isep
École d'ingénieurs du numérique

Rapport du projet Java Domi'Nations

[II-1102] Algorithmie et programmation – Semestre 1

Source : https://www.topsimages.com/images/animated-kingdom-0c.html

Mathieu VALENTIN, Sébastien VIGUIER, Laurent YU
13/01/2019

Table des matières

I - Présentation du projet Domi'Nations

Dans le cadre du projet de java, nous devions réaliser un jeu de plateau qui s'appelle Domi'Nations. C'est un jeu multi-joueur inspiré du jeu Kingdomino/Queendomino, pouvant se jouer de deux à quatre personnes. Chaque joueur se voit attribuer un roi avec lequel il devra construire le royaume le plus valorisant afin de battre les autres joueurs. Pour cela, il doit construire son royaume autour de son château.

Les royaumes sont constitués de dominos à deux cases. Sur l'une des faces est inscrit le numéro du domino et sur l'autre les deux terrains du domino ainsi que le nombre de couronne par terrain. Il existe plusieurs terrains différents : les champs, les forêts, les mines, les mers, les prairies et les montagnes. Pour faire augmenter la valeur de son royaume, chaque joueur devra prendre en compte plusieurs paramètres.

Il y a d'abord la pose du domino. Elle peut se faire horizontalement ou verticalement. Il en est de même pour les connexions entre les terrains. Pour poser un domino sur son royaume, le domino à poser doit avoir au moins un de ses terrains en contact avec un terrain du même type appartenant au royaume ou avec son château. Dans le cas où le joueur ne pourra pas poser le domino sur son plateau, il devra alors le défausser.

Le vainqueur est déterminé grâce à un système de points. Le score dépend du royaume, de la taille de ce dernier et du nombre de couronnes présentes. Le score du royaume correspond à la somme des produits entre le nombre de cases adjacentes et le nombre de couronnes présent dans la zone. Le vainqueur est celui qui a obtenu le plus de points.

L'objectif de ce projet est de réaliser un jeu fonctionnel en utilisant la console proposée par notre IDE. Nous devons développer les fonctionnalités de base comme, initialisation du jeu, le déroulement d'un tour du jeu, le calcul de point et l'identification du/des vainqueurs.

Dans un premier temps, nous allons vous décrire l'architecture de notre projet avec un diagramme de classe. Dans un second temps, nous allons décrire le répertoire du projet puis l'implémentation technique. Enfin nous préciserons certaines fonctionnalités et les bibliothèques utilisées dans notre projet.

II - Structure en MVC

Concernant la structure du jeu, nous avons choisi l'architecture MVC qui est très couramment utilisé pour les sites web. Nous avons décidé de le mettre en place très rapidement. L'objectif de ce pattern est de diviser le code en 3 parties différentes :

- M représente le modèle. On peut retrouver dans cette partie, du code permettant d'avoir une liaison avec une base de données. Le modèle permet de gérer toutes les données du site.
- V représente la vue. Dans notre cas, nous avons utilisé des fichiers FXML générés par le logiciel SceneBuilder, et la librairie JFoenix.
- C représente le contrôleur. Cette partie permet de faire le lien entre les données provenant du modèle et de la vue qui affiche ces informations.

Cette architecture a des bons et des mauvais points. Nous pouvons noter que l'architecture MVC permet de regrouper des parties de codes similaires. Par exemple, le code décrivant les règles du jeu se trouvent dans la partie modèle. Cette segmentation du code permet aussi de développer des fonctionnalités simultanément. De plus, elle permet aussi de retrouver des bugs très rapidement, car on sait d'où peut provenir les bugs. Mais, l'architecture MVC a des points faibles. Naviguer entre les 3 parties n'est une chose aisée. De plus, il faut aussi avoir de solides bases sur les différents langages de programmation (Java et XML).

III - Description des classes du jeu

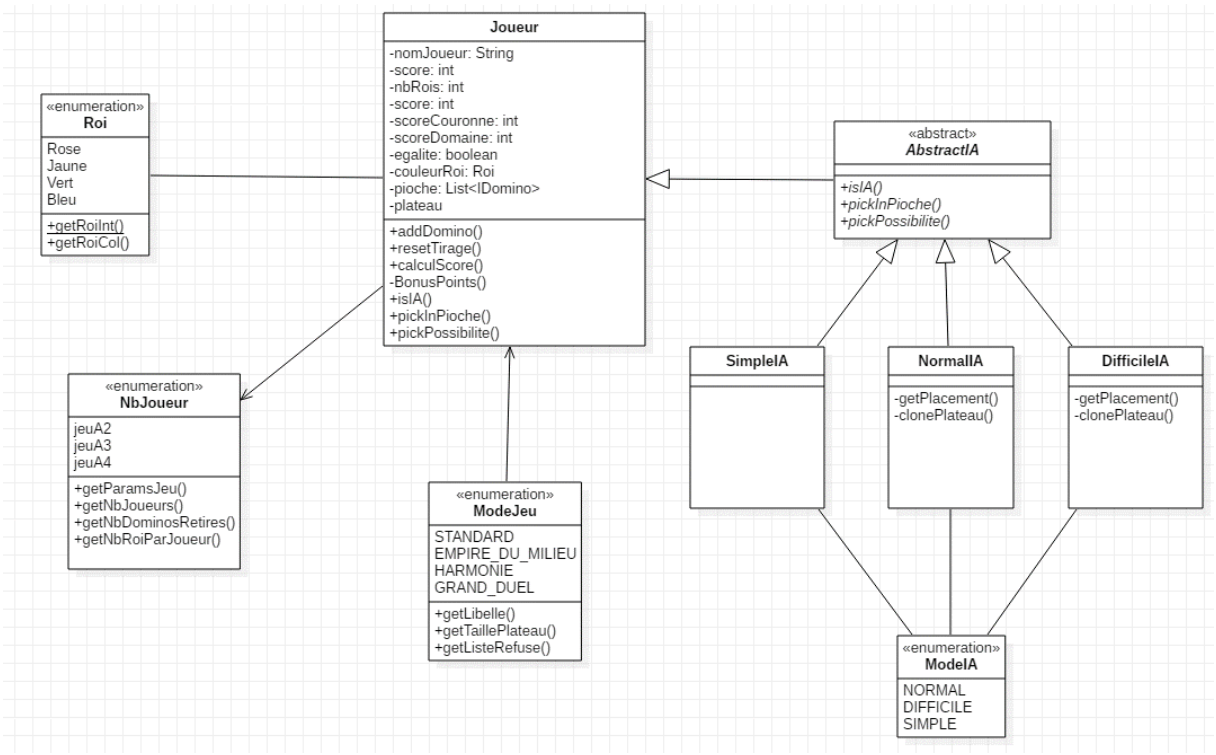
Le diagramme de classe étant trop grand et trop complexe, nous ne pouvons le mettre dans ce rapport car il serait alors illisible. A la place, nous préférons montrer le diagramme partie par partie afin de permettre une meilleure compréhension de l'architecture mise en place. Il sera néanmoins en dernière partie afin d'avoir une vue d'ensemble sur la structure globale de l'application.

