

# Diaballik : Rapport de modélisation

Adrien BURIDANT, Antoine CHAFFIN

Novembre 2018

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Présentation du projet</b>	<b>3</b>
2.1	Règles du jeu . . . . .	3
2.2	Création d'une partie . . . . .	3
<b>3</b>	<b>Phase de conception</b>	<b>4</b>
3.1	Diagramme UML . . . . .	4
3.2	Diagrammes de séquences . . . . .	8
3.3	API Rest . . . . .	11
3.4	Maquettes du projet . . . . .	12

# 1 Introduction

Dans le cadre de notre projet de Programmation et Modélisation Orientée Objet (POO / MOO), nous allons implémenter un jeu appelé Diaballik. Ce projet va être l'occasion d'utiliser les différents patrons de conceptions vus en cours.

Ce premier rapport a pour but de présenter les contraintes du projet (règles du jeu, patrons à utiliser, etc) ainsi la façon dont nous prévoyons d'implémenter le jeu (schéma UML, maquettes de l'application).

## 2 Présentation du projet

Pour comprendre comment réaliser ce projet, il est nécessaire de savoir comment jouer au Diaballik. Nous allons donc dans un premier temps énoncer les règles du jeu ainsi que les différentes étapes du lancement d'une partie.

### 2.1 Règles du jeu

Le Diaballik est un jeu de plateau à deux joueurs qui se déroule au tour par tour. Chaque joueur dirige sept pions et une balle et effectue trois actions par tour. Ces actions se limitent à déplacer un pion ou passer la balle à un autre pion. Le plateau du jeu est un damier carré de 7 cases.

Un pion ne peut être déplacé que sur une case voisine et uniquement à gauche, à droite, devant et derrière (pas en diagonale). En revanche, la balle peut être déplacée dans n'importe quelle direction mais doit obligatoirement reposer sur un pion, on ne peut donc déplacer la balle que de pion en pion et si aucun pion ennemi n'est entre les deux.

Au début de la partie, les joueurs disposent leurs pions sur une ligne, chacun à une extrémité du plateau, face à face. Pour gagner une partie, un joueur doit réussir à faire traverser sa balle sur le plateau, c'est à dire à positionner sa balle sur la ligne opposée de celle de départ.

### 2.2 Création d'une partie

L'utilisateur peut soit lancer une nouvelle partie soit charger une partie sauvegardée pour la continuer. Si l'utilisateur choisit de démarrer une nouvelle partie, il doit choisir entre une partie joueur contre joueur ou joueur vs IA. Il doit ensuite saisir son nom (le nom de son joueur) ainsi que sa couleur. Il n'y a que deux couleurs possibles (jaune et violet), l'utilisateur choisit donc la sienne et, indirectement, celle de son adversaire.

Si l'utilisateur souhaite jouer contre l'ordinateur, il doit alors choisir le niveau de difficulté parmi les suivants : "Noob" (facile), "Advanced" (difficile) et "Progressive" (10 premiers tours en "Noob" puis passe en "Advanced").

L'utilisateur doit ensuite sélectionner un scénario de départ. Il peut choisir le scénario "Standard" (les pions d'un même joueur sont tous disposés sur la même ligne et la balle est positionnée sur le pion central), le scénario "Aléatoire" (standard mais la balle est positionnée sur un pion au hasard) ou bien le scénario "Enemy Among Us" (les pions des deux joueurs sont mélangés sur les

deux lignes de départ).

Finalement, une fois ces paramètres sélectionnés, le plateau est affiché, les pièces et balle des deux joueurs sont positionnées et la partie peut commencer.

### 3 Phase de conception

Après avoir découvert le jeu, ses règles et ses contraintes, nous avons pu démarrer la phase de conception. Cette étape consiste à déterminer la manière avec laquelle nous allons implémenter le jeu.

#### 3.1 Diagramme UML

La première étape de cette phase de conception a été la réalisation d'un diagramme UML qui schématise l'ensemble du projet, ses composants et leurs interactions.

Dans un premier temps, nous avons construit la classe Game qui correspond au jeu en lui-même. Un jeu est composé d'un plateau et de deux joueurs (nous avons opté pour une liste de joueurs).

