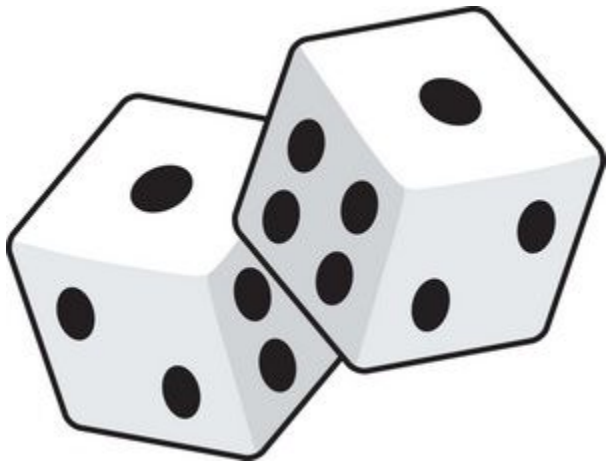
DICE GAME

Mourey Arthur
Pierson Guillaume



Rappel du Sujet

L'objectif de ce projet est la conception et réalisation d'un système distribué concernant un jeu de lancé de dés (Dice Game). Afin de mener à bien ce travail, un cahier des charges nous a été délivré pour connaître les attentes clients.
Une partie standard dans l'application du jeu de dés se déroule de la manière suivante :

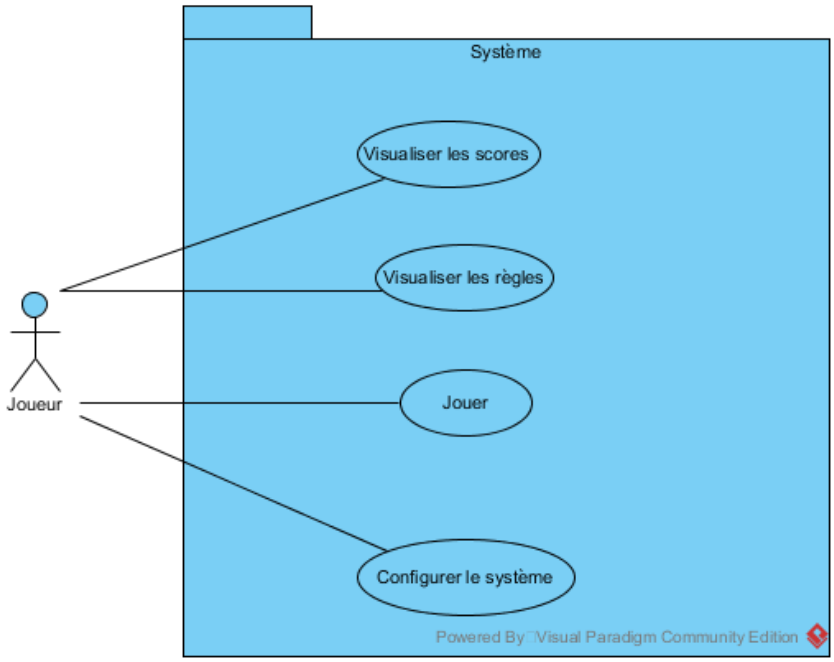
- Un joueur s'identifie avec son nom et prénom.
- Il peut alors lancer les dés 10 fois en une partie.
- A chaque fois que le lancé indique une valeur égale à 7, le joueur gagne 10 points.
- A la fin de la partie, son score s'inscrit dans son tableau des scores

Le joueur identifié peut également consulter ses scores personnels et les scores de la totalité des joueurs.
D'un point de vue plus technique, la persistance des parties jouées s'effectue de plusieurs façons différentes. Les sauvegardes se font à l'aide d'un fichier, d'une base de données MariaDB et MongoDB. Au chargement, les données sont récupérée de manière aléatoire entre ses multiples bases.
Aussi, il était demandé d'utiliser le patron Factory pour réaliser la persistance et un singleton pour la classe Randomizer, qui permet de générer un nombre aléatoire représentant le résultat du lancé de dés.