A. Le package Jeu

Le package jeu est l'un des packages de la partie modèle de l'architecture MVC, mise en place pour ce projet. Il est composé de deux parties : la gestion du jeu et la gestion des joueurs.

1. La gestion des joueurs

Le système de gestion des joueurs se présente sous cette forme :



La classe Joueur est au centre de toutes les dépendances dans cette partie. Cette classe contient toutes les données personnelles du joueur : ses scores, son plateau de jeu, son ou ses rois et les dominos qu'il pioche. La gestion des données se fait par plusieurs méthodes permettant de calculer ses scores, d'ajouter un domino à son plateau, de piocher un domino dans la pioche et de choisir un placement possible pour le domino sur son royaume.

La classe Joueur utilise trois classes d'énumérations qui sont essentielles au fonctionnement du jeu. La première est Roi. Elle permet au joueur de stocker la couleur de son ou de ses rois et aussi de connaître l'hexadécimal de la couleur attribuée. La seconde est NbJoueur. Cette classe regroupe les paramètres de jeu en fonction du nombre de joueurs participant à la partie de jeu. Les paramètres de jeu dépendent du nombre de joueurs. A partir de ce dernier, le nombre de domino à retirer du jeu et le nombre de rois possédés par chaque joueur sont renseignés. Cette classe facilite grandement la gestion du jeu en fonction de ses paramètres et de simplifier l'architecture du projet. La dernière classe

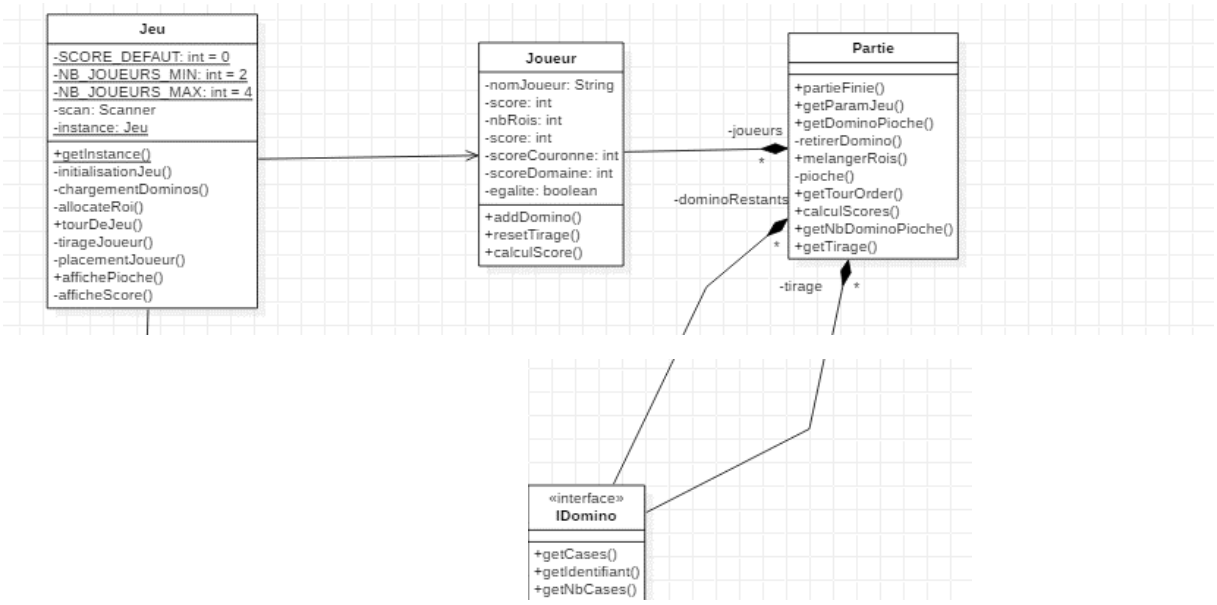
d'énumération est ModeJeu. Elle contient les valeurs de la taille du plateau mais aussi des points bonus obtenus en fonction du mode de jeu sélectionné au démarrage de la partie.

Les différents IA présents dans le jeu dépendent de la classe joueur car elles héritent de la classe abstraite AbstractIA qui elle-même hérite de la classe Joueur. Ce polymorphisme d'héritage permet d'utiliser les mêmes méthodes et données à la fois pour un joueur comme pour une IA. Trois classes héritent de AbstractIA : SimpleIA, NormalIA, DifficileIA. SimpleIA décrit une IA de base qui pioche et pose un domino au hasard sur son plateau de jeu en fonction des possibilités de placement du domino. L'IA normale quant à elle calcule le meilleur placement possible pour obtenir le meilleur score possible. Son fonctionnement sera détaillé dans la suite du rapport. DifficileIA n'est pas implémenté dans sa totalité. Elle reprend le fonctionnement de NormalIA (qui fonctionne) et l'applique à tous les autres joueurs, pour, choisir le domino et son positionnement en fonction de leur impact sur les plateaux des joueurs adverses afin qu'ils aient des possibilités de placements donnant un score plus restreint. Cet IA sera complété pour le concours IA, donc après le rendu de ce projet. Ces IA sont instanciées grâce au choix du niveau de difficulté dans le jeu via la classe d'énumération ModeIA.

Les classes d'énumérations dans le package jeu ont une importance cruciale dans la structure du jeu. En effet, elles permettent de factoriser les accès aux données et rendent le code plus générique. L'ajout ou la suppression d'un des éléments de ces classes peut se faire facilement ce qui contribue à la maintenabilité de la gestion des joueurs.

