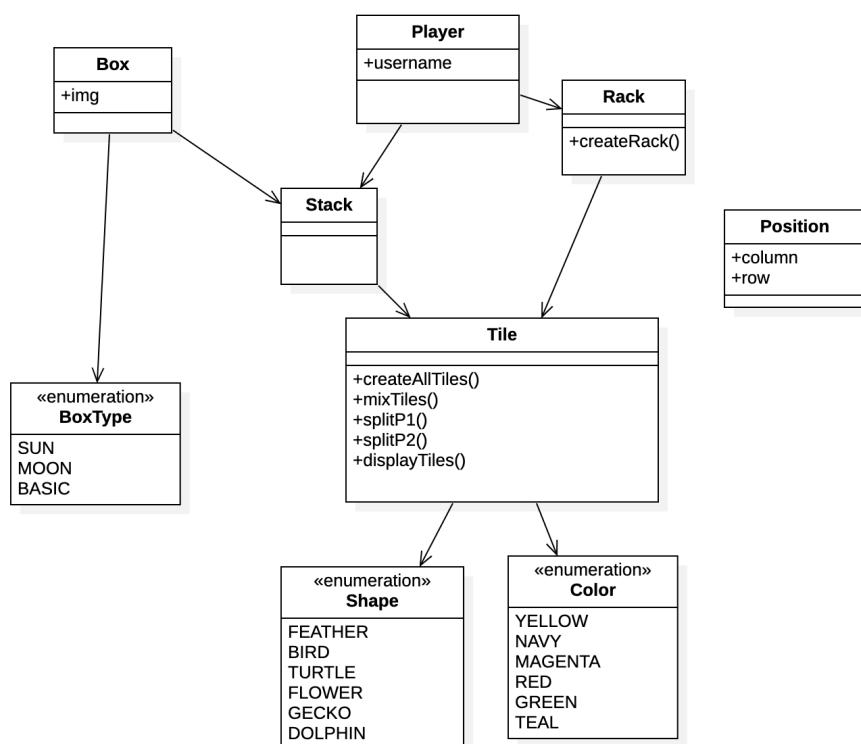
Cette étape est, comme la précédente étape, l'une des parties les plus importantes de la production de notre application. En effet, l'objectif ici est de créer notre espace de jeu, la zone où l'on va poser nos tuiles.

Il faut alors faire attention et passer un bon moment pour comprendre comment cette zone de jeu va être utilisée dans le futur, car en effet, il ne suffit pas de poser une zone de jeu inutilisable mais un réel espace de jeu qui pourra être utilisé dans les étapes futures. Il faut que notre plateau de jeu soit prêt à recevoir des tuiles.

Cette étape nous a pris environ 7 heures.

### Conception

#### DIAGRAMME DE CLASSE



## **PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION**

Pour cette étape, le plus gros problème lors de la conception a été de choisir avec quel composant nous allions créer notre plateau de jeu, en effet, nous avions le choix entre faire une liste ou d'utiliser des HashMap.

En effet, nous avons vu durant un TP réalisé quelques semaines auparavant, l'utilisation des HashMap alors nous avons décidé de les utiliser malgré le fait qu'il s'agisse de composants très neufs pour nous.

De plus, durant la conception nous ne savions pas comment mélanger la partie graphique et la partie pas graphique. En effet, nous avons eu du mal à savoir où est-ce qu'il fallait mettre les images et où est-ce qu'il ne fallait pas en mettre.

Finalement, nous avons eu assez de mal dans la conception en général de cette étape car nous ne savions pas ce qui marcherai et ce qui ne marcherai pas.

Nous ne savions alors pas s'il fallait faire une HashMap de tuile ou de boites qui contiennent les tuiles. Est-ce qu'on part sur quelque chose d'inutilisable ou non ?

## **Implémentation**

### **COMMENT AVEZ-VOUS MENÉ LA PHASE D'IMPLÉMENTATION ?**

Pour la phase d'implémentation, nous avons décidé de nous donner des rôles, dans un premier temps l'un du duo devait créer les images de tuiles en format *png*, car celles données étaient mal coupées.

Pendant ce temps, le second camarade continuait l'implémentation du plateau de jeu grâce aux HashMap. Même si nous avions bien préparé la conception, on s'est vite rendu compte que ce que l'on imagine ne fonctionnait pas. En effet, on a alors été obligé d'imaginer une conception nouvelle en cours d'implémentation.

Finalement, l'un d'entre nous s'est occupé de la partie Interface Homme-machine qui a été assez compliqué à réaliser, comme nous l'expliquerons dans la partie dédiée.

Nous avons effectué environ 10 commits pour mener à bien l'implémentation de cette étape.

### **INTERFACE HOMME MACHINE**

Un des nouveaux concepts qui a été difficile à assimiler a été l'interface homme-machine avec JavaFX. En effet, nous avons décidé de créer le plateau de jeu avec JavaFX afin d'obtenir une version graphique et beaucoup plus utilisable que les tableaux affichés dans la console.

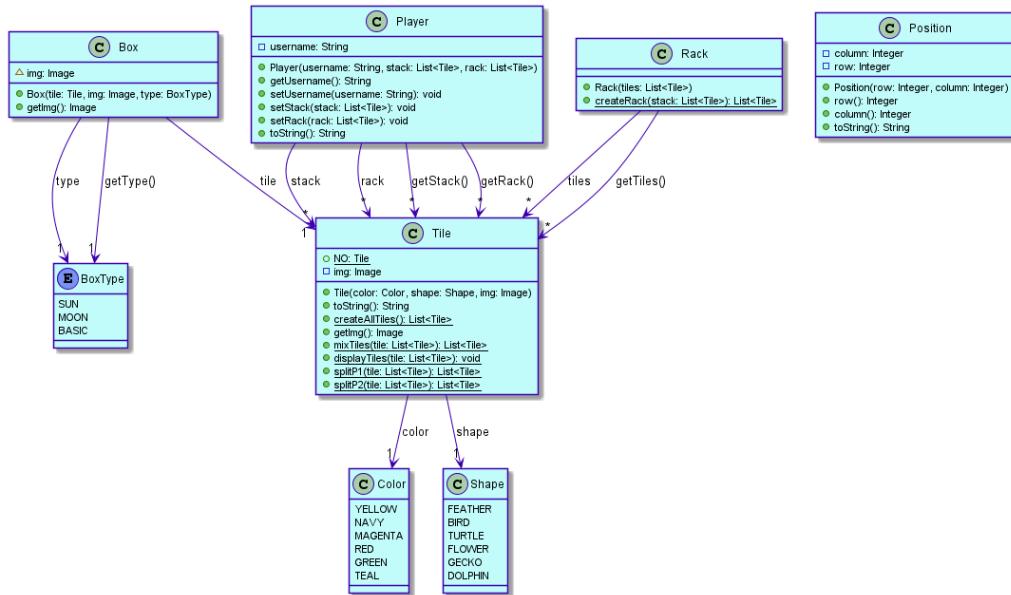
Il était pour nous alors, essentiel d'utiliser JavaFX car cela nous paraissait beaucoup plus intéressant.

Ce choix s'est accompagné malheureusement accompagné avec son lot de problèmes. En effet, nous avons, dans un premier temps, mélangé la partie graphique et la partie non graphique. Ce qui nous a ensuite donné des problèmes pour assurer les tests.

De plus, graphiquement nous avons choisi d'utiliser une GridPane pour le terrain, car la forme de celle-ci collait bien avec la forme du terrain de jeu. Les tuiles quant à elles étaient représentées par des images dans des ImageView.

Ce qui a été compliqué était de lier la GridPane constitué d'ImageView avec le HashMap que nous venions de créer. En effet, pour les futurs étapes, il était essentiel que chaque case (qui sont des ImageView) du plateau de jeu soit liée au HashMap et que l'on puisse récupérer ses coordonnées.

## RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET



## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION

Un des plus gros problème était le fait qu'on ne connaissait pas forcément bien les HashMap, par conséquent, on a perdu beaucoup de temps sur certaines choses qui pouvaient être assez simples.

Nous avons par exemple perdus plus de 2 heures de travail, car nous n'arrivions pas à sélectionner un certain élément du HashMap, finalement nous avons remarqué qu'il suffisait de générer les hashCode() et equals().

De plus, comme dit précédemment, l'implémentation de l'interface homme-machine a été très longue, nous nous y sommes pris à plusieurs fois avant d'obtenir quelque chose qui fonctionne.

## Jeux d'essais



# Démarche réflexive

Après plusieurs essais, l'utilisation du HashMap à été payante car une fois l'application lancée, le rack de chaque joueur et le plateau de jeu apparaissent grâce à une implémentation structurée. On obtient le résultat des jeux d'essais après divers essais sur l'affichage de cette IHM.

