Dossier Ingénierie Logicielle

Groupe 5 : Jeu 2584

Louis Behr – Léa Thamié – Théo Driutti

13/04/2018

Introduction

Nous avons eu pour mission lors de ce projet de réaliser une variante du jeu 2048, le 2584 qui utilise la suite de fibonacci en lieu et place de la table de multiplication de 2.
Dans ce rapport nous allons présenter les différents achievements que nous avons réussi à atteindre ainsi que présenter les diagrammes demandés.

Recette informatique

Dans cette partie nous verrons quels sont les achievements que nous avons réalisé et nous les détaillerons un par un.
Liste des réalisations:

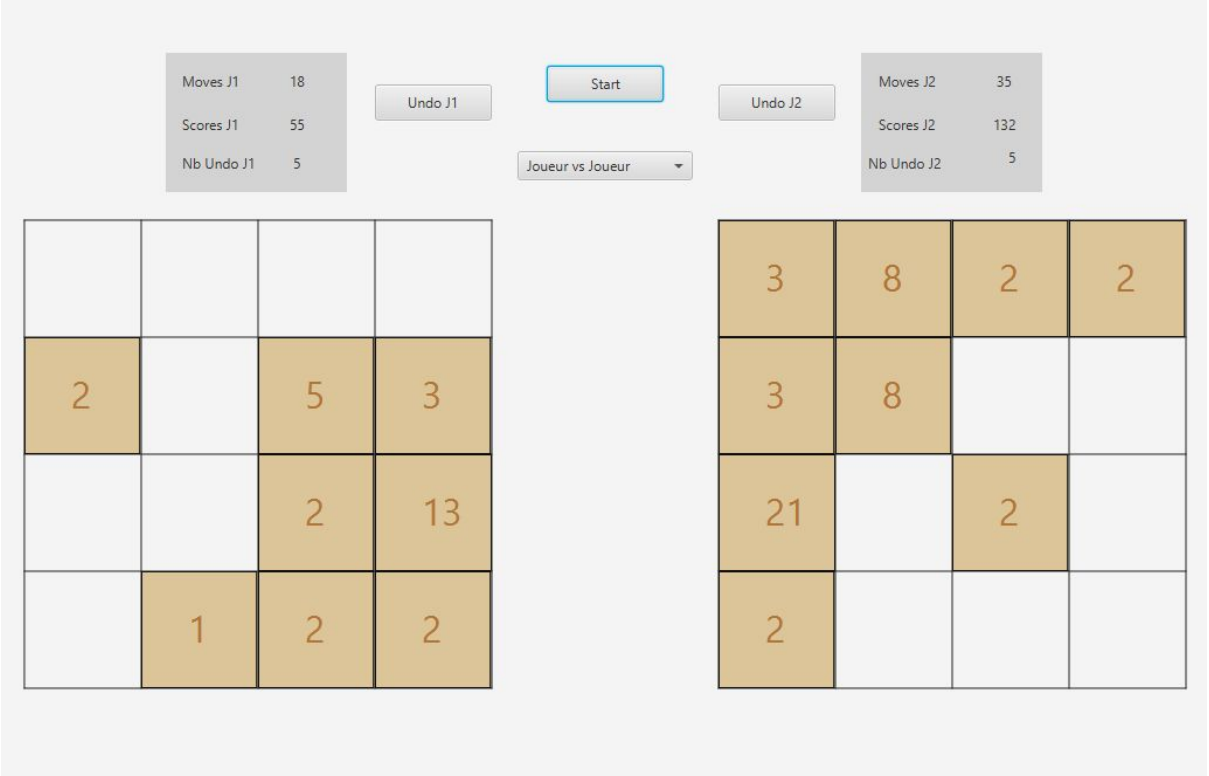
- Jeu 2584 à deux joueurs dans la console (1)
- Jeu à deux joueurs avec interface graphique sur le même ordinateur(2)
- Possibilité pour chaque joueur d'annuler son dernier coup avec un bouton "undo", utilisable seulement 5 fois (4)
- Possibilité de jouer contre un joueur aléatoire (5)
- Possibilité de voir le résultat des 10 dernières parties (8)

Nous avons choisi ces objectifs en fonctions de nos compétences, mais aussi en fonction de leur indépendance dans leur réalisation, nous voulions pouvoir nous répartir les tâches de manière à pouvoir les sans dépendre de l'avancement des autres membres de l'équipe.