2. La gestion du jeu

Le système de gestion du jeu est composé de deux classes : les classes Jeu et Partie.



La classe Jeu a un impact plutôt global sur les composants utilisés dans le déroulement d'une partie de jeu. Elle contient le chargement des dominos à partir d'un fichier csv, l'allocation des différents aux joueurs et définie les tours de jeu. La gestion des actions de pioche et de placement des dominos y est aussi présente. Elle permet de lancer les services pour chaque joueur et gère leurs actions en fonction de leur statut, à savoir s'il s'agit d'une IA ou d'un joueur normal.

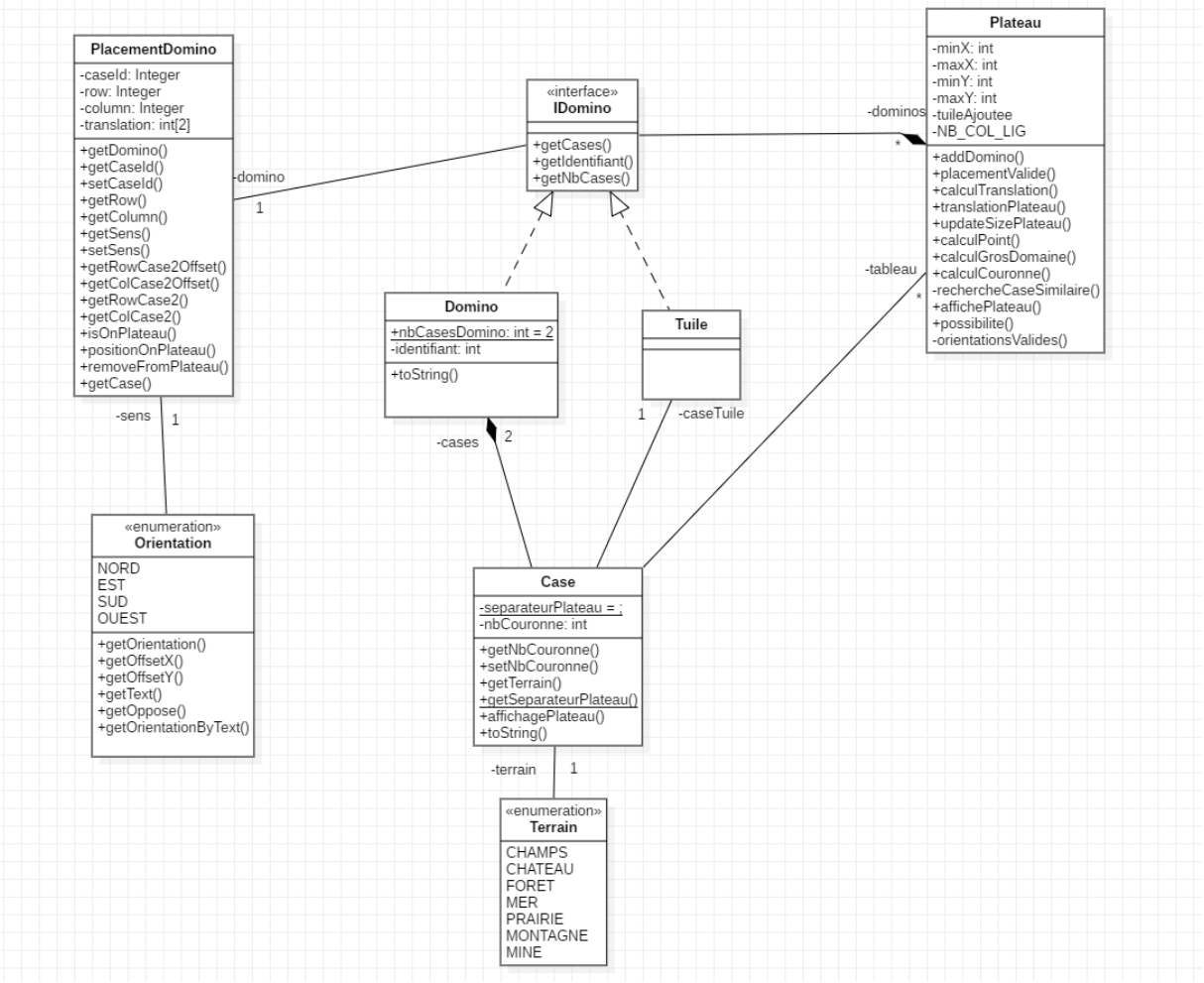
La classe Partie contient la liste de tous les joueurs, la liste des dominos restants en jeu et les dominos de la pioche. Les méthodes dans cette classe sont plus d'ordres particulières, contrairement à la classe jeu qui a une vision globale. La classe Partie contient les différentes phases internes à une partie de jeu. Cela comprend l'action de piocher les dominos parmi ceux qui restent, de mélanger les rois, de

retirer les dominos en trop en début de partie, de connaître l'ordre du tour de placement du domino et de connaître la liste des joueurs classée par ordre décroissant en fonction de leurs scores. Ces deux classes sont au cœur de la gestion du jeu dans le mode console mas aussi avec l'interface graphique afin de permettre un déroulement performant des étapes de la partie.

B. Le package plateau

Le package plateau contient l'intégralité des composants constituant un plateau de jeu et les méthodes de placement des dominos sur ce plateau.

Le diagramme de classe du package plateau s'exprime de cette forme :



Cette partie sera décomposée en trois sous-parties :

- Le traitement de l'implémentation des dominos
- La gestion des positionnements des dominos
- La gestion du plateau.

1. Implémentation des dominos

Un domino est composé de deux cases. Chaque case contient un nombre de couronne et possède un type de terrain. Il existe un type de domino particulier qui est la tuile. Elle ne contient pas de couronne et son type de terrain est un château.

Nous avons décidé de créer une interface pour que Domino et Tuile puissent l'implémenter. L'interface ne comprend que peu de méthodes car le principe de l'encapsulation est utilisé. Les propriétés cherchées peuvent être facilement retrouvées au travers de ces méthodes.

2. La gestion du positionnement des dominos

La classe PlacementDomino est une classe utilisée dans le modèle et dans la partie contrôleur. Elle s'occupe d'un seul domino. Cette classe contient toutes les informations d'un placement d'un domino dans le plateau d'un joueur. Elle contient les coordonnées en x et en y du domino, le domino à placer et le tableau de translation.