Une fois refaite les tuiles sont plus harmonieuses que celles d'avant. On s'est permis d'ajouter un fond pour que cela colle avec l'univers.

## 4. Étape 3 : Les joueurs et l'arbitre dans la place

### Description de l'étape

Pour cette étape nous devons :

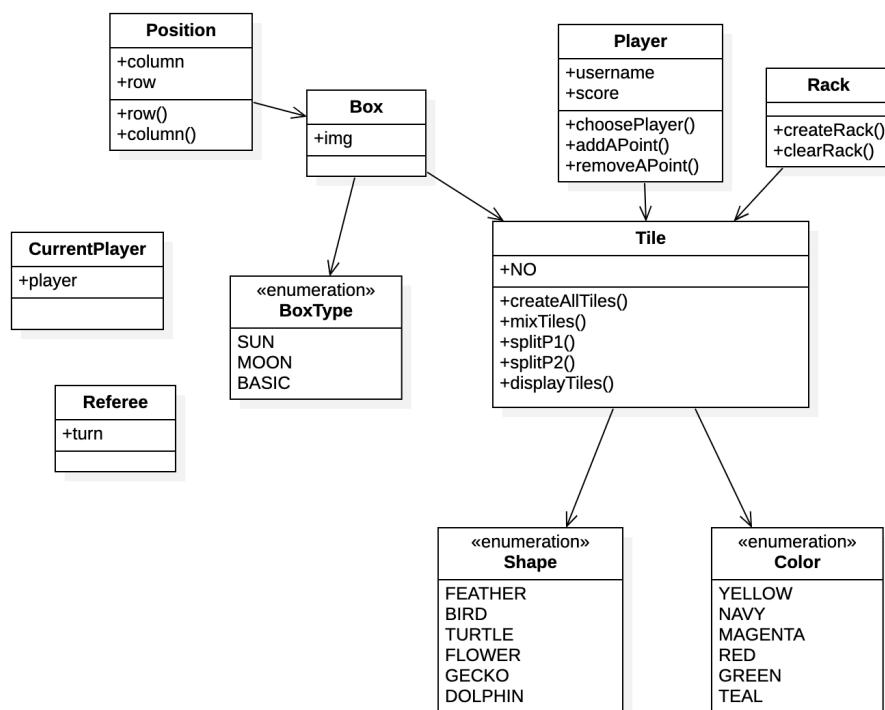
- Distribuer le matériel et commencer la partie
- Lancer la partie en choisissant un joueur aléatoirement
- Lancer un tour de la partie

Cette étape n'est pas la plus compliquée en soit, mais il s'agit du début de la mise en place des règles. En effet, on commence alors à distribuer le matériel et lancer les parties. De plus un mécanisme de hasard permet de choisir le premier joueur aléatoirement.

Il s'agit alors d'une étape importante où il faut penser aux futurs éventualités.  
Cette étape nous a pris environ 2 heures.

## Conception

### DIAGRAMME DE CLASSE



### PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION

Pour cette étape la plus grosse difficulté à été de faire la conception de la gestion d'un tour.

En effet, nous avions déjà fait la conception voire implémenté le système de distribution du matériel, nous savions alors comment faire.

Le système de hasard quant à lui n'était pas compliqué.

Cependant, nous avons eu du mal à réaliser la conception du système de tour. Nous n'arrivions pas à comprendre comment est-ce qu'on pouvait réussir à faire défiler les deux joueurs (l'un puis l'autre) et ce de façon à pouvoir interagir totalement avec le premier puis totalement avec le second.

Finalement, nous avons décidé de créer de nouvelles classes "arbitres" qui nous ont permis de compter les tours et de connaître le joueur actuel.

## Implémentation

### COMMENT AVEZ-VOUS MENÉ LA PHASE D'IMPLÉMENTATION ?

Durant l'implémentation de cette phase, nous avons décidé de se répartir les tâches de sorte à ce que uniquement l'un d'entre nous deux travaille sur l'implémentation et que l'autre puisse réaliser l'étape 4 qui est l'étape du Drag And Drop.

En effet, d'après la phase de conception il n'y avait rien d'urgent dans cette étape. Du coup, ce n'était pas grave si seulement une personne avançait cette étape toute seule.

De ce fait, pendant que nous avancions dans les étapes 4 et 5 ensemble car ces étapes sont dites assez compliquées, l'un d'entre nous réalisait petit à petit le système de tour de cette étape.

Nous avons effectué environ 10 commits pour mener à bien l'implémentation de cette étape.

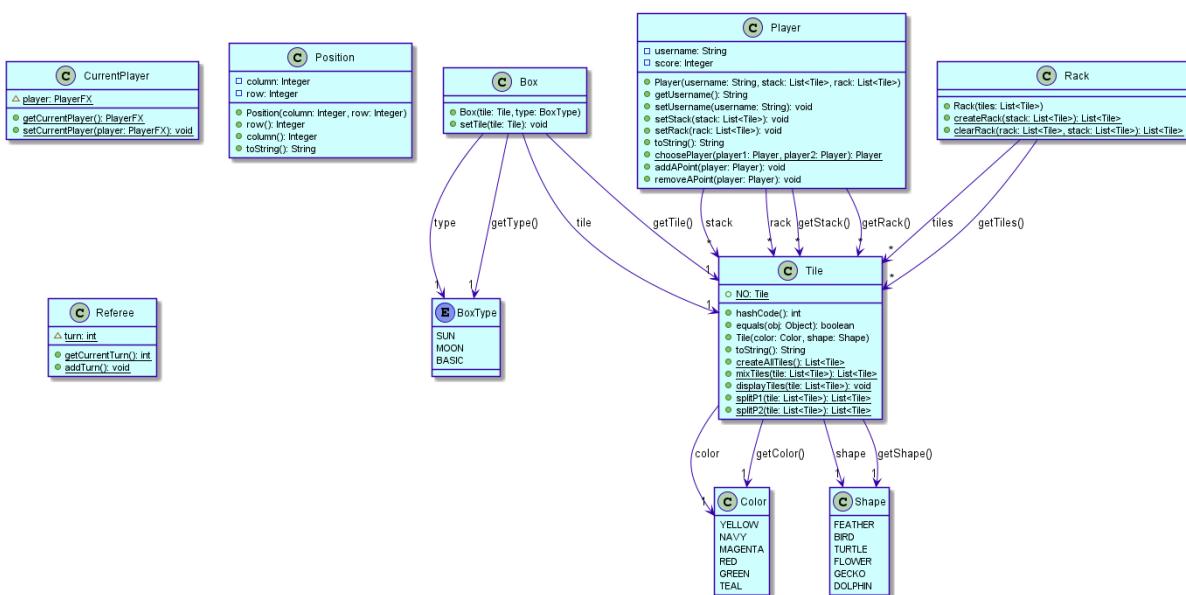
En effet, même si, pour réaliser cette étape nous avons mis beaucoup de temps, en soit on n'a pas beaucoup travaillé, elle a été réalisée sur le long terme.

### INTERFACE HOMME MACHINE

Pour cette étape, nous n'avons quasiment pas utilisé d'interface homme-machine. En effet, nous avons seulement demandé à ce que le nom du joueur soit affiché.

En effet, il s'agissait plus d'une étape arbitraire, permettant le déroulement du programme que d'une étape qui demande à être affichée.

### RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET



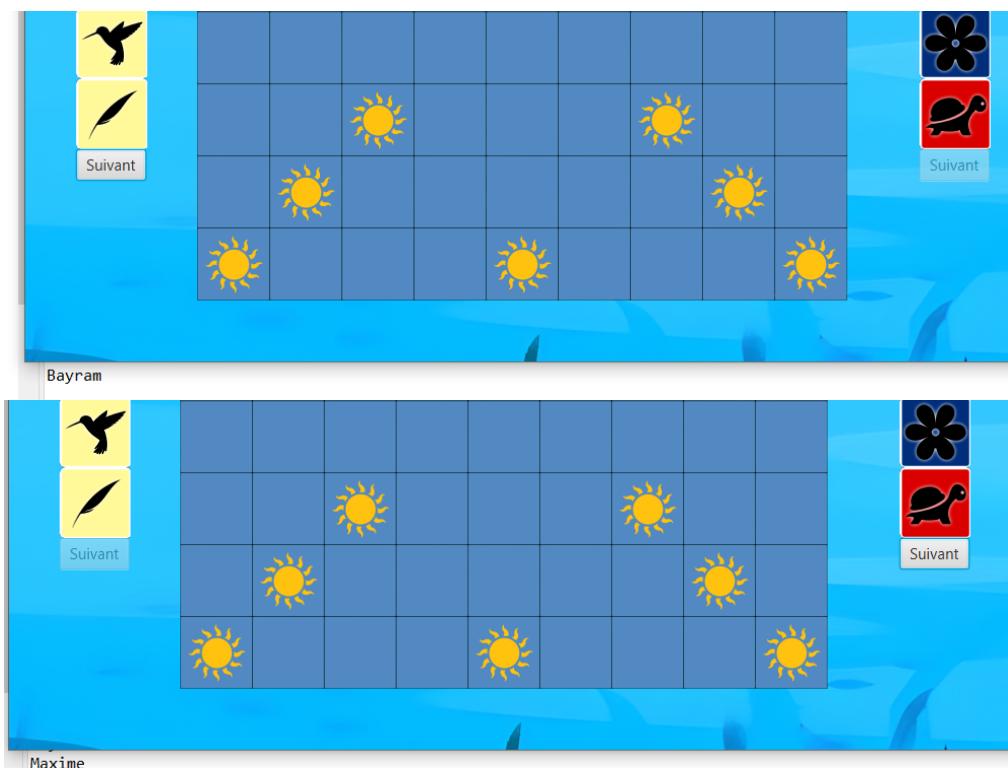
## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION

La gestion d'un tour a été un réel défi pour nous, en effet il a fallu l'implémenter en même temps que les étapes 4 et 5 par manque de temps.