Conception

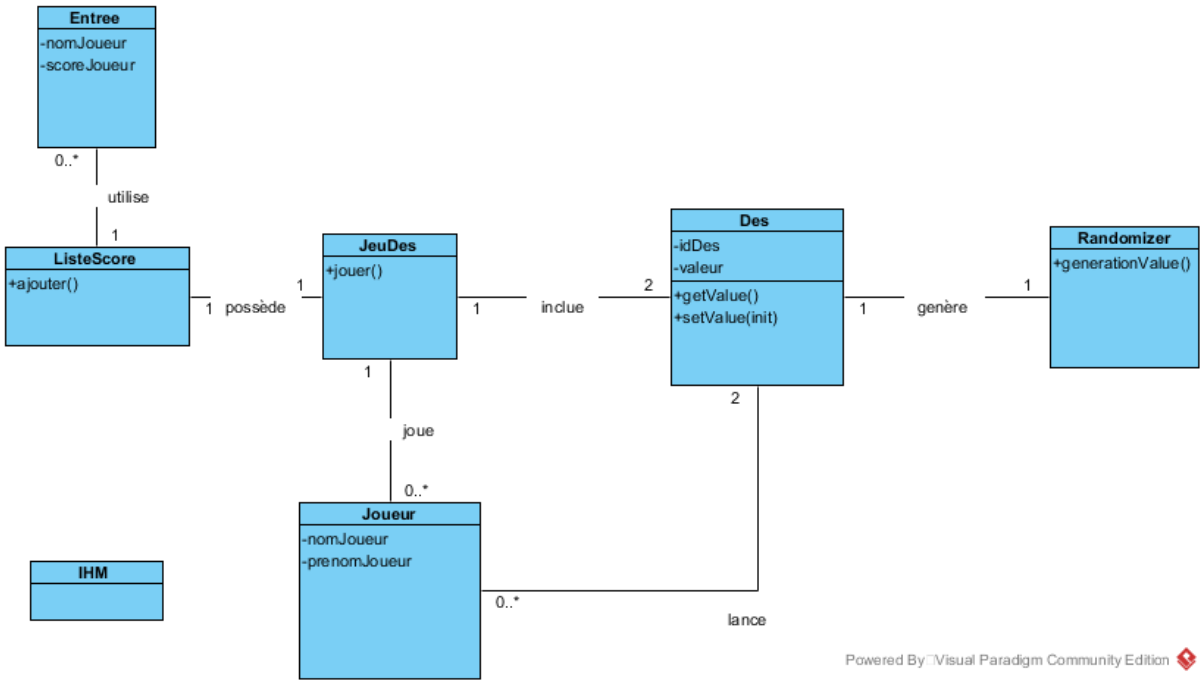
Tout d'abord, nous avons modélisé notre application à l'aide des diagrammes UML suivants, réalisés grâce à Visual Studio :