Jeu du 2584 dans la console

Pour l'exécution du jeu dans la console nous nous sommes inspirés du tp de la deuxième année de licence.
Les deux joueurs commencent avec la même grille. Ils jouent à tour de rôle dans la console, la partie s'arrête quand l'un des joueurs a perdu ou gagné. Le joueur a perdu lorsqu'il ne peut plus effectuer aucun mouvement, c'est à dire que toutes les cases sont occupées et qu'elles ne peuvent pas être fusionnées. Les cases se fusionnent lorsqu'elles sont voisines de suite, pour vérifier si elles sont voisines dans la suite on d'abord généré toute la suite de Fibonacci jusqu'à l'objectif dans la Classe SuitesMathématiques, ensuite dans la classe Case on teste si la somme des deux cases appartient à la liste générée dans SuitesMathématiques. Le joueur gagne lorsqu'il a atteint l'objectif, l'objectif est défini dans Parmetres qui est une classe abstraite qui contient les constantes et quelques fonctions du projet. Lorsqu'une case est déplacée une nouvelle apparaît cette nouvelle case est un 1 ou 2. Les deux cases ont des probabilité d'apparition différentes.

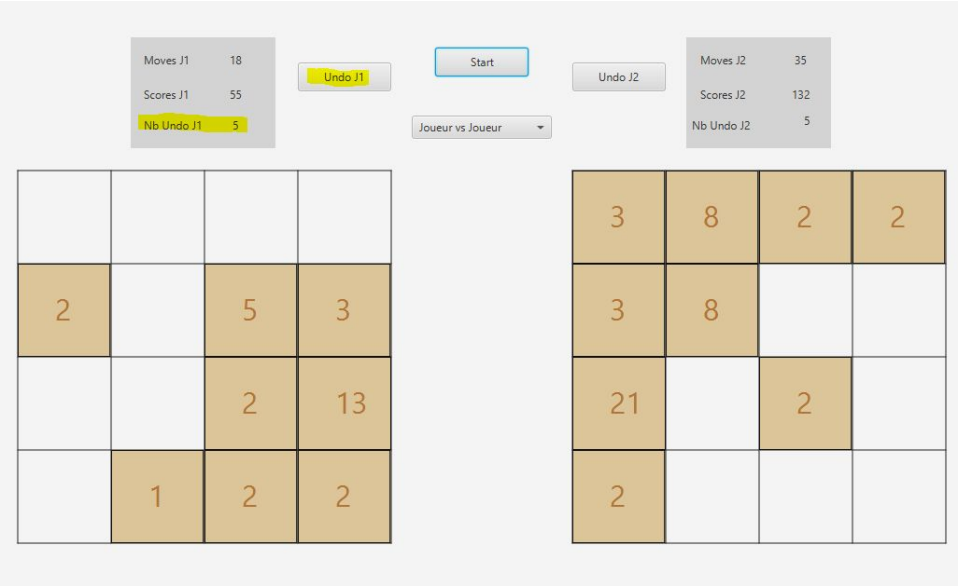
Jeu à deux joueur avec interface graphique



Au lancement du programme une fenêtre graphique s'ouvre, le jeu se lance en cliquant sur le bouton "Start" et les deux joueurs peuvent déplacer leurs tuiles dans les quatres directions avec les touches du clavier.
Le joueur un utilise les touches "q", "z", "s" ou "d" et le joueur 2 utilise les touches "o", "k", "l" ou "m". Chaque joueur peut jouer à son rythme il n'a pas à attendre que l'autre joueur est joué pour jouer à son tour. Cependant les mouvements sont comptés et affichés pour chaque joueur au dessus des grilles. Quand un joueur est bloqué ou qu'il gagne la partie s'arrête. Les scores sont mis à jour à chaque déplacement le case, c'est l'addition de toutes les fusions qui ont eu lieu au cours de la partie (une fusion a lieu lorsque deux cases sont voisines dans la suite de Fibonacci).