Nous avions alors, durant la phase de conception, imaginé des nouvelles classes arbitres qui nous auraient permis de connaître le nombre de tour et le jour actuel. Cependant, nous ne savions pas comment lier ces classes aux classes déjà présentes.

La personne s'occupant de son implémentation a donc dû comprendre comment fonctionnaient les attributs statiques et les classes statiques.

## Jeux d'essais



## Démarche réflexive

Le concept de cette étape était de faire un bouton pour passer au joueur suivant et d'implémenter le nombre de tour. Des comparatifs ont été réalisés pour voir quelle méthode étais la plus efficace. Au final, nous avons choisis de placer le bouton en dessous du rack car après l'avoir fait tester à nos camarades, il semblait plus pratique de le mettre à cette place.

## 5. Étape 4 : Interaction joueur & arbitre : jouer une tuile

### Description de l'étape

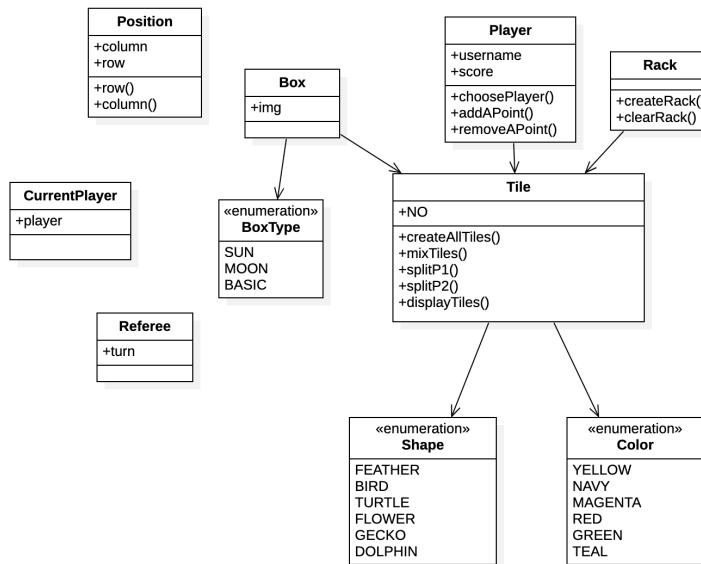
Pour cette étape nous devons mettre en place le principe de jouer une tuile. Cette méthode sera gérée via le mécanisme de Drag And Drop. Nous devons aussi calculer les points gagnés lors d'une action valide ainsi que prévoir un panneau affichant les scores de chaque joueur. Sur l'IHM le bouton est placé en dessous du rack pour une raison évidente de compréhension

Il s'agit de l'étape qui nous effrayait le plus mais aussi l'étape qui une fois réalisée nous donnerait le plus un sentiment d'avancer le projet. Pour les futures étapes il s'agit de la phase la plus primordiale, en effet, c'est en menant à bien cette partie et en pensant aux futures éventualités que nous nous faciliterons les futures étapes.

Cette étape nous a pris environ 10 heures.

## Conception

### DIAGRAMME DE CLASSE



### PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION

Pour cette étape, la plus grosse difficulté a été de conceptualiser le système de déplacement des tuiles. En effet, l'un d'entre nous a commencé dès l'étape 3 à réaliser la conception du DragAndDrop, mais nous ne savions pas comment le réaliser.

En effet, le principe de Drag And Drop était encore très nouveau pour nous et nous n'avions pas encore appris à l'utiliser pour déplacer des classes non primitives (que l'on a créé). Nous avons alors vite compris qu'il existait de nombreuses manières de conceptualisé cette fonctionnalité et nous ne savions pas laquelle choisir.

Un autre problème a été que nous avions besoin d'essayer l'implémentation de notre conception pour vérifier qu'il fonctionne. Alors, la phase de conception nous paraissait un peu maladroite, nous avons proposé une conception qui allait sûrement changer.

## Implémentation

### COMMENT AVEZ-VOUS MENÉ LA PHASE D'IMPLÉMENTATION ?

Pour mener la phase d'implémentation de cette étape, nous avons décidé de travailler comme pour la première étape en pair-programming. En effet, l'un d'entre nous avait déjà avancé le système de Drag and Drop, le second est donc venu vérifier et relire le code.

Nous avons alors dans un premier temps respecté la conception mais cela n'a pas complètement fonctionné, nous avons eu des problèmes que nous détaillerons juste après.  
Ce qui nous a poussé, en second temps, à implémenter le système de Drag And Drop sur le fil.

Le pair-programming nous a donc permis de se répartir 2 rôles. Le premier imaginait une nouvelle conception pendant que le deuxième essayait d'implémenter. Nous voulions être certain que cette étape fonctionne bien et soit adapté pour les futures étapes.

Nous avons effectué environ 30 commits pour mener à bien l'implémentation de cette étape.

## INTERFACE HOMME MACHINE

L'interface homme-machine a joué un rôle très important pour cette étape, il s'agit de la base de cette phase. En effet, nous avons décidé d'utiliser le système de Drag And Drop pour déplacer les tuiles.

Alors, nous avons fait face à de nombreux problèmes :

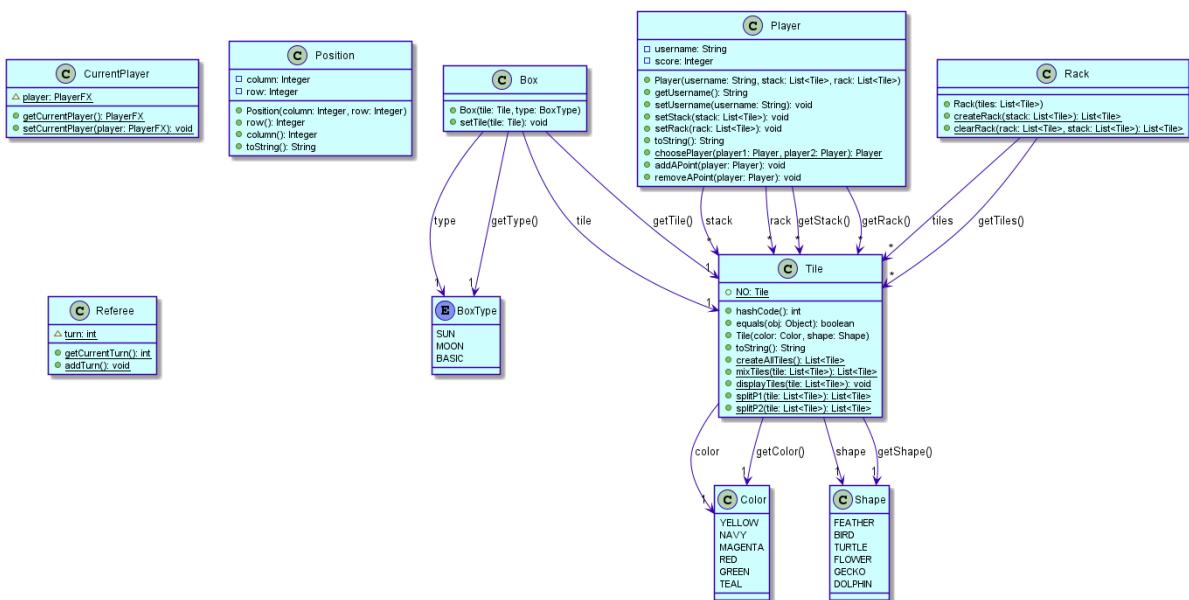
Premièrement, nous avons vu qu'il existait de nombreuses façons d'implémenter le système de Drag And Drop, nous ne savions alors pas comment le réaliser pour une classe pas primitive.