Use Case



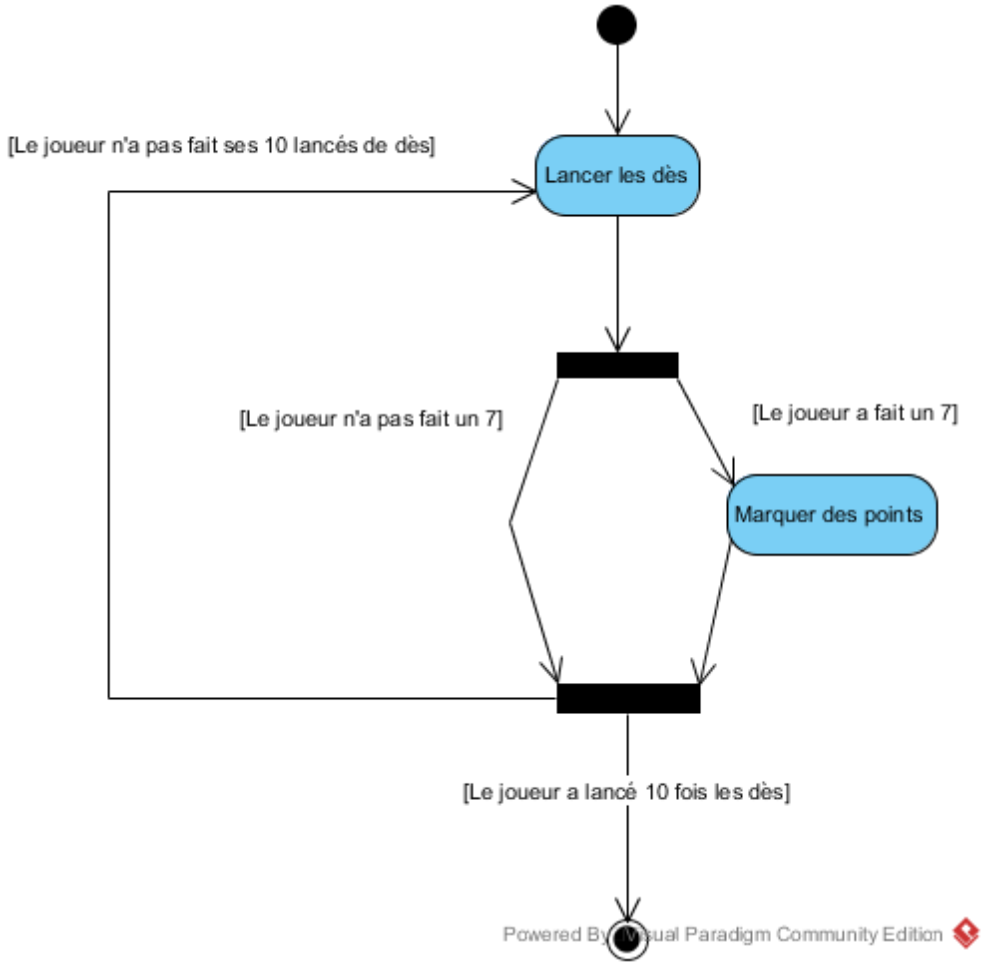
On peut constater sur ce diagramme qu'il n'y a qu'un seul acteur, étant donné qu'il n'y a pas de différenciation entre les diverses utilisateurs, ils sont tous des joueurs. Le joueur de l'application peut donc jouer, visualiser les règles et les scores et également configurer le système.