Ces méthodes consistent à avoir accès au domino, à sa position mais aussi aux *offsets*. Comme un domino est constitué de deux cases, le sens choisi pour ce dernier peut avoir un impact sur la taille du plateau. Les *offsets* en x et en y permettent donc de savoir si le royaume va devoir augmenter le nombre maximal de lignes ou de colonnes. La classe utilise une classe d'énumération qui est Orientation. Elle comprend toutes orientations possibles d'un domino mais aussi les *offsets* de chaque direction (exemple : Nord : x = -1, y = 0).

Cette classe est utilisée dans la création d'une liste de possibilités pour une IA, et dans la partie affichage pour conserver les informations d'un placement souhaitée par l'utilisateur.

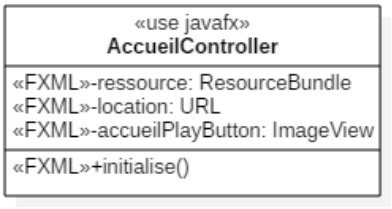
3. La gestion du plateau de jeu

Le plateau de jeu se définit en un nombre maximal de lignes et de colonne ainsi que d'un tableau de cases ainsi que la liste des dominos le composant. Le plateau possède un tableau de cases de dimension 5x5 ou 7x7. Cette dimension peut être contraignante car le positionnement du château sur le plateau est définitif. A la création d'un joueur ou d'une IA, nous plaçons le château au milieu du tableau (coordonnées : 2 ;2). Pour éviter cela, nous avons décidé de créer des fonctions permettant de translater les cases. Si le joueur place son château à la position 0, 0 et qu'il souhaite placer un domino sur les positions -1, 0 et -2, 0. Les fonctions de translation vont faire bouger les cases. Nous décrirons plus en détail cette fonctionnalité dans les parties suivantes.

C. Le package Contrôleur

Attention, nous attirons votre attention sur le fait que nous avons utilisé des *lambdas expressions* (fonction anonyme qui n'est pas définie dans le code). Les lambdas sont disponibles à partir de Java 8.

Notre projet possède la version console et la version graphique. Le package controller regroupe toutes les fonctions permettant de faire la transition entre la partie modèle et la vue. SceneBuilder permet de générer une partie du code des contrôleurs pour pouvoir interagir avec les vues en FXML. Le digramme de classe se trouvant en bas, concerne qu'une partie des contrôleurs présents dans le projet.



1. L'accueil

Cette partie concerne la première vue du jeu, où il est possible de cliquer sur le bouton jouer. La classe Java permettant cela est AccueilController.

2. Le menu

Cette partie concerne la deuxième vue du jeu où les joueurs doivent renseigner toutes les informations avant de pouvoir lancer une partie. Le contrôleur MenuController doit récupérer :

- Les différents modes de jeu présents dans l'énumération ModeJeu.
- Le nombre de joueur
- Les noms des joueurs ou IA
- La difficulté des IA

Le contrôleur va envoyer à la vue les différentes options.

3. La pioche

Le contrôleur PiocheController va faire dérouler les différentes étapes de la sélection d'un domino. Le contrôleur va récupérer depuis le modèle, les dominos qui ont été choisis depuis le deck (tous les

dominos). Par la suite, le contrôleur va envoyer des informations telles que le choix des joueurs concernant les dominos vers le modèle.

4. La partie

Le contrôleur PartieController va faire dérouler les différentes étapes du jeu (les temps d'attente entre les joueurs et les IA, la partie placement du domino sur dans le royaume, la fin du jeu).

5. Le score

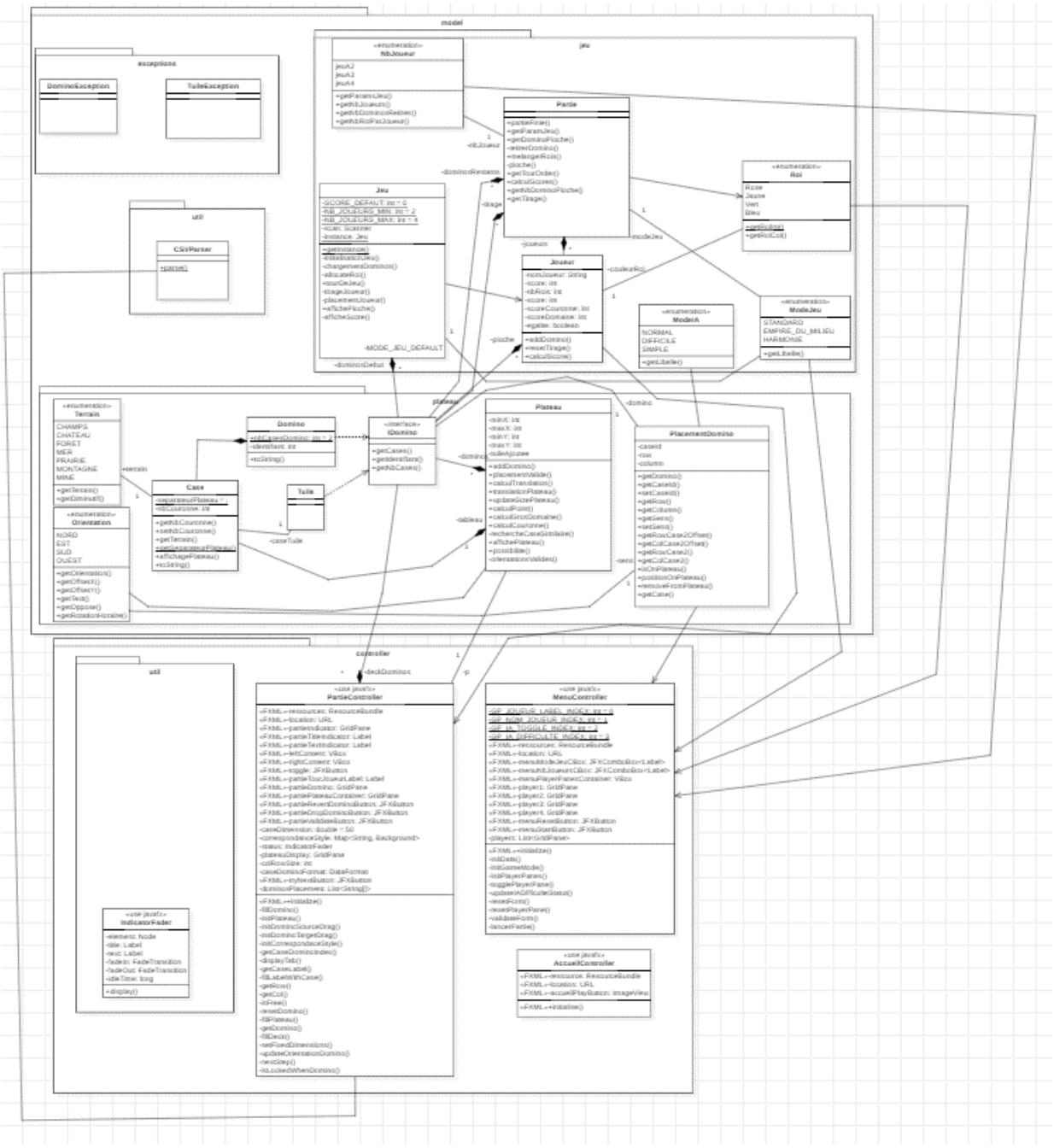
Le contrôleur ScoreController va récupérer le score des joueurs qui ont joué dans la partie et va donner les résultats à la vue qui va l'afficher.