Nous avons essayé d'utiliser l'héritage "Serializable", cependant nous ne pouvions pas l'utiliser car une classe de peut pas hériter de plusieurs classes et notre classe héritait déjà d'une classe. Puis, nous ne sommes résilié à transmettre les tuiles avec des chaînes de caractères ce qui nous a permis de transmettre les spécificités (la couleur et la forme) des tuiles à travers le Drag And Drop.

L'une des autres difficultés a été de comprendre le lieu où il faut mettre les lignes de codes, en effet, étant donné que nous avons quasiment pas réalisé de Drag And Drop auparavant, nous ne savions pas si les événements étaient à réaliser dans le Drag and Drop de la source ou de la cible.

De plus, on a commencé en parallèle de cette étape à vérifier si un mouvement est valide ou non, en effet, nous avons commencé à créer des règles et à rendre le jeu praticable, l'un de nous deux a commencé l'étape 5 pendant que l'autre finissait cette étape 4.

## RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET



## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION

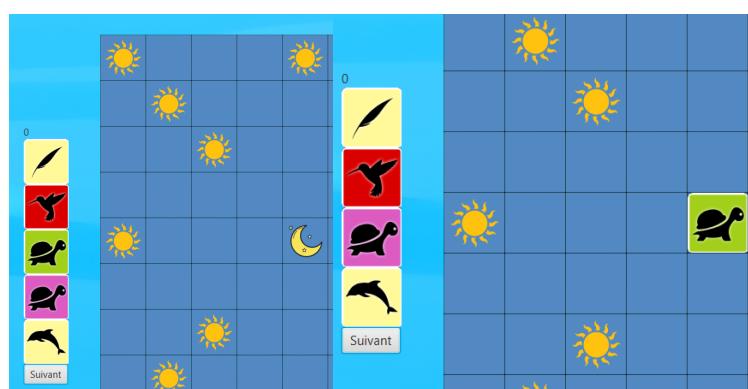
Les problèmes rencontrés durant la phase d'implémentation étaient grandement lié à l'interface homme-machine. En effet, dans l'implémentation pure et dure, des classes modèles, nous n'avons pas eu de réels problèmes.

Cependant, la phase d'implémentation de l'interface homme machine a durait, elle, longtemps et a été compliqué.

De plus, l'implémentation du système de score a était assez simple, en effet, nous avons décidé de mettre au propre l'étape 3 qui nous a permis de créer un système de tour et de joueur actuel. Alors la plupart du travail était déjà presque réalisé.

En somme, cette étape nous a pris beaucoup de temps. En effet, le fait qu'on n'ait pas d'expérience en JavaFX nous a vraiment rendu la création de l'interface homme-machine difficile comme nous l'avons expliqué précédemment

## Jeux d'essais



## Démarche réflexive

Lorsqu'on joue une tuile, elle se déplace du rack jusqu'au plateau via le mécanisme de DragAndDrop. Cette manière était une des deux méthodes présentées dans le sujet. Nous avons préférés nous orientés vers celle-ci car elle est plus visuelle que le fait de cliquer sur la case pour jouer la tuile. Nous avons trouvés que le DragAndDrop était plus « immersive » pour le jeu.

## 6. Étape 5 : Rôle de l'arbitre

### Description de l'étape

Pour cette étape nous avons du nous occuper de l'arbitre c'est à dire :

- Déterminer si un mouvement est valide ou pas
- Déterminer la fin d'un tour
- Déterminer la fin de la partie
- Proclamer les résultats

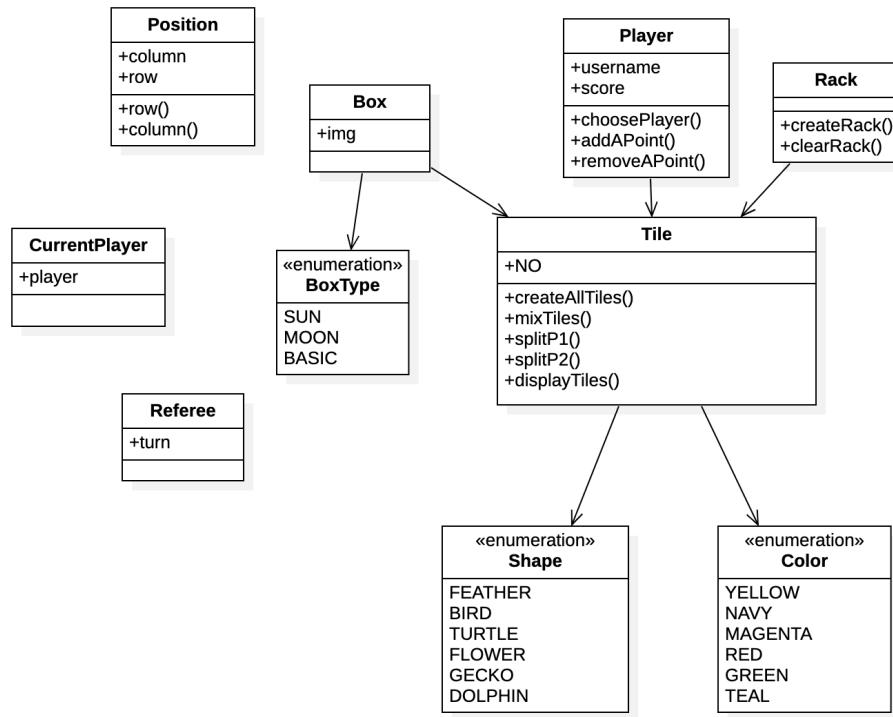
Comme pour l'étape 3, cette étape 5 est une étape arbitraire, graphiquement il n'y a pas grands changements, mais nous devons rendre le jeu jouable avec les actions de bases.

De plus, nous devons bien faire attention à prendre en compte uniquement les mouvements valides. Il s'agit d'une partie de l'arbitrage qui nous a pris beaucoup de temps.

Cette étape nous a pris environ 5 heures.

# Conception

## DIAGRAMME DE CLASSE



## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION

La phase de conception s'est déroulée assez rapidement, en effet, étant donné que durant l'étape 3 nous avions réalisé les systèmes de joueurs actifs et de tours, nous avions finalement seulement besoin de déterminer la fin de la partie et de proclamer les résultats.

Durant la conception, une des questions auxquels on a dû répondre était comment proclamer les résultats, en effet, nous ne savions pas où proclamer les résultats, étant donné que tout l'écran était déjà rempli, nous avons alors longuement échangé pour arriver à une finalité qui est une dialogue d'alerte.

# Implémentation

## COMMENT AVEZ-VOUS MENÉ LA PHASE D'IMPLÉMENTATION ?

Nous avons décidé durant la phase d'implémentation de cette étape de donner tous les rôles à un seul étudiant. En effet, étant donné que tous les objectifs étaient liés, l'un d'entre nous à réaliser les étapes dans son coin pendant que le deuxième préparait le menu du jeu, en effet il s'agit d'une partie graphique qui nous semblait importante.

Dans un second temps, nous avons mis en commun les codes pour relire et vérifier ce qui a été implémenté et pour vérifier que l'implémentation n'a rien cassé.

Nous avons aussi remarqué que dans la conception, il manquait un attribut au joueur, pour que l'on sache si il peut jouer ou non, il s'agirait d'un booléen.

Nous avons effectué environ 10 commits pour mener à bien l'implémentation de cette étape.