Diagramme de classe



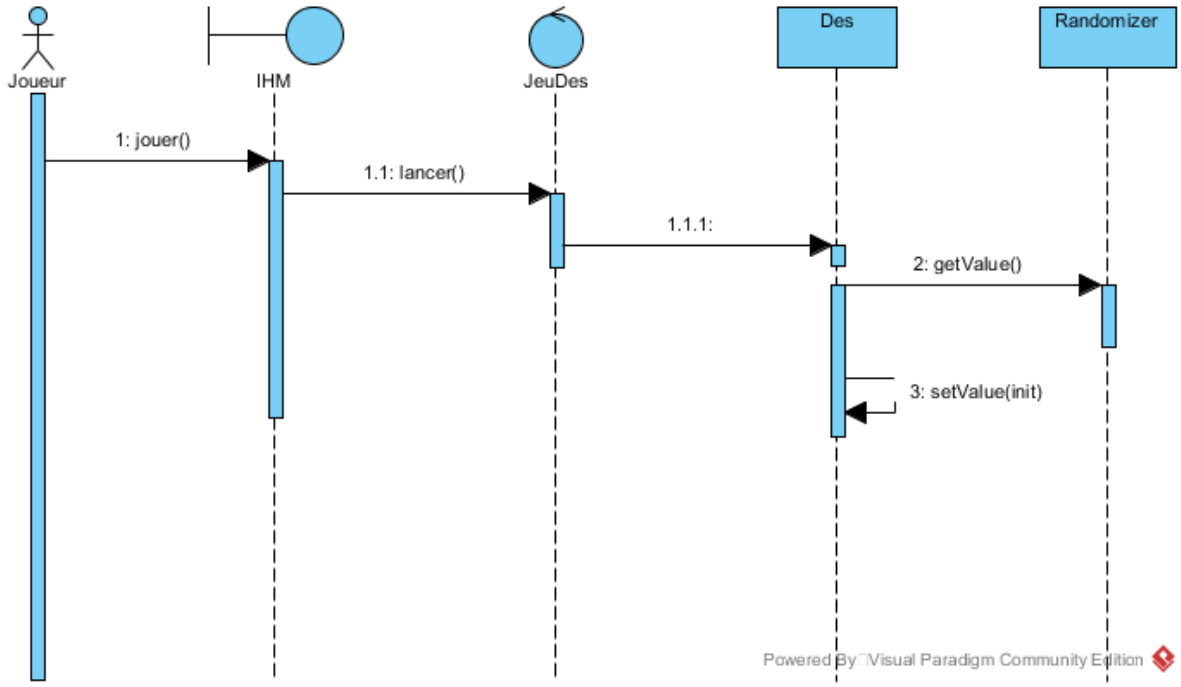
Dans ce diagramme de classe, nous avons évidemment la classe JeuDes qui va servir de moteur pour l'application. Les différentes classes contrôleur, ainsi que les différentes classes vues ne sont pas représentées ici, car peu pertinente pour cette première modélisation.
On distingue alors d'autres entités essentielles, telles que les dés, qui permettront aux joueurs d'obtenir les résultats attendus. Ces dés utilisent la classe Randomizer qui va générer pour eux 2 nombres aléatoires pour chaque valeur de dés.
La classe Joueur va regrouper toutes les informations des joueurs, et les scores vont être stockés pour être regroupés dans des listes, afin de permettre aux utilisateurs de visualiser les meilleurs scores enregistrés par eux ou d'autres joueurs.

Diagramme d’activité



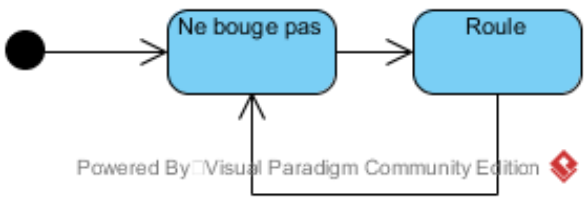
Nous avons choisi de présenter ici un seul diagramme d’activité. Les autres, ne comprenant que 3 ou 4 éléments, semblaient moins pertinents.
On peut constater que l’action du lancé de dés se découpe en plusieurs parties. En effet, lorsque le joueur lance les dés, il y a une possibilité qu’il réalise un 7. Si c’est le cas, alors le joueur marque des points (10 est la valeur que nous avons fixé ultérieurement) et peut relancer les dés, sinon le joueur relance les dés.
Ce mécanisme peut s’effectuer 10 fois seulement, ensuite la partie est terminée. (Nous avons choisi de créer un bouton pour relancer la partie une fois terminée pour permettre au joueur de rejouer sans avoir à quitter et relancer l’application)

Diagramme de séquence



Les joueurs, inscrits dans l'application, peuvent passer par l'IHM pour lancer les dés. Dans ce diagramme de séquence, on peut constater que la classe JeuDes va démarrer le processus de génération de valeur des dés. La classe Des va appeler le Randomizer pour récupérer les valeurs aléatoires, les dés vont alors changer leur valeur.