Undo

Le undo permet au joueur de revenir au coup précédent, cette action ne peut être réalisée seulement cinq fois, au delà le bouton n'a plus aucun effet. Le bouton n'a aucun effet lorsque le joueur effectue des mouvements "dans le vide" ou lorsque la partie vient de commencer, dans ces cas le nombre de undo restants au joueur ne diminue pas.
Le nombre de undo restant est affiché à côté des scores avec le bouton en haut de la fenêtre.



Le undo est géré dans la classe Joueur.

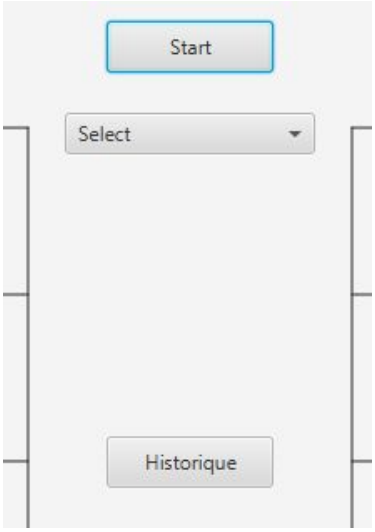
```
public class Joueur implements Parametres {

    protected Grille grilleActuelle;
    protected Grille grilleTampon;
    protected int scoreMax;
    protected int score;
    protected int nbUndo;
    protected int deplacement;
```

Chaque joueur a comme a pour attribut un nombre de undo disponibles, une grille actuelle et une grille tampon, de sauvegarde. Dans une méthode undo de la classe Joueur, après avoir testé que le undo était utilisable, on remplace la grille actuelle par la grille tampon et on décrémente le compteur de undo. Ce compteur est initialisé à 5 dans le constructeur de joueur.

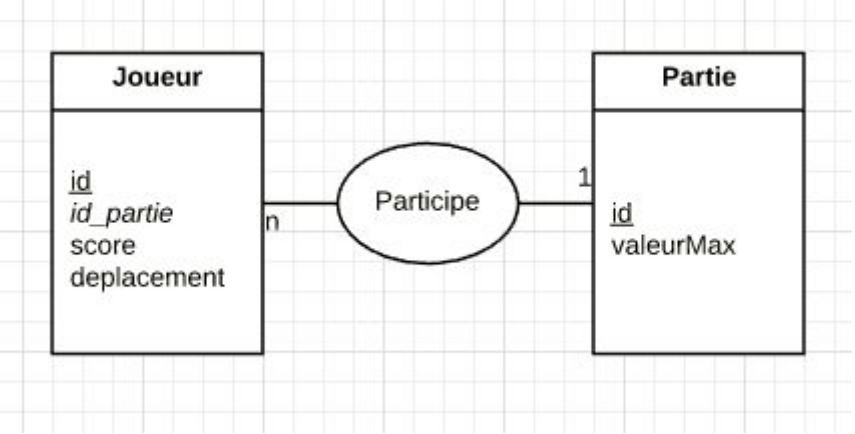
Le joueur aléatoire

Au début de chaque partie on choisi le mode de jeu avant de lancer la partie avec le bouton “Start”. Lorsque qu’on joue contre le mode random, le joueur artificiel joue à notre rythme, il joue à chaque coup du joueur et il ne joue que si le coup a entraîné un déplacement de tuiles.
Le joueur aléatoire est géré dans une classe RandomPlayer qui hérite de la classe joueur. Les directions sont codées par des chiffres de -2 à 2, le choix du préplacement se fait dans une fonction “jouer” qui tire des entiers au hasard entre -2 et 2 jusqu’à ce que la direction entraîne un déplacement, ainsi il n'effectuera pas de déplacements inutiles (qui n'entraînent pas de déplacement des tuiles). Dans cette version le joueur aléatoire n’est supporté qu’en mode graphique.