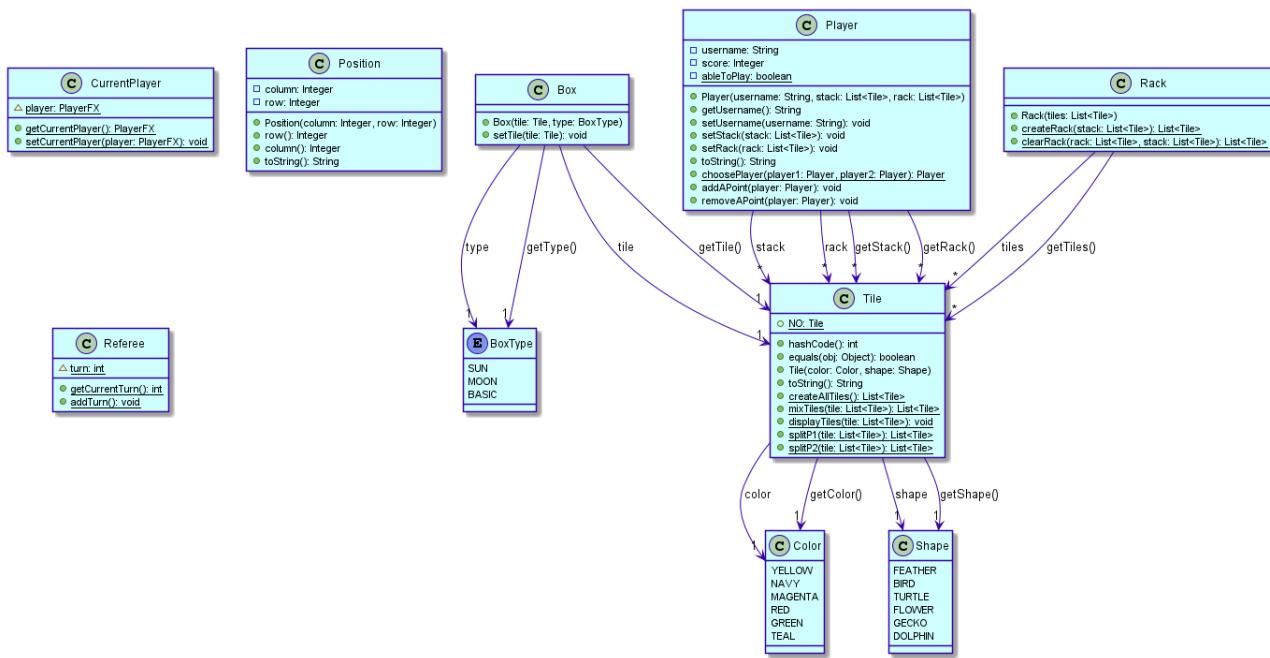
## INTERFACE HOMME MACHINE

Dans cette partie, l'interface homme-machine concerne essentiellement la vérification des mouvements. En effet, nous avons eu à vérifier la forme et la couleur des tuiles de manière à ce que mathématiquement, la tuile puisse être posée

De plus, nous avons ajouté un dialogue d'alerte

Finalement quand une tuile ne peut pas être déposée, l'interface nous prévient que nous ne pouvons pas déposer de tuile. De même quand on peut déposer une tuile, nous le voyons.

## RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET



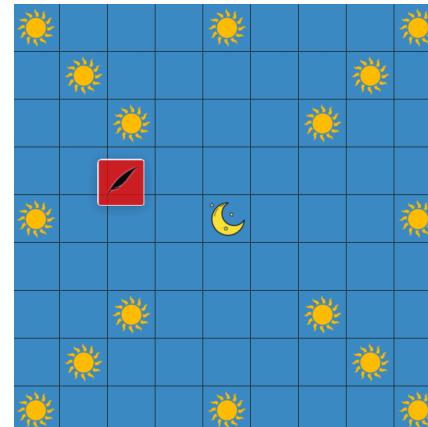
## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION

La phase d'implémentation de cette étape a été, elle aussi, assez longue, en effet nous avons dû comprendre quelles conditions étaient importantes et où est-ce qu'il fallait les mettre dans le système de Drag And Drop.

De plus, étant donné que nous transportons des chaînes de caractères, nous avons rencontré des difficultés, non pas à créer et placer les tuiles ( que nous venons de déposer ), mais plus à les enlever du rack du joueur, qui les a déposé.

En effet, nous n'arrivions pas à enlever les tuiles du rack du joueur, mais au final c'est juste que nous ne sélectionnions pas le bon joueur (l'autre joueur n'avait du coup pas la tuile).

## Jeux d'essais



## Démarche réflexive

En fonction des systèmes d'exploitation, la réaction au drag and drop varie c'est-à-dire :

- Sur windows le curseur change si on peut poser une tuile ou pas
  - Lorsque la tuile peut se poser, le curseur reste normal
  - Lorsqu'on ne peut pas poser la tuile, le curseur ressemble à un panneau de stationnement interdit
- Sur macOS, le curseur disparaît et on peut tout simplement déplacer la tuile
  - Si la tuile peut être posée sur la case, elle se pose
  - Si la tuile ne peut pas être posée sur la case, elle revient automatiquement dans le rack

## 7. Étape 6 : Interaction joueur & arbitre : acheter une action supplémentaire

### Description de l'étape

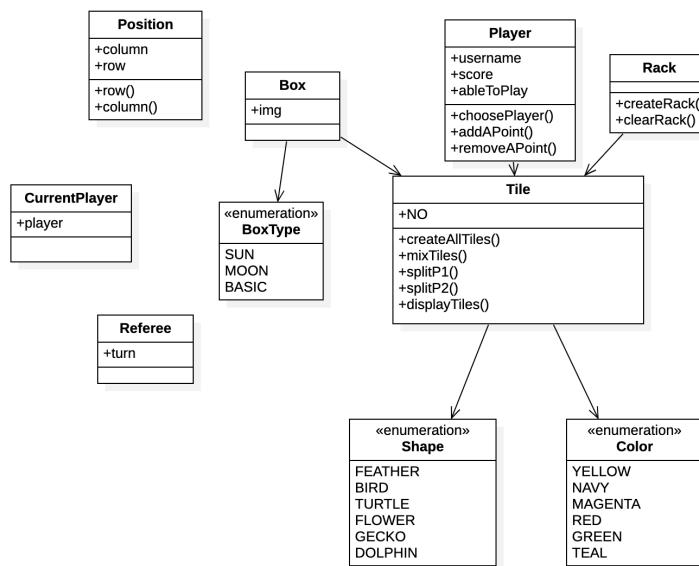
Pour cette étape nous avons implémenté le fait de dépenser des points pour acheter une action supplémentaire.

Il s'agit de l'une des dernières fonctionnalités à implémenter, mais aussi une des plus intéressantes pour jouer. En effet, c'est grâce à cette étape que les points que les joueurs gagnent tout au long de la partie trouvent leur utilité.

Cette étape nous a pris environ 2 heures.

### Conception

#### DIAGRAMME DE CLASSE



## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION

Aucun réel problème n'a été rencontré durant cette phase de conception.

En effet, il suffisait de redonner la possibilité au joueur de rejouer dès lors qu'il achetait une action supplémentaire.

De ce fait, grâce à l'implémentation judicieuse des précédentes étapes, cette phase se résumait à la création d'un bouton qui change un booléen, le joueur peut à nouveau jouer.

## Implémentation

### COMMENT AVEZ-VOUS MENÉ LA PHASE D'IMPLÉMENTATION ?

Nous avons décidé, compte tenu du temps qu'il nous restait, pour les étapes 6 et 7 de se partager les rôles de façon à ce que l'un finisse l'implémentation de cette étape et l'autre commence la conception de la prochaine étape.

En effet, comme pour certaines petites étapes du projet, un seul membre du binôme à réalisé cette étape seul. Puis l'autre a relu et vérifié celui-ci, à deux personnes, ça aurait été sûrement encore plus compliqué, il y aurait eu trop peu de tâches à se répartir.

En somme, même si durant la phase de conception cette étape nous paraissait simple, nous avons eu quelques petits imprévus. En effet, il fallait faire des vérifications :

- le joueur qui veuille acheter une action supplémentaire ait les points nécessaires.
- le joueur ait déjà joué son action.

Finalement nous avons mis un peu plus de temps que prévu pour réaliser cette étape, nous avons effectué environ 3 commits pour mener à bien l'implémentation de cette partie.