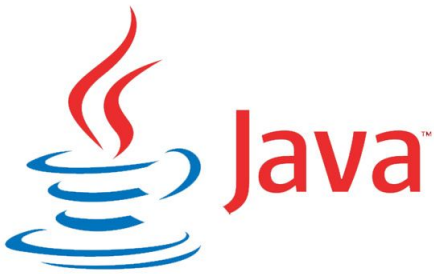
Diagramme d'état



Le diagramme d'état ci-dessus représente les états possibles des dés. En effet, on peut voir qu'à l'initialisation, les dés ne bougent pas. Lorsque le joueur décide de lancer les dés, ces derniers roulent (on pourra le voir à l'aide d'une animation courte)
Les autres diagrammes n'ont pas été représentés ici, car la plupart du temps les autres entités ne possèdent qu'un seul état.

Réalisation

Java



Nous avons choisi d'utiliser le langage JAVA pour réaliser ce projet, car les membres de notre groupes ont l'habitude de l'utiliser. Cela a facilité et accéléré le développement de l'application.

JavaFX et FXML

Ensuite, nous avons également intégré une interface graphique grâce à la librairie Javafx. Pour un meilleur rendu visuel, l'utilisation de fichiers FXML nous a permi de mieux structurer nos pages et donner un résultat au plus proche de nos attentes.
Les fichiers en questions ont été réalisés à l'aide de Scene Builder, qui est un logiciel permettant de générer le code des pages FXML à partir d'une interface dans laquelle nous pouvons placer les différents éléments de la scène à notre convenance par "drag and drop".

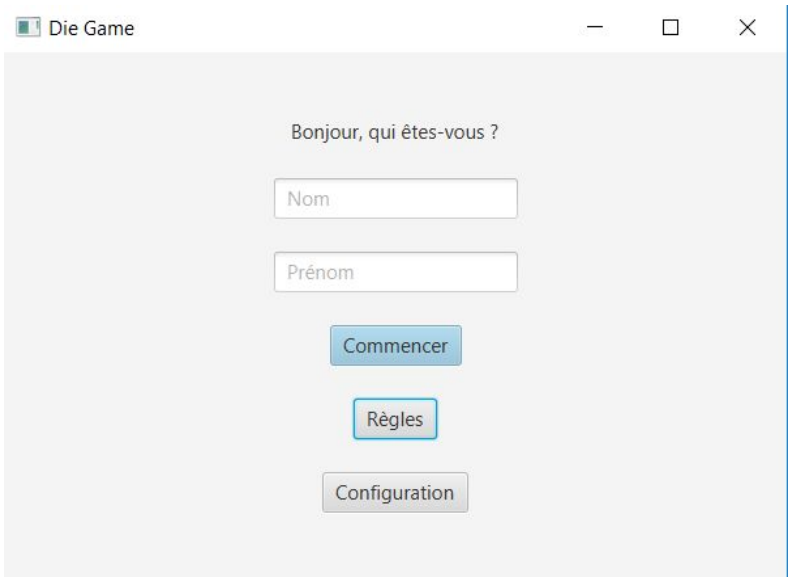
Base de données

La première base de données est enregistrée sur un fichier. Ce fichier est au format XML. Il nous a permis de structurer plus aisément les scores des différents joueurs ayant enregistrés leurs parties.