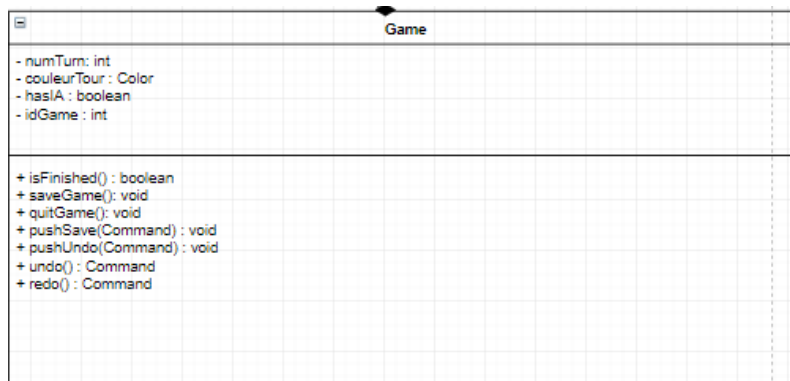


FIGURE 1 – Classe Game

La méthode `isFinished()` permet de vérifier après chaque coup que la partie n'est pas finie et, si elle l'est, de savoir quel joueur a gagné.

La classe Game possède deux listes de commandes : une liste "Save" pour sauvegarder les actions de la partie et une liste "Undo" pour sauvegarder les actions annulées et les rejouer dans le cas d'un redo.

Une partie est créée grâce à un builder, que nous avons nommé GameBuilder. Ce builder va servir à créer la partie en fonction du type de partie que l'utilisateur souhaite jouer : humain contre humain, humain contre IA ou bien charger une partie sauvegardée.

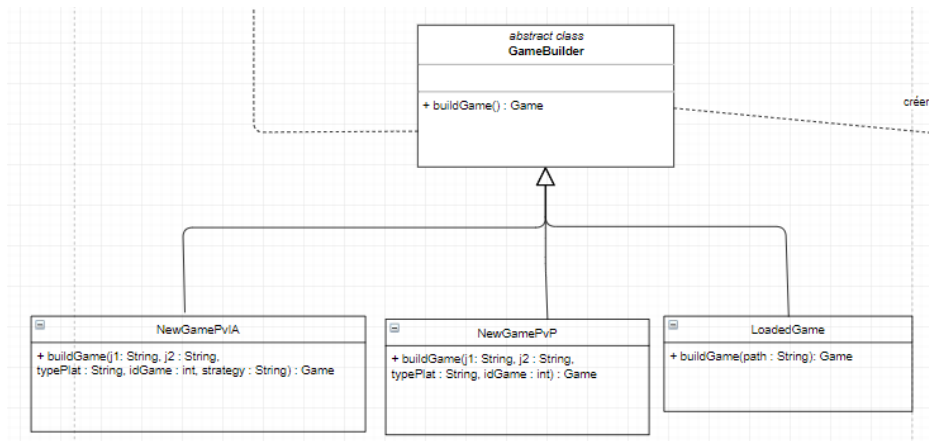


FIGURE 2 – Builder d’une partie

Un joueur est modélisé par l’interface Player et possède un nom et une couleur qui l’identifie. Les classes ‘Humain’ et ‘IA’ implémentent cette interface car un joueur peut soit être un humain soit une IA (l’ordinateur). Nous avons utilisé le patron de conception Strategy pour modéliser les trois niveaux de difficulté de l’IA.