Historique de jeu



L’affichage des dix dernières parties se fait dans la console mais se déclenche à partir d’un bouton dans la fenêtre graphique. le bouton historique est présent même avant le début de la partie.
La connexion à la base de donnée sur fait dans la classe EnchangeBDD du Model.
Plusieurs fonctions s’y trouvent, la fonction recupPartie est appelée par le bouton “historique” elle comporte deux requêtes SQL. La première récupère les id et les scores des dix dernières parties, la deuxièmes récupère et affiche les scores.
Rq1 : “SELECT id, valeurMax FROM `partie` ORDER BY ID desc LIMIT 10”
Le résultat de la requête est ensuite stocké dans deux listes. La première contiendra les dix derniers id des parties de la base données et la deuxième contiendra les plus grandes valeures des tuiles des dix dernières parties.
Rq2 : "SELECT score, deplacement FROM Joueur WHERE id_partie=" + id.get(i)
Cette requête dépend de la première, dans une boucle, pour chaque partie elle va aller chercher les joueurs.
Une autre fonction est appelée en fin de partie et permet d’ajouter la partie à la base de données. On ajoute en premier la partie, puis on récupère le dernier id, et on ajoute des joueurs avec comme id_partie le dernier id. Les id sont générés automatiquement.
Ci dessous le diagramme des tables de notre BDD.



Description de l’application

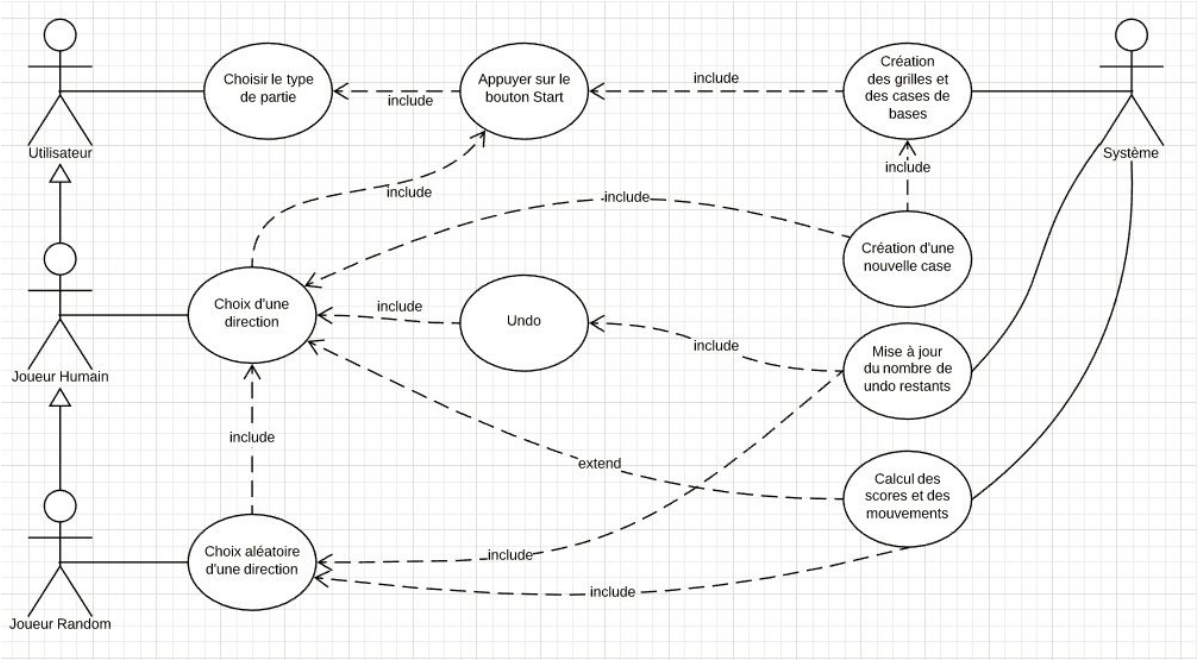
Il est possible de lancer l’application via un exécutable .jar qui sera trouvable dans le dossier dist (ainsi que l’ensemble de la javadoc)