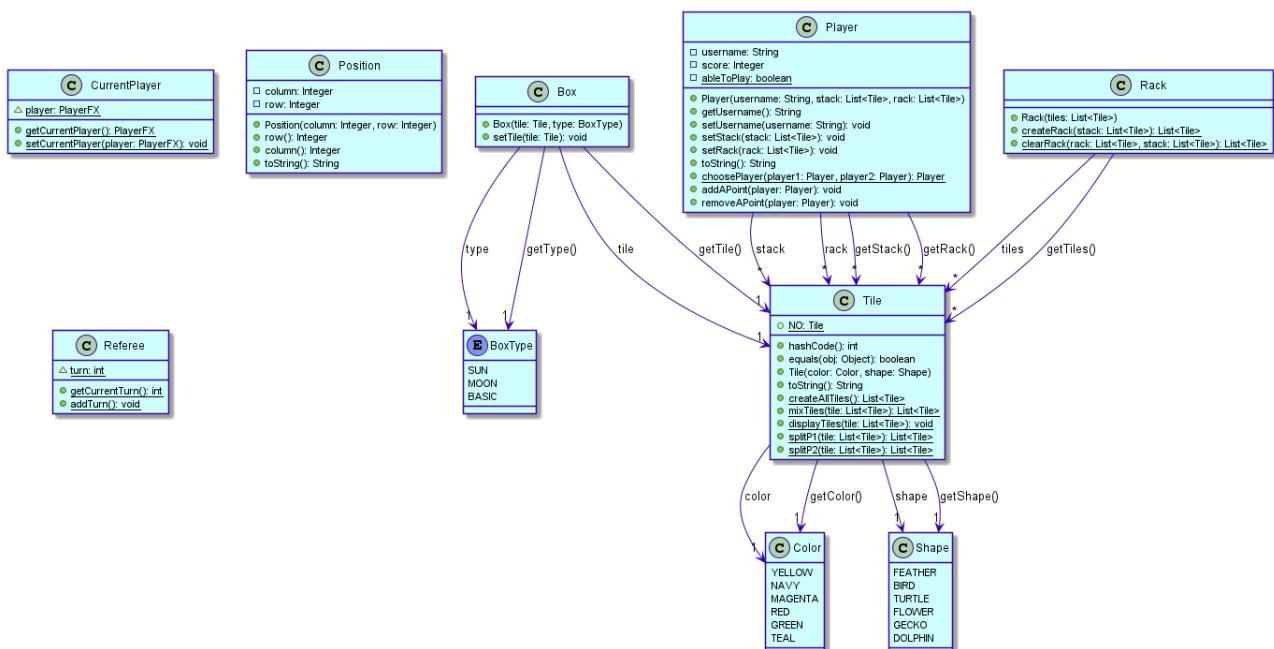
## INTERFACE HOMME MACHINE

Pour l'interface homme-machine, une des étapes les plus lourdes a été d'activer et de désactiver les boutons permettant d'acheter une action supplémentaire. En effet, nous avons décidé de rendre ces boutons inactifs lorsque le joueur ne peut pas les utiliser.

Il nous a fallu alors créer une méthode permettant d'activer le bouton seulement lorsque le joueur a assez de points.

Et finalement, nous devions utiliser cette méthode quand il le fallait.

## RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET

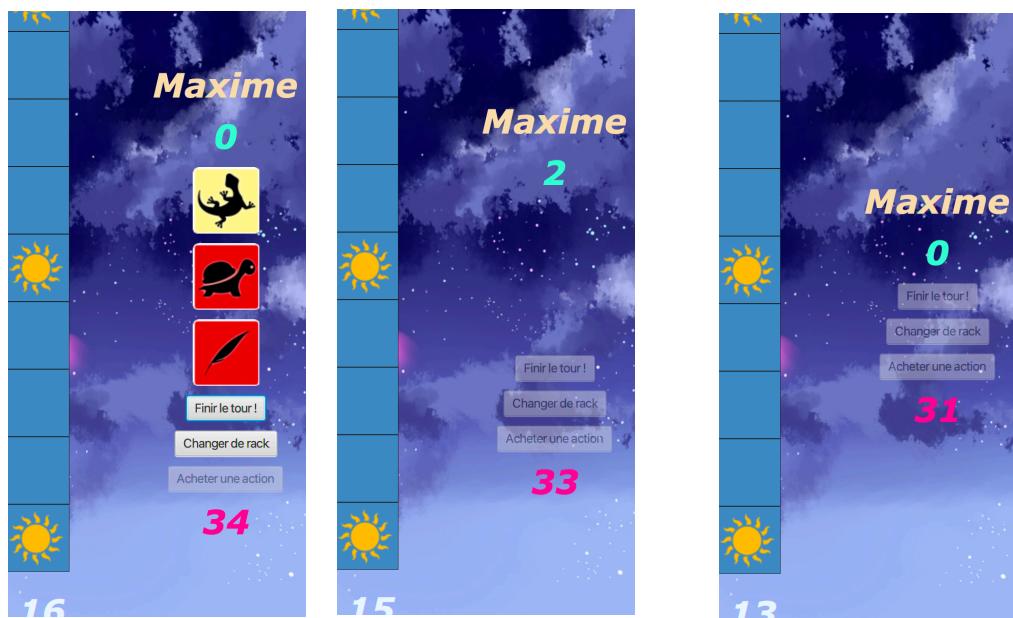


## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION

Comme dit précédemment, la plupart des problèmes rencontrés pendant l'implémentation ont été des petits problèmes d'interface graphique, il faut gérer les boutons et les activer seulement quand le joueur peut l'utiliser. Hormis, ce problème, il n'y a pas eu d'autres difficultés, les méthodes précédentes étaient suffisamment bien pensées pour satisfaire cette étape.

Pour que le joueur puisse jouer, nous avons alors utiliser l'attribut booléen AbleToPlay que nous avons introduit à l'étape précédente.

## Jeux d'essais



## Démarche réflexive

Sur la première capture d'écran on peut voir que nous avons 0 points, le tour d'après nous récupérons 2 points en se plaçant sur une case soleil ce qui nous permet de pouvoir jouer deux tuiles dans le tour. On peut le remarquer avec le nombre de pile dans la pioche qui passe de 34 à 33 puis de 33 à 31. Lorsque nous avons les points nécessaires, le bouton « acheter une action » devient cliquable et donc nous permet de rejouer dans le même tour.

## 8. Étape 7 : Interaction joueur & arbitre : échanger toutes les tuiles

### Description de l'étape

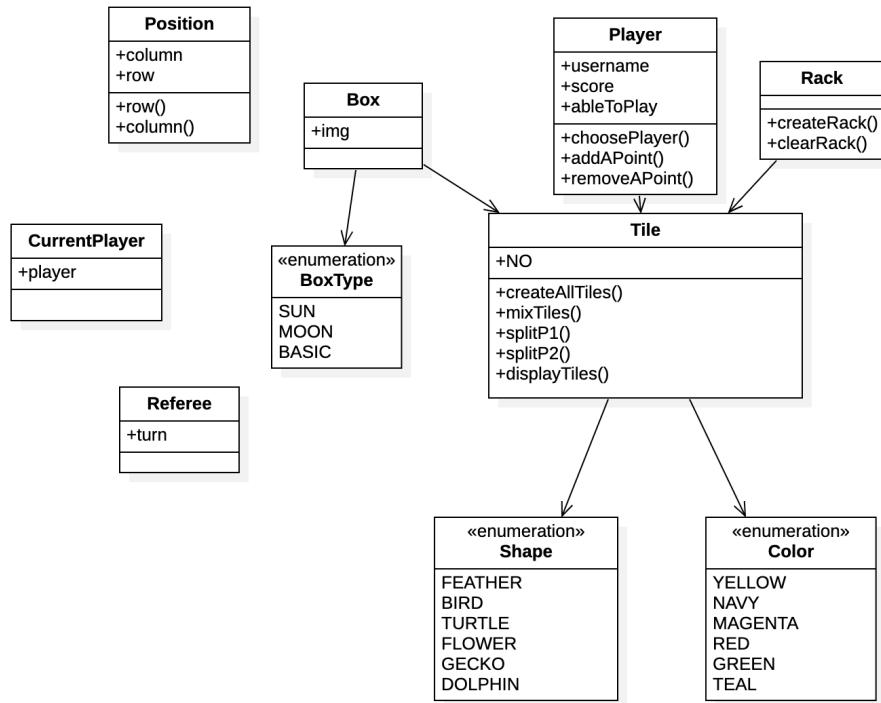
Pour cette étape nous avons implémenté le fait de dépenser des points pour échanger toutes les tuiles de son rack.

Cette étape est la dernière étape du développement mais il s'agit d'une des bases du plan de jeu car en effet, changer de rack est un des mouvements les plus utilisés dans une partie de Latice. De plus, après cette étape, le jeu de Latice doit normalement être jouable avec toutes les interactions.

Cette étape nous a pris environ 2 heures.

## Conception

### DIAGRAMME DE CLASSE



### PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE DE CONCEPTION

Durant la phase de conception, un point important nous a marqué pour cette étape. Même si normalement, tout semble bon pour changer les racks. Nous avons beaucoup de doute sur comment nous allons réussir à changer la partie graphique des racks.

En effet, depuis la 2ème étape, nous avons déjà une méthode qui permet d'obtenir un rack mélangé depuis un tas de tuiles. Mais c'est bien la partie JavaFX qui nous inquiète.

## Implémentation

### COMMENT AVEZ-VOUS MENÉ LA PHASE D'IMPLÉMENTATION ?

Comme dit dans l'étape précédente, nous avions décidé entre binome pour ces deux étapes de chacun faire une partie. C'est pourquoi dans le déroulement de cette étape, seulement l'un d'entre nous deux a réalisé l'implémentation tout seul.

En effet, même si la conception graphique nous effrayait un peu, nous avons décidé à cause du manque de temps de seulement relire et vérifier l'implémentation à deux, le reste s'est fait en autonomie.