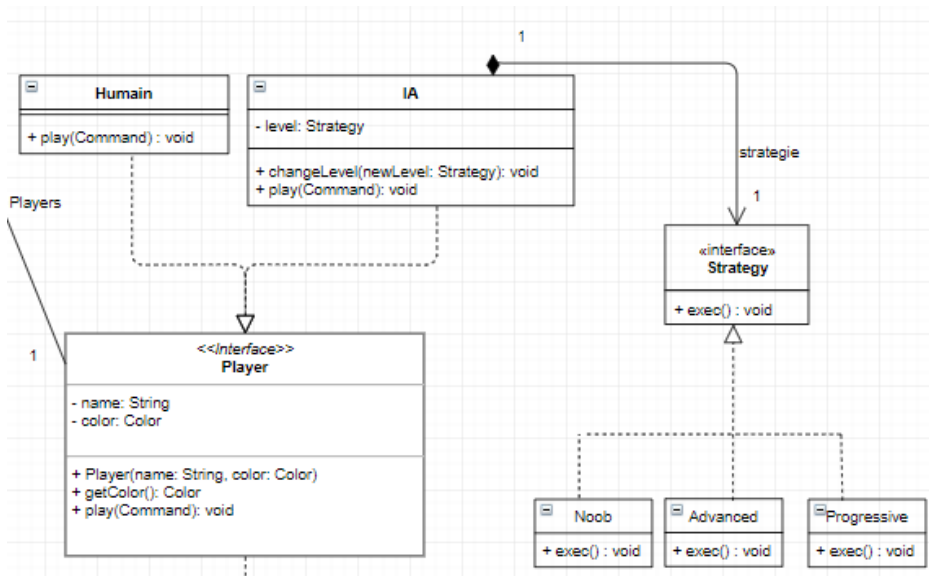


FIGURE 3 – Interface Player et Strategy

Le jeu possède deux types de pièces (ou éléments) : les pions et les balles. Un pion est identifiable par son id et peut posséder une balle. Une balle doit obligatoirement être sur un pion. Les pions et la balle d’un joueur ont la même couleur que le joueur.

La méthode movePlayable() d’un pion permet de rendre la liste des commandes réalisables par ce pion à un instant t pour l’afficher au joueur, une commande pouvant représenter un déplacement du pion ou une passe de la balle. Cette liste sera vide si la couleur du pion n’est pas la couleur du tour en cours.

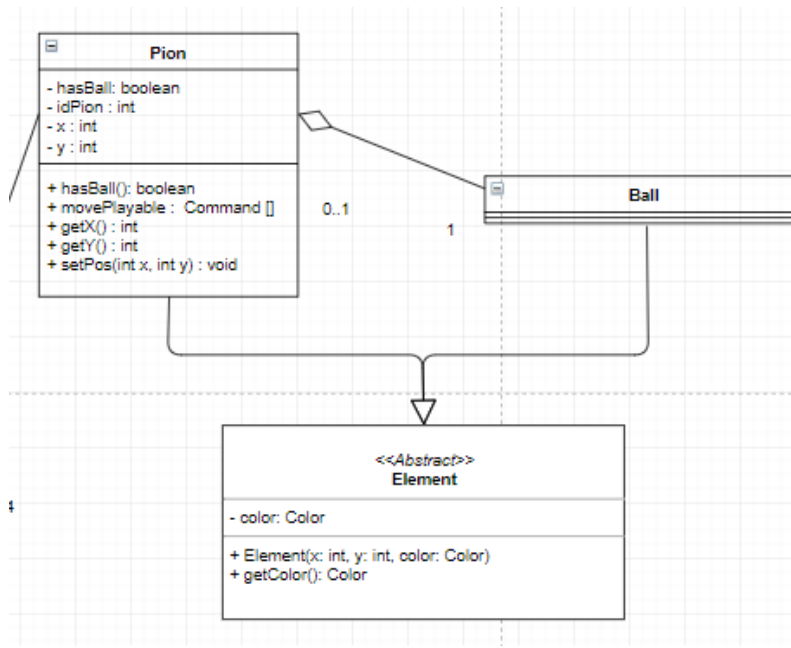


FIGURE 4 – Pions et balle d’un joueur

Le plateau est créé à l’aide d’un builder afin de pouvoir modifier seulement le builder utilisé pour choisir le type de partie voulue c’est à dire le scénario de départ (Standard, EnemyAmongUs ou Random).