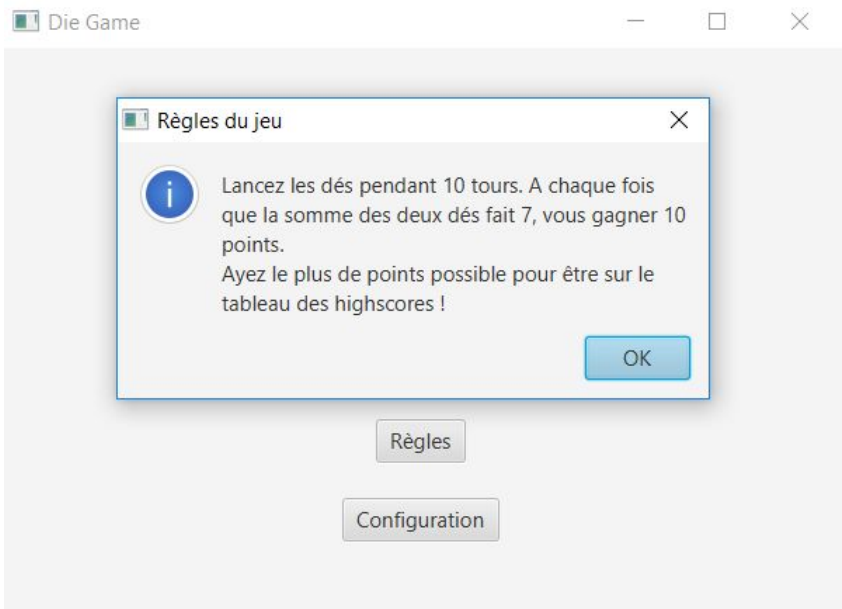
La deuxième et troisième base de données sont respectivement des bases de données MariaDB et MongoDB. Afin de permettre une sauvegarde rapide et optimisée, nous avons utilisé le framework Hibernate. Pour ce faire, nous avons intégré les dépendances nécessaires dans le POM de notre projet Maven et utilisé les annotations liant le code à la base de données.

Aperçu de l’application

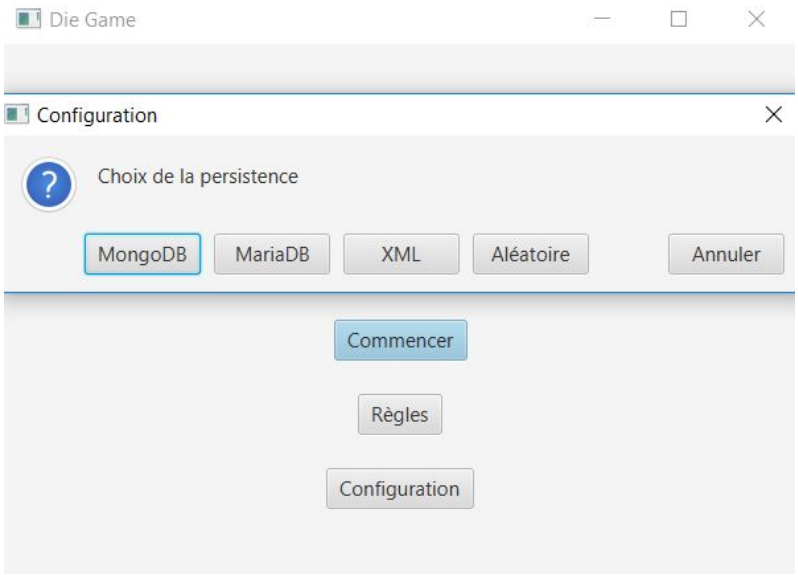
Page d’accueil



Affichage des règles



Configuration de l'application



Fenêtre de jeu

Choix personnels

Nous avons choisi de simplifier au plus l'interface pour permettre à l'utilisateur de naviguer entre les différentes pages sans difficultés.
Nous avons également intégré des titres accrocheurs, des images et des animations dynamiques pour le lancé des dés afin d'améliorer l'ergonomie et l'expérience utilisateur pour ce jeu.

Problèmes rencontrés

Les principales difficultés que l'on a rencontrées lors de la création de cette application ont concernées le temps, ou plutôt la planification et l'organisation des séances de travail entre les différents projets qui nous ont été attribués.
Aussi, la configuration du framework Hibernate OGM était légèrement différentes et inconnus des membres de notre groupe pour ce qui était de l'adapter à la base MongoDB qui fait partie de la mouvance NoSQL.

Conclusion

Ce projet nous a permis d'appliquer les concepts et méthodes vues en cours dans le cadre de la réalisation d'une étude concrète. Comme le premier projet de cette matière, il nous a fait prendre conscience de l'importance d'une conception UML dans un projet.
D'autre part, nous avons passé plus de temps sur la mise en place de patrons tels que le singleton et la factory, ainsi que sur des contraintes de persistances associées à différentes bases de données.