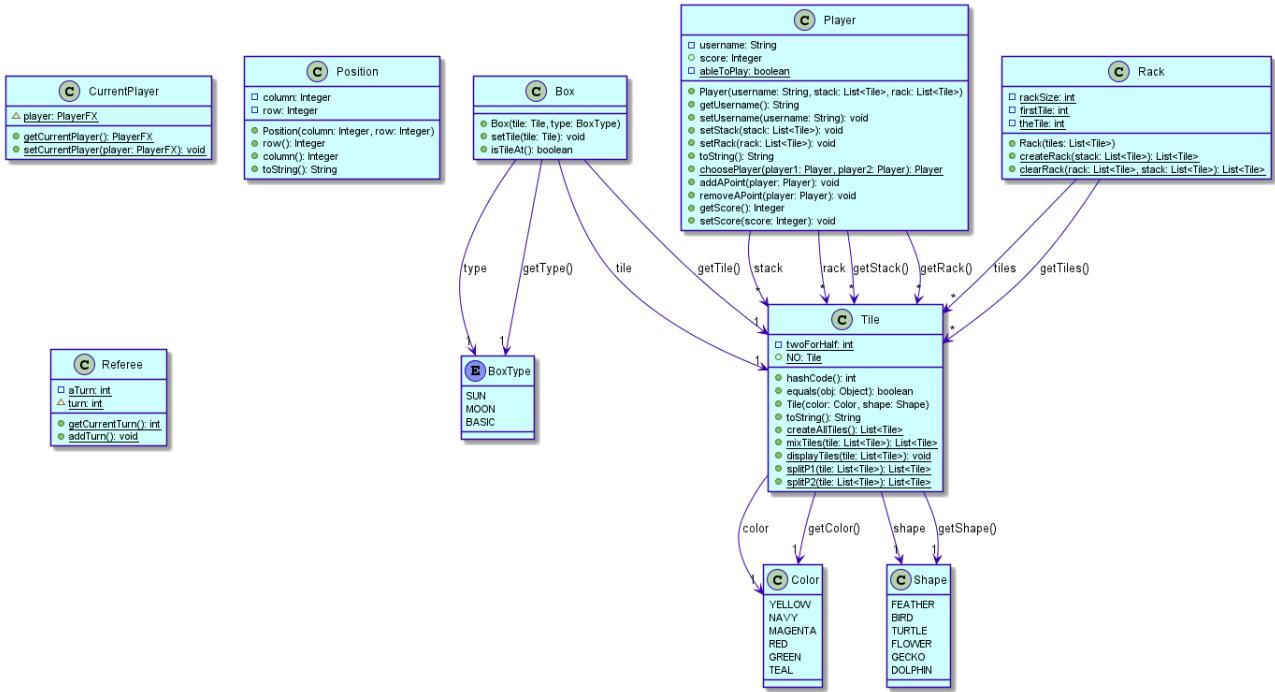
Nous avons effectué environs 2 commits pour mener à bien l'implémentation de cette étape.

### INTERFACE HOMME MACHINE

L'interface homme-machine dans cette étape consiste en un bouton "Changer de rack". En effet, nous avons implémenté un bouton qui permet de recréer la partie graphique des racks. Comme dit précédemment, ce qui a réellement été compliqué c'était de mettre à jour les VBox, en soi c'est simple, mais étant donné que durant la conception nous ne savions pas comment faire, nous avons dépensé beaucoup de temps pour rien.

En effet, au final, il suffisait de mettre un attribut VBox atteignable dans tout le GameManager, c'est ce que l'on a compris après avoir essayé de nombreuses fonctionnalités inutilisables.

## RÉTRO-CONCEPTION : DIAGRAMME DE CLASSE ACTUEL DU PROJET



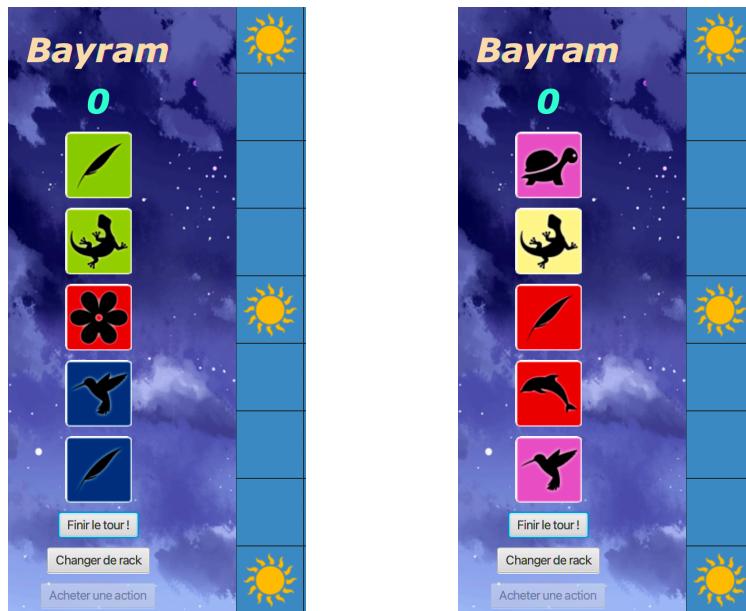
## PROBLÈMES ET DIFFICULTÉS RENCONTRÉES AU COURS DE LA PHASE D'IMPLÉMENTATION

Comme dit dans la partie sur l'interface homme-machine, en soi, l'implémentation de cette partie n'était pas compliquée, nous avions déjà toutes les méthodes et attributs nécessaires à sa conception.

Cependant, pour la partie graphique, nous avons perdu bien plus de temps, parfois pour des erreurs bêtes. Dans la plupart des cas, car nous n'arrivions pas à conceptualisé le schéma graphique en amont.

Finalement, nous avons vite réussi à nous adapter pour répondre aux exigences de l'étape, en effet, même si l'implémentation a été réalisée seule au départ, au final nous avons travaillé ensemble pour comprendre et répondre au problème et il est certain qu'on peut avancer très vite en pair-programming.

## Jeux d'essais



## Démarche réflexive

Cette étape étant la dernière de notre jeu elle aurait du être faite à la fin mais nous avons pensés qu'il serait plus facile de la faire en même temps que l'étape 3. Ce pari a effectivement payant car pour faire l'évenement du changement de rack, il faut d'abord veiller à ce que le rack soit vide avec la méthode `removeRack()` pour ensuite le re-remplir avec la deuxième méthode `addRack()`. Ces deux méthode couplées au bouton permette de changer de rack sur l'IHM

## 10. A propos de la qualité ...

Choix des structures de données, notamment le plateau de jeu

Nous avons commencé à en parler durant la première étape, mais nous avons mis énormément de temps au début pour choisir notre structure pour le plateau de jeu.

En effet, nous hésitions beaucoup car plusieurs types de structures nous paraissaient fonctionnels et nous voulions choisir celle qui correspondait le mieux à notre cas, celle qui nous faciliterai le plus pour l'implémentation.

C'est pourquoi nous avons comparé les deux structures qui nous paraissaient les plus pratiques pour construire un tableau : Les **HashMaps** et les **liste de liste**.

Premièrement la **HashMap** était encore une structure toute neuve pour nous, en effet, nous avions vu ce type de structure durant un TP et ce assez rapidement. Cependant nous avons vite remarqué que la plupart des méthodes étaient déjà implémentées et que c'était très pratique et assez simple d'utilisation. De plus, la structure fonctionnait bien pour créer un tableau.

Dans le cas des listes de liste, nous en avons fait pendant les cours de Python, il ne s'agit donc pas d'un système qui nous est complètement inconnue. Cependant, c'est moins pratique, car nous avons besoin de créer des méthodes pour remplir, vider, utiliser les différentes cases du tableau.

Notre choix s'est alors porté sur les HashMap car en effet c'est plus pratique et c'est assez instinctif. De plus dans notre cas, c'est une HashMap qui représentait mieux la situation.

## Qualité de code :

### Terminologie métier :

La terminologie métier est très importante dans un projet informatique, car en effet, dans la langue française, nombreux sont les mots qui possèdent plusieurs définitions ou même qui n'ont pas la même signification en fonction des personnes.

C'est pourquoi, il est important de poser des définitions sur ce que représentent certains mots. Dans cet optique, tout le monde a la même définition d'un mot et il n'y a pas de quiproquo.

Dans notre cas, nous avons passé beaucoup de temps en début de projet pour se mettre d'accord sur les termes que nous allions choisir et ce en anglais :

- Stack : Toutes les tuiles d'un joueur.
- Rack : Les 5 tuiles que possède le joueur dans la main.
- Tour : Une personne a joué
- Cycle : Deux personnes ont joué

### Mauvaise odeur (code smell)

Durant ce projet, nous avons fait attention à différents code smell. Parmi ceux-là, nous avons fortement fait attention à ce qu'il n'y ait pas de "magic number" donc de nombres qui apparaissent sans aucun sens, car en plus de rendre le code brouillon, ça peut fortement empêcher sa compréhension.

De plus, nous avons évité le plus possible de répéter des parties de code. C'est pourquoi nous utilisons des méthodes : moins on se répète, moins il y aura besoin de passer de temps sur un problème en effet. Plus il y a de répétition, plus un changement peut prendre du temps (corriger plusieurs fois)...