L’application permet de jouer au jeu 2584 à deux, soit à deux joueurs qui s’affrontent l’un contre l’autre, soit contre un joueur artificiel qui joue aléatoirement. Pour un seul lancement de l’application plusieures parties peuvent être lancées, il n’est pas nécessaire de finir une partie pour en commencer une autre, il suffit de choisir un mode de jeu et d’appuyer sur le bouton Start en haut de l’écran. Chaque joueur bénéficie de 5 undo, le undo lui permet d’annuler son dernier coup. Les scores sont affichés en continu dans des cadres au dessus des grilles. Le déplacement des cases des grilles se fait avec les touches du clavier. Lorsque le mode de jeu est contre le joueur artificiel, les touches du joueur 2 sont désactivées.
Le projet comporte deux main un pour l’affichage graphique classique l’autre pour une version console. Dans le cas présent la version console est moins aboutie, elle ne permet ni le undo ni la possibilité de jouer contre un joueur aléatoire. Si l’on veut lancer la version du jeu en console il suffit d’ouvrir le code de l’application dans netBeans, cliquer sur l’onglet files, puis clic droit sur build.xml puis run target puis run-without-gui. Deux utilisateurs pourront alors jouer chacun leur tour en suivant les instructions données par la console.

Diagrammes UML

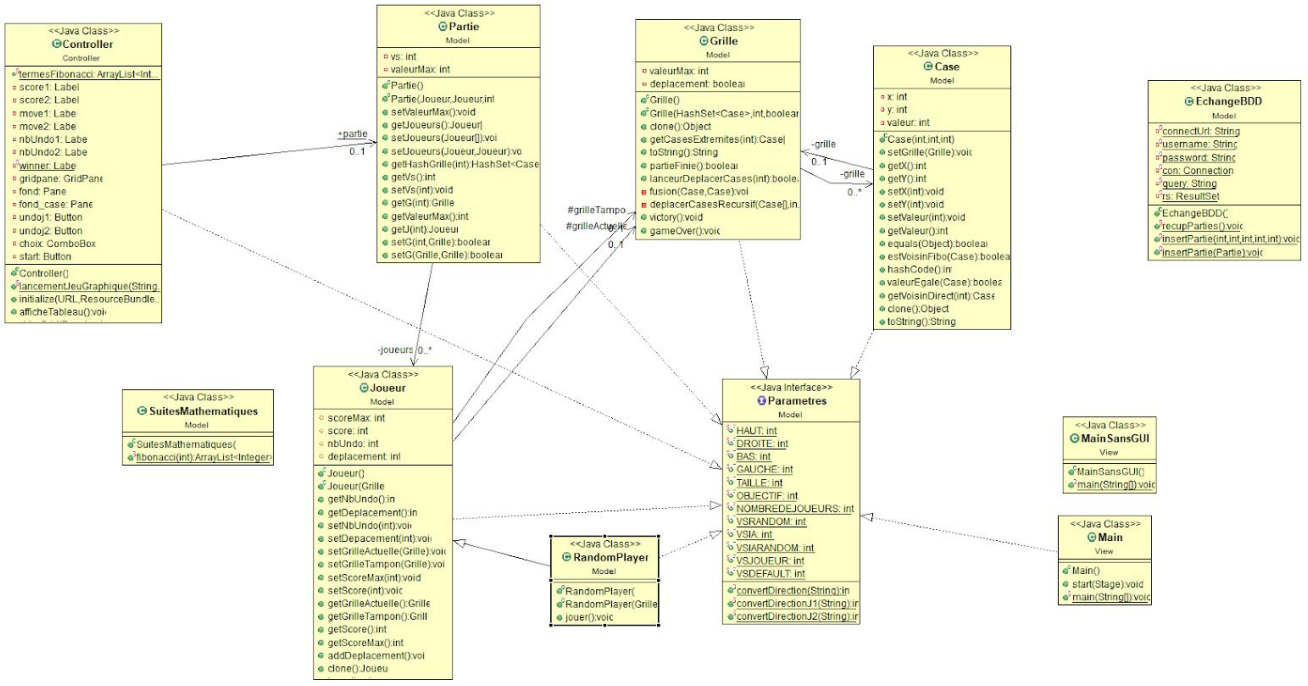
Dans cette partie nous allons décrire les différents diagrammes que nous avons créé dans le cadre de ce projet. Nous avons réalisé un diagramme de classes, un diagramme de cas d’utilisation ainsi que deux diagrammes de séquences représentant le déroulement de deux fonctions de notre projet.