D. Les packages Util et Exceptions

Ces packages contiennent les classes java : CSVParser, TuileException et DominoException. CSV Parser permet de charger les dominos à partir d'un fichier csv. Les deux autres exceptions permettent de lever des exceptions spécifiques comme la non-présence de la tuile château sur le plateau d'un joueur, ou une mauvaise utilisation du domino.

E. Diagramme de classe entier

Voici le diagramme de classe complet de notre application :



IV - Fonctionnalités particulières

A. La gestion des IAs

La gestion d'une intelligence artificielle implique 2 points importants : la sélection d'un domino depuis une pioche et la sélection d'un placement du domino sélectionné.

1. La sélection d'un domino depuis la pioche

Pour l'IA simple, nous avons décidé que la sélection d'un domino depuis la pioche est aléatoire. L'IA normale va choisir un domino qui va lui faire rapporter le plus de points. Ainsi, l'IA va constituer une liste de toutes les possibilités possibles, et va choisir la meilleure. Cette fonctionnalité a nécessité la création d'un clone du plateau de l'IA (création d'un nouvel objet avec les mêmes propriétés, mais pas la même référence), car nous devons vérifier si le positionnement du domino est valide ou non, et le nombre de points du plateau. Mais l'IA ne prend pas en compte les points de bonus des différents modes de jeu.

Enfin, pour l'IA difficile, nous allons reprendre la même méthode. Nous allons rajouter aussi une fonction qui va prendre en compte les autres joueurs. L'IA pourra choisir un domino pour gêner l'adversaire. Cette IA prendra aussi en compte le mode de jeu utilisé.

2. Les décisions du placement

Après avoir fait la sélection du domino, l'IA n'a plus qu'à trouver le meilleur placement pour son plateau en reprenant les résultats trouvés précédemment.

B. Le calcul des scores

La fonction permettant de calculer le score n'était pas simple à réaliser. En effet, on doit déterminer tous les domaines différents du jeu. Un domaine peut être composé d'une ou plusieurs cases adjacentes (horizontalement et verticalement) du même type de terrain. Dans le domaine, on doit compter le nombre de cases et on fait le produit avec la somme des couronnes présentes dans les cases. Enfin, on fait la somme des points de chaque domaine pour obtenir le total des points. Pour faire cela, on va parcourir toutes les cases du tableau. Une liste de cases visitées est créée permettant de négliger les cases déjà prises en compte. Sur chaque case non visitée on va rechercher les cases similaires avec la fonction rechercherCaseSimilaire. La fonction va vérifier les cases adjacentes (haut, bas, gauche et droite). Si la case n'a pas été prise en compte, et qu'il est du même type de terrain que la case témoin, alors on va rappeler la fonction rechercherCaseSimilaire pour cette case. Avec cette fonction récursive, nous pouvons compter les points de chaque joueur.

En cas d'égalité, nous devons départager en fonction du plus grand domaine. Cette fonction reprend le même principe, mais ici on compte uniquement le nombre de cases similaire. En cas de nouvelle égalité, on va uniquement parcourir le tableau en comptant le nombre de couronnes présentes. S'il y a de nouveau égalité, les joueurs sont désignés comme vainqueurs *exæquos*.

C. Le placement de dominos sur le plateau de jeu

Nous avons décidé que le plateau sera de la taille donnée par le mode de jeu (soit 5x5 ou 7x7). Mais nous ne devons pas gêner les choix du joueur. Il faut que les choix de placements des dominos soient libres dans la mesure du possible. Si un joueur place son château à la position 0, 0. Il souhaite placer un domino sur les positions -1, 0 et -2, 0. Avec les fonctions de translation comme calculTranslation et

translationPlateau, on vérifie si en abscisse le nombre ne dépasse pas la taille du plateau de jeu. Dans notre exemple, on arriverait à 3 de longueur. Par conséquent, il est possible de translater et de faire passer la tuile Château de la position 0,0 à la nouvelle position 2,0. Le domino se placera en 0, 0 et en 1, 0. Cette fonctionnalité doit s'activer dans certains cas précis, comme le placement d'un domino en -1 ou 6 la limite du plateau.

La fonction placement valide permet de vérifier si un placement est valide dans le plateau de jeu. Elle retourne un message d'erreur si le placement serait invalide.

- La première erreur concerne les positions. Si les positions ne sont pas dans les limites autorisées alors la fonction enverra un message d'erreur : « placement invalide ».
- La seconde erreur concerne la limite autorisée pour le plateau. Il n'est pas possible de mettre plus de 5 ou 7 cases horizontalement ou verticalement. Ainsi cette fonction renverra un message d'erreur.
- La troisième erreur concerne le chevauchement d'un domino. En effet il est possible de placer un domino sur une case déjà prise. Par conséquent, la fonction ne doit pas laisser passer cette erreur.
- La quatrième erreur concerne le placement des dominos. Pour rappel, un domino doit être placé à côté du même type de terrain, ou à côté du château. Si ce n'est pas le cas, alors la fonction devra envoyer un message d'erreur concernant ce point.