Finalement, nous avons tout fait pour donner noms de variables précises et adéquates. En effet, plus une variable possède un nom précis, plus son utilité est rapide à comprendre. Et c'est vraiment ce qui permet de comprendre le code, un code avec des variables mal nommées est vraiment dur à prendre en main. Entre "compteurDeTuiles" et "x", il y en a un que l'on comprend mieux.

## Revue de code :

Étant donné que nous ne sommes pas (encore) de réels développeurs, mais que dans notre classe se trouvent des personnes avec plus d'expériences, nous avions décidé de faire relire notre code par nos camarades, il ne s'agit peut-être pas d'une relecture complètement sûre, mais parfois, on apprend des choses que l'on ne savait pas ou des erreurs d'inadvertisances (comme par exemple, inverser les lignes et les colonnes...).

De plus, comme nous l'avons dit précédemment, quand l'un d'entre nous écrivait du code seul, le second relisait le code dans le but de le vérifier mais surtout de le comprendre. On était donc sûr de ne pas être perdus.

## Détection et correction de bug :

Quand, l'un d'entre nous deux faisait face à un bug, il essayait dans un premier temps de corriger le problème avec des essais, car dans quasiment tous les cas, il s'agissait d'un petit problème qui peut être résolu avec une relecture.

Si après même une relecture il n'arrivait pas à comprendre le problème, il demandait à l'autre du binôme de venir pour essayer de comprendre ce qui n'allait pas. Ca ne nous est pas arrivé souvent, car le problème de gêner l'autre personne, c'est qu'elle va devoir arrêter ce qu'elle fait pour se plonger dans du nouveau code.

En soi, le fait de prendre du recul aussi pouvait permettre de résoudre quelques problèmes, en effet après avoir pris du recul c'est souvent plus simple de comprendre le problème.

Effectivement, car souvent il s'agit d'un petit problème qui est compliqué à attraper sur le fil.

De plus, nous avons souvent découpé le code quand il y avait un bug. En effet, plus on est précis sur la zone du problème, moins on peut s'éparpiller et plus c'est simple de le trouver.

Pour la correction de bug en elle-même, ce qui marchait vraiment dans notre cas, c'était de prendre une feuille de papier et de décrire le bug en détail, à l'écrit, nous faisions alors des schémas ou nous prenions des notes sur ce qui devait se passer comme il se passe et ce qui devait être différent.

Le bug qui nous a fait perdre le plus de temps, était vraiment un bug très bête, en effet pendant la partie où il fallait vérifier le Drag And Drop et mettre des conditions pour le placement d'une tuile, nous avons eu un problème où nous pouvions poser les tuiles mais pas au bon endroit.

C'est grâce à un schéma que nous avons vu qu'en réalité nous pouvions poser les tuiles mais que de façon symétrique. Nous avons alors compris que le plateau était juste à l'envers, nous avions mélangé les colonnes et les lignes.

## Qualité du développement

### Présentation de la stack technique :

Pour vérifier la qualité du développement, nous avons utilisé une extension nommée SonarLint, celle-ci nous permettait de vérifier le nommage des variables ou si une méthode est bien utilisée. Nous avons alors pris en compte les conseils de SonarLint et avons vérifié avec nos connaissances pour savoir si c'était vraiment important.

### Gestionnaire de version :

Au début du projet nous utilisions mal git. En effet, nous développions sur les mêmes fichiers et ce en même temps ce qui signifie que lorsqu'on voulait push notre avancement on se retrouvait tout le temps avec des conflits. Nous avons donc eu la brillante idée d'utiliser afin de palier à ce problème. Résultat, nous développions sur nos branches respectives. La solution optimale aurait été de créer une branche pour chaque fonctionnalité de Latice et non une branche par personne sur le projet. Nous faisions des merges entre les branches pour ajouter les différentes fonctionnalités de notre jeu. Les commits étaient fait une fois la fonctionnalité opérationnelle et les push se faisaient à la fin des séances de saé

## Architecture de notre application

L'architecture de notre application est découpée en plusieurs package, 6 pour être précis.

Le premier package s'appelle "application", il contient :

- LatticeApplicationConsole qui nous permet de voir les affichages en console des méthodes dont on a besoin
- LatticeGameManager qui est la fenêtre principale de notre jeu, celle ci affiche le jeu et nous permet de jouer à Lattice.
- Menu qui est le menu de notre jeu avec un bouton jouer, un bouton règles et un bouton quitter

Le second package est "controller" celui ci contient :

- DndChecking, il permet de vérifier si notre action est valide ou non
- DndImageViewController, c'est le fichier principal du DragAndDrop
- DndScoreController, ce fichier permet de gérer les scores des joueurs. Il est incrémenté si une tuile est posée sur une case soleil ou si il y a 2,3 ou 4 tuiles autour de la case jouée.

Le troisième package est un des plus importants il se prénomme "model", il est constitué :

- des fichiers Box, BoxType, Color, CurrentPlayer, Player, Position, Rack, Referee, Shape et Tile
- il est aussi composé de deux autres packages

- constant qui, comme son nom l'indique est composé d'un fichier de constantes. Ce fichier est très utile pour ne pas avoir de magic number.

- visual. Ce package regroupe deux fichiers : GameBoard qui contient la génération du plateau en Java ainsi que GameBoardFX qui contient la même chose mais pour JavaFX.

Le dernier package s'appelle "visual", son nom explicite se suffit à lui-même :

Il contient la version JavaFX des classes suivantes :

- Box
- Player
- Rack
- Tile

Ces classes héritent des classes normales, elles rajoutent les images afin qu'elles s'affichent sur notre application.

## A propos des tests automatisés

Au commencement de la saé, nous nous sommes dit qu'il serait préférable de faire les tests à la fin une fois le code complètement fonctionnel. Cette réflexion c'est avérée être une grave erreur car une fois bien avancé dans le développement, on s'est rendu compte que les tests étaient une partie importante de toute application. Les tests d'une application sont une phase très importante dans les cycles de développement et de maintenance d'une application. Ils permettent de détecter des bugs et de s'assurer que l'application réponde au cahier des charges et aux spécifications. Nous avons plus perdu du temps à faire des tests à la fin plutôt que de les faire pendant les différentes phases d'implémentations.

## Petit bilan

Si on applique les principes de la qualité de code, le développement semble plus facile que ça soit pour s'y retrouver dans le projet ou encore pour un travail de maintenance dans le futur. Cependant nous nous sommes penchés sur ces aspects trop tard ce qui nous a ralenti. On aurait du penser dès le début pour gagner en productivité.

## Démarche réflexive sur la qualité accordée à votre développement

Comme dit précédemment la qualité de développement est un atout majeur lors de l'implémentation d'un projet si elle est correctement utilisée et mise en place. Que ça soit pour l'implémentation ou l'élaboration des conceptions simples, les essais ou la gestion et l'analyse des problèmes. La qualité de code permet de mieux montrer au client les implémentations car le code est propre grâce au refactoring. Enfin certaines composantes essentielles ne sont pas développées comme la CE4 de la compétence 1 (Respecter la législation, les normes).

professionnelles et les enjeux sociétaux) car ici nous n'avons pas eu le temps de nous pencher sur la question éthique de refaire un jeu Latice ou la CE4 de la compétence 2 (Formaliser et mettre en oeuvre des outils mathématiques pour l'informatique).

## Conclusion

En somme, nous avons **beaucoup appris** pendant ces deux SAE, c'est nous pensons ce qui nous a le plus marqué au bout de ces 6 semaines de travail.

En effet, nous sommes partis de très petites bases, aucun de nous deux n'avait jamais auparavant codé en Java ou en n'importe quel langage orienté objet. De plus, nous n'avions pas un niveau excellent en programmation Python ou C, nous n'étions donc pas dans un espace de travail très habituel.

Nous avons, alors, dû faire face à l'énoncé et nous avons du coup appris beaucoup sur le coup, sur le fil.

Nous avons compris les différents principes de la programmation, nous avons réalisé notre premier réel Drag And Drop, nous avons fait attention à la qualité de notre code. Nous avons réalisé tout plein d'étapes que nous avons détaillé, il s'agissait sûrement du meilleur moyen pour apprendre à coder une application.

Nous avons obtenu un résultat carrément surprenant.

Nous savons désormais, réaliser un tableau de jeu, nous savons utiliser le Drag And Drop et surtout nous avons remarqué que nous hésitions beaucoup moins vers la fin qu'au début. De plus, nous avons appris à versionner avec git, ou à faire des tests avec JUnit.

Finalement, ce projet nous a permis de faire un premier pas dans le développement orienté objet et d'acquérir les bases de ce type de langage.