Diagramme de cas d’utilisation



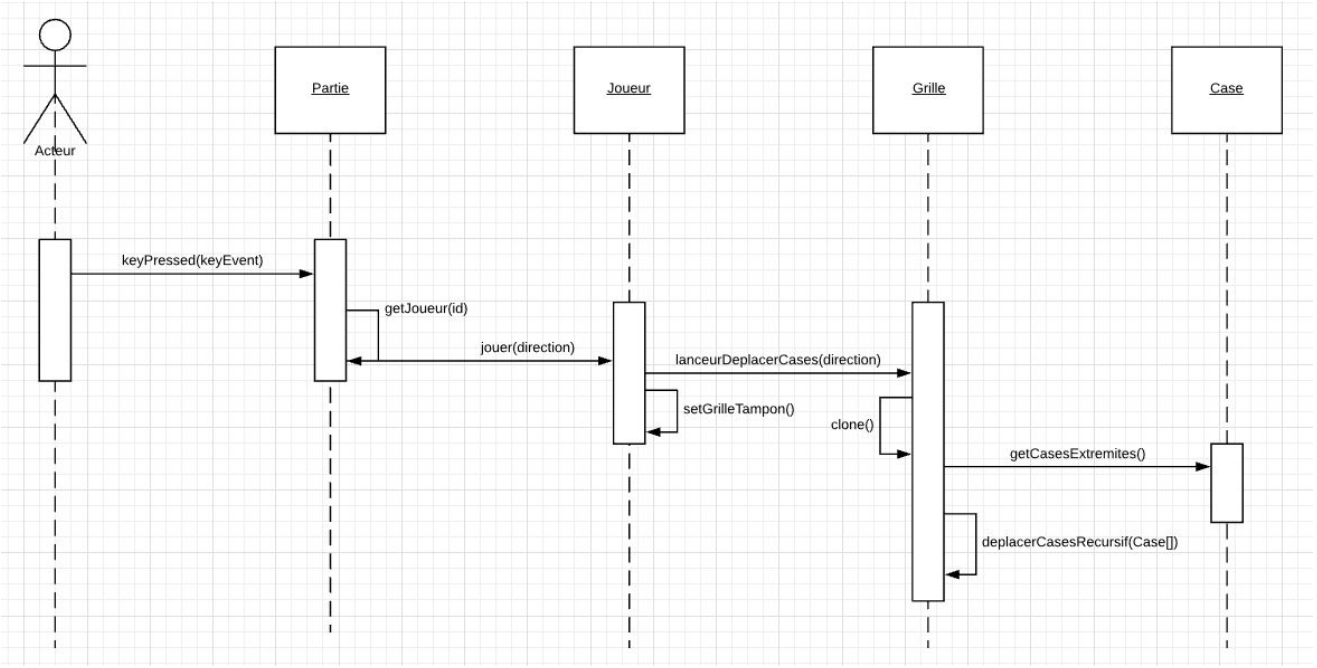
Comme on peut le voir dans ce diagramme il y a plusieurs types d’acteurs. Tout d’abord l’utilisateur qui est la personne qui va choisir le type de partie ainsi que décider quand commencer la partie. Il va ensuite devenir un joueur et pouvoir décider de comment jouer ou si il souhaite défaire son coup précédent. On remarque que le Joueur Random est aussi un joueur mais pas humain, et qu’il attendra que le joueur humain joué pour effectuer un coup aléatoire.
Le dernier acteur est le système qui va effectuer un certains nombre d’action en fonction du déroulement de la partie et des choix du/des joueurs.