Si les conditions sont validées, la fonction renvoie *null*, pour signifier que le placement est valide.

D. L'affichage du jeu avec une interface graphique

Dans JavaFX, il y a nativement 2 threads : le thread d'application, qui est le thread principal, exécute les modifications sur l'interface graphique tandis que le thread d'évènement est en écoute de tous les évènements déclenchés sur l'interface graphique. Seul le thread d'application peut modifier l'interface, sinon une exception est lancée. On peut utiliser la méthode *Platform.runLater(Runnable r)* pour exécuter une modification d'interface sur le thread d'application depuis un autre thread. Or dans notre projet, il faut parfois réaliser des attentes, notamment quand le programme laisse les joueurs placer leurs jetons et leurs dominos. On utilise des attentes de type *wait / notifyAll()* sur un objet dédié pour réaliser ces attentes, ainsi que des *Thread.sleep()* pour laisser un certain visuel pour que les joueurs humains voient l'avancement à vitesse convenable.

Le problème est qu'il est inconcevable des réaliser des attentes sur les deux threads présentés qui ne doivent pas être bloqués de par leur importance. Ainsi, il a fallu rajouter un troisième thread pour gérer une partie de jeu. C'est lui qui se met en attente des phases de pioche et de placement de dominos. Cela implique qu'il faut modifier le comportement de JavaFX pour *kill* ce troisième thread, car sinon l'application continuerait de faire tourner un thread pour rien.

Concernant JavaFX, c'est la vue (fichiers *.fxml*) qui va charger le contrôleur, automatiquement ou manuellement. Il est plus facile d'éditer la structure d'une scène dans le fichier plutôt qu'en Java. Ainsi chaque vue présentée (accueil, menu, partie) a un contrôleur dédié. En plus de cela, des sous-parties de la scène de partie (scores et pioche) ont elles-mêmes des contrôleurs et s'exécutent indépendamment de la scène mère. C'est une sorte « d'imbrication » d'une scène dans une autre. Il est possible de récupérer l'instance de contrôleur fils dans le contrôleur père et donc de faire interagir les contrôleurs entre eux. C'est sur cela que s'appuie les attentes d'actions. Par exemple, la pioche réalise un *notifyAll()* sur une objet passé en paramètre par la partie pour indiquer au thread de la partie de fermer la fenêtre modale de la pioche et passer au placement des dominos.

V – Problèmes rencontrés

A. L'utilisation de SceneBuilder

L'utilisation de SceneBuilder nous a permis de gagner du temps lors de la création des vues et la création du code pour les contrôleurs. Cependant, l'application n'est pas totalement fiable, car elle peut cesser de fonctionner.

B. La gestion du placement d'un domino

Cette gestion était particulièrement difficile à coder. En effet, il existe plein de conditions qui font que le placement d'un domino ne soit pas valide. Par ailleurs lors des tests du jeu, nous avons dû recoder les fonctions pour que la gestion soit parfaite. Cette partie nous a coûtée du temps.

C. La partie Interface graphique

Nous n'en avions jamais fait durant nos précédentes années de formation. La partie interface graphique a pris beaucoup de temps dans notre projet. En effet, il était important de réutiliser les fonctions du modèle, pour éviter les duplications du code. Il était nécessaire de créer du code dans le modèle qui soit réutilisable pour la version console et la version graphique.

D. Le calcul de translation

Cette fonction est une particularité de notre projet. Nous devons calculer les nouvelles positions si le placement du domino provoque un dépassement des bornes du tableau. Cette fonctionnalité a dû attendre les tests pour comprendre les bugs.

VI – Comment jouer à Domi'Nations ?

Il existe deux manières de jouer au jeu Domi'Nations : le jeu avec console et la version avec l'interface graphique. Le jeu comporte plusieurs modes qui sont tous jouables, indépendamment du choix du support.

A. Les modes de jeu

Il y a quatre modes de jeu implémenté dans notre solution du projet Domi'Nations :

- Mode Standard : Le jeu se déroule sur un plateau de 5x5 cases. Il n'y a pas de points bonus.
- Mode Grand Duel : Le jeu se déroule sur un plateau de 7x7 cases et se joue avec la totalité des dominos. Ce mode n'est jouable qu'à deux joueurs.
- Mode Empire du Milieu : Le plateau du jeu est standard. Les joueurs ayant leur château au centre du plateau gagne un bonus de 10 points.
- Mode Harmonie : Le plateau du jeu est standard. Les joueurs ayant remplis la totalité des cases de leur royaume reçoivent un bonus de 5 points.

B. Jeu avec la console

Pour jouer au jeu avec le mode console, il faut ouvrir l'IDE de votre choix (IntelIJ IDEA ou Eclipse) et lancer la fonction main de la classe DomiNationsConsole se trouvant à la racine du projet (dossier src). Une fois exécutée, un affichage semblable apparaît en console :

```
Bienvenue sur Dominations quel mode de jeu souhaitez-vous ?
0 - STANDARD
1 - GRAND_DUEL
2 - EMPIRE_DU_MILIEU
3 - HARMONIE
```

Pour choisir le mode de jeu, le joueur doit rentrer un nombre en 0 et 3.

Une fois le mode de jeu sélectionné, le nombre de joueur participant à la partie doit être renseigné à la suite du message suivant :

```
Vous avez sélectionné le mode de jeu : STANDARD
Nombre de joueurs ? (2 à 4)
```

Ce message ne s'applique pas au mode de jeu « Grand Duel » car le nombre de joueur est fixé à deux.

Vient ensuite l'inscription des joueurs dans le jeu. Tour à tour, chaque joueur va devoir renseigner s'il est une Intelligence Artificielle (IA) ou un joueur réel.

- Cas d'un joueur normal :

```
Joueur 1 veuillez renseigner votre pseudo ou écrivez ia pour créer un bot
laurent
laurent / vous êtes le roi Rose
```

Le roi est attribué au joueur après que son pseudonyme soit renseigné. La couleur du roi est affichée pour permettre au joueur de se repérer lors du déroulement des tours de jeu.