Diagramme de classes

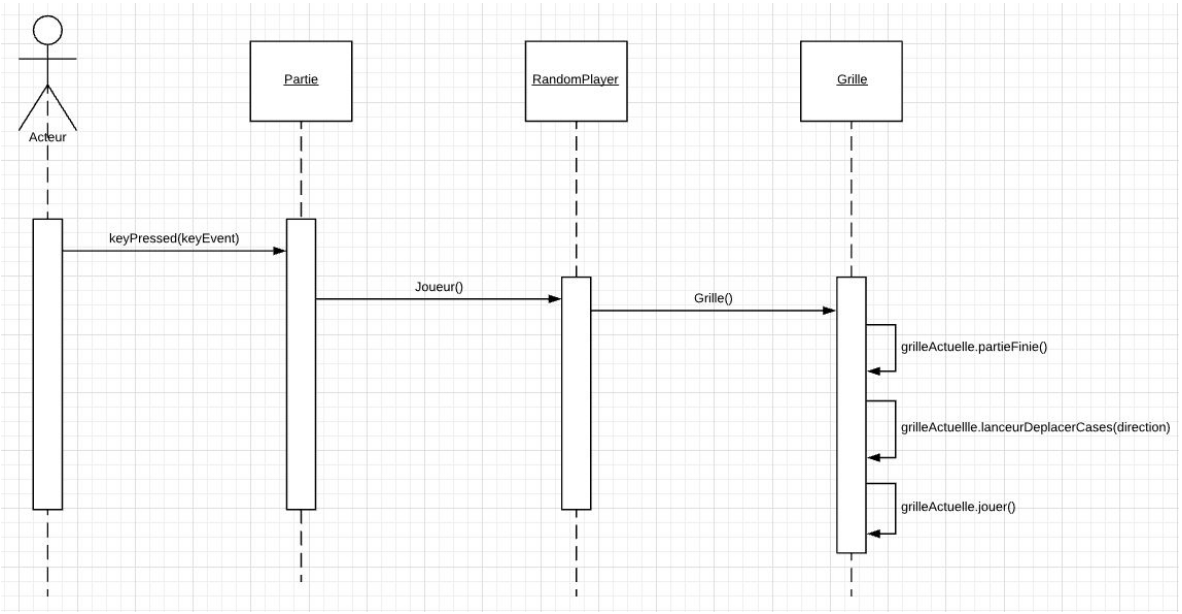


Nous avons un total de 11 classes dans notre projet. Les plus importantes étants Controller, Partie, Grille, Case et Joueur.
Controller contient toute les fonctions qui permettent de manipuler le visuel du projet ainsi qu'un grand nombre des fonctions qui permettent de manipuler les différentes données dont nous allons avoir besoin (les objets Grilles qui contiennent les objets Cases par exemple)
L'interface Paramètres nous permet d'avoir un certains nombre de constantes que nous allons utiliser tout au long du projet.
Une classe que nous avons fortement utilisé qui n'est pas dans ce diagramme est le FXMLDocument où l'on a pu créer, à l'aide de SceneBuilder, le visuel du projet.

Diagramme de séquence



Ce diagramme de séquence représente de manière simplifié les différentes actions effectuées pour obtenir le déplacement d'une case dans le programme.



Ce diagramme de séquence représente d'une manière simplifiée, encore une fois, le fonctionnement de la méthode Jouer() du joueur aléatoire.

Liens vers les espaces collaboratifs

Lien vers le GitHub:
https://github.com/behr22u/groupe5_ingenierie_logicielle

Lien vers le Trello:
https://trello.com/invite/b/ZxZxr48q/9fdbd67ababbd3fa3c7bfd356f072300/r%C3%A9paration-des-t%C3%A2ches

Documentation (Javadoc)

Conclusion

Notre projet est incomplet et présente quelques bugs mais il est fonctionnel. Parmi les achievements que nous avons pu obtenir il est encore possible d'effectuer certaines améliorations et optimisations.

Le code du projet est adapté à l'implémentation du reste des achievements et plus particulièrement celui de l'IA qui cherche à perdre toute seule.