- Cas d'un joueur IA :

```
Joueur 2 veuillez renseigner votre pseudo ou écrivez ia pour créer un bot
ia
Veuillez choisir un niveau IA :
0 - Simple
1 - Normal
2 - Difficile
1
IA Créé avec comme niveau : Normal
veuillez saisir un nom pour ce bot :
Bot Manu
Bot Manu / vous êtes le roi Jaune
```

Dans ce cas, il faudra choisir la difficulté et ensuite renseigner le nom donné à cet IA. La couleur du roi est attribuée aussi à l'IA à la fin de son inscription.

Lorsque que tous les joueurs sont inscrits, la partie commence. Tant qu'il reste des dominos en jeu, chaque tour se déroule de la manière suivante :

- Affichage des dominos piochés
- Tri aléatoire de l'ordre de passage des rois qui vont piocher
- Choix des dominos par les rois
- Pose des dominos par les rois en fonction de leur numéro et de l'ordre des rois

Voici un exemple de début du tour de pioche avec le tirage par un joueur :

```
[ 30 ]
[ 33 ]
[ 37 ]
[ 43 ]
[ {MER, Couronnes: 1},{CHAMPS, Couronnes: 0} ]
[ {MER, Couronnes: 1},{FORET, Couronnes: 0} ]
[ {MER, Couronnes: 0},{PRAIRIE, Couronnes: 1} ]
[ {CHAMPS, Couronnes: 0},{MINE, Couronnes: 2} ]
Entrez le numéro pour choisir le domino
Tour : Rose
Entrez le num du domino : (entre 0 et 3)
0 - domino : [ {MER, Couronnes: 1},{CHAMPS, Couronnes: 0} ]
1 - domino : [ {MER, Couronnes: 1},{FORET, Couronnes: 0} ]
2 - domino : [ {MER, Couronnes: 0},{PRAIRIE, Couronnes: 1} ]
3 - domino : [ {CHAMPS, Couronnes: 0},{MINE, Couronnes: 2} ]
3
Domino tiré : [ {CHAMPS, Couronnes: 0},{MINE, Couronnes: 2} ]
```

L'IA pioche aussi (comme un joueur normal) et c'est écrit dans la console comme suit :

```
Tour : Jaune
IA a choisi le domino : [ {MER, Couronnes: 1},{CHAMPS, Couronnes: 0} ]
```

Les tours de pioches continuent jusqu'à ce qu'il ne reste plus de dominos dans la pioche.

Ensuite s'enchainent les tours de placement des dominos par les joueurs :

- Cas d'un joueur normal :

```
Placement : Rose
[[Id: 37, Case 1: {MER, Couronnes: 0}, Case 2: {PRAIRIE, Couronnes: 1}], [Id: 43, Case 1: {CHAMPS, Couronnes: 0}, Case 2: {MINE, Couronnes: 2}]]
     1       2       3
 1


 2         CHAT
           --C0

 3


Domino : [Id: 37, Case 1: {MER, Couronnes: 0}, Case 2: {PRAIRIE, Couronnes: 1}]
x : 2
y : 1
Orientation : n
     0       1       2       3
 0


 1         PRAI
           --C1

 2         MER-    CHAT
           --C0    --C0

 3


Fin tour de : Rose
```

La coordonnée en x correspond au numéro de la ligne voulue et celle en y à la colonne.

L'orientation du domino correspond à l'une des valeurs suivantes : e pour EST, n pour NORD, s pour SUD et o pour OUEST. Cette orientation se fait à partir des coordonnées choisies et de la première case du domino à placer, c'est à dire que la première case prendra ces coordonnes et l'autre case prendra celle calculée en fonction de l'orientation.

- Cas d'un joueur IA :

```
IA Placement : Jaune
IA a décidé de placer son domino en x : 4/ y : 2 / sens : N
Domino concerné : [Id: 30, Case 1: {MER, Couronnes: 1}, Case 2: {CHAMPS, Couronnes: 0}]
     1       2       3
 1


 2         CHAT
           --C0

 3         MER-
           --C1

 4         CHAM
           --C0

 5
```

Une fois que tous les dominos ont été placés ou défaussés par les joueurs, le score final s'affiche et le vainqueur est annoncé :

```
Participants :
Joueur : Bot Manu/ Roi : Jaune / score : 18
Joueur : laurent/ Roi : Rose / score : 10
Vainqueur : Roi Jaune Bot Manu
Fin du jeu !
```

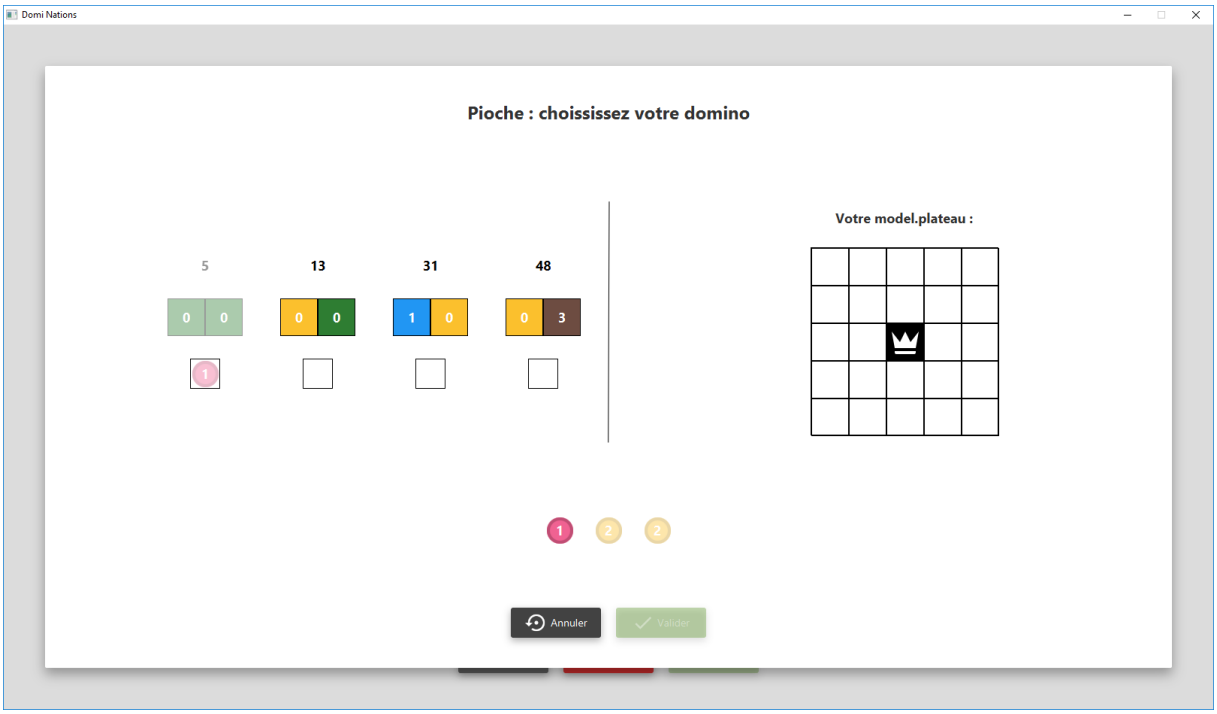
Le jeu est fini, pour rejouer, il faudra alors relancer la fonction main de la classe Domi'Nations.

C. Jeu avec l'interface graphique

Pour lancer le jeu en version graphique, il faut lancer le main Domi'Nations.

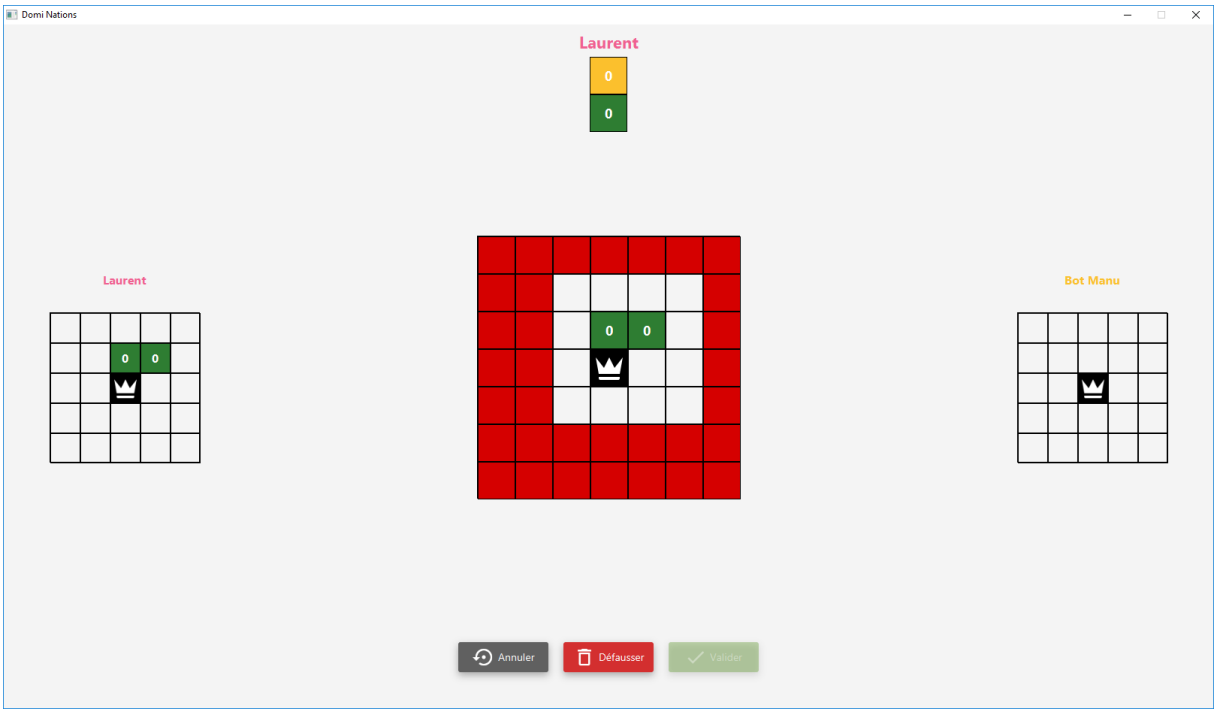
Pour commencer à jouer, il faut cliquer sur le bouton Jouer.

La deuxième fenêtre permet de sélectionner les options de jeu. Il est important de remplir tous les champs.



Après avoir validé les options de jeu, les joueurs vont pouvoir découvrir la pioche. Et chacun à son tour, les joueurs vont pouvoir sélectionner un domino grâce au système de drag and drop avec le jeton. Ils doivent valider la sélection.

Après cela, il y a une phase de placement domino pour les joueurs.



Les joueurs vont avoir leur plateau sous leurs yeux. Ils vont pouvoir placer leur domino avec le même système de drag and drop, puis ils vont pouvoir valider. Ils peuvent annuler leur placement avant

validation, ou alors défausser le domino. Il est aussi possible de tourner le domino en faisant un clic droit.

A la fin du jeu, une fenêtre va s'afficher pour donner le score ainsi que le vainqueur du jeu.

Concernant ce point, Le vainqueur sera départagé en fonction du score, du plus gros domaine en cas d'égalité et en nombre de couronnes s'il le faut.

VII – Bibliothèques et logiciels utilisés

Concernant la partie interface graphique, nous avons utilisé une librairie et aussi une application. La librairie est JFoenix : https://github.com/jfoenixadmin/JFoenix. Elle permet d'avoir une interface graphique de style Google/Android pour des composants JavaFX. Nous avons utilisé l'application SceneBuilder, pour créer toutes les interfaces du jeu plus facilement. Nous avons choisi JavaFX car c'est une librairie très complète pour la réalisation d'applications de bureau. SceneBuilder permet de placer nos composants visuellement, de modifier le fichier fxml sans y toucher à la main et enfin générer le code de base pour le contrôleur : les attributs et la méthode *initialize,* appelée à l'instantiation. Les champs annotés @FXML seront instantiés automatiquement en allant chercher les composants portant le même nom (ou identifiant en JavaFX, avec l'attribut de balise *fx:id*), ce qui facilite le développement.

VIII - Conclusion

Pour ce projet, nous avons réussi à créer la version console et la version applicative du projet Domi'Nations. Nous avons pu mettre en place l'architecture MVC pour ce projet qui était complexe mais riche en enseignement. Nous avons eu des problèmes concernant le débogage de l'application que ce soit pour la version console mais aussi la version applicative. Le temps alloué pour ce projet semble être un peu court. En effet nous n'avons pas pu modéliser l'IA difficile que nous comptions le faire pour le concours d'IA pour mi-février. De même que pour la partie applicative, nous avons manqué de temps pour cette partie.