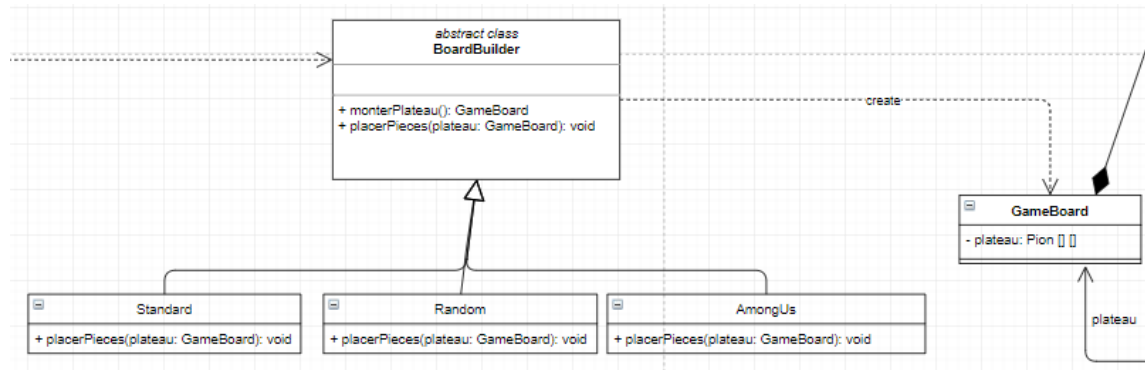


FIGURE 5 – Classe Board et son Builder

Seules deux actions sont possible lors d'un tour : déplacer un pion ou déplacer la balle. Ces commandes sont exécutables par le joueur afin de jouer le coup et sont ajoutées dans la liste save de la classe Game. De cette façon, on garde un historique des actions, qui sont annulables et permettront de sauvegarder et charger une partie. En effet, en conservant les paramètres de création de la partie (noms des joueurs, type de plateau, IA ou non), on peut la récréer puis rejouer toutes les commandes afin de retrouver l'état de la partie au moment de la sauvegarde.

La fonction save permet de créer un fichier Json qui stock les informations nécessaires pour continuer la partie ou la visionner.

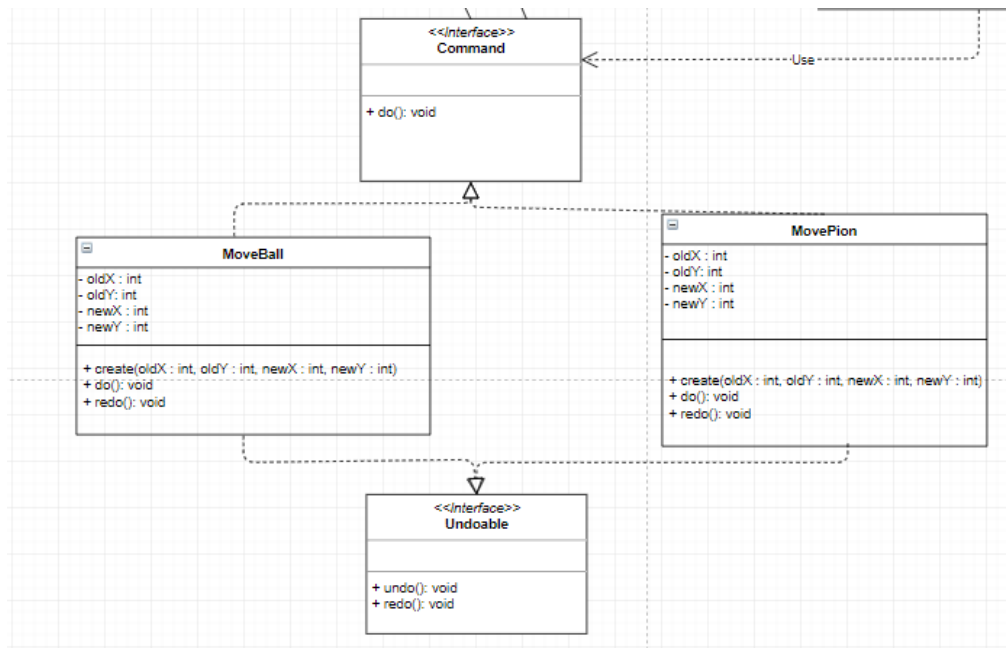


FIGURE 6 – Classe Command

## 3.2 Diagrammes de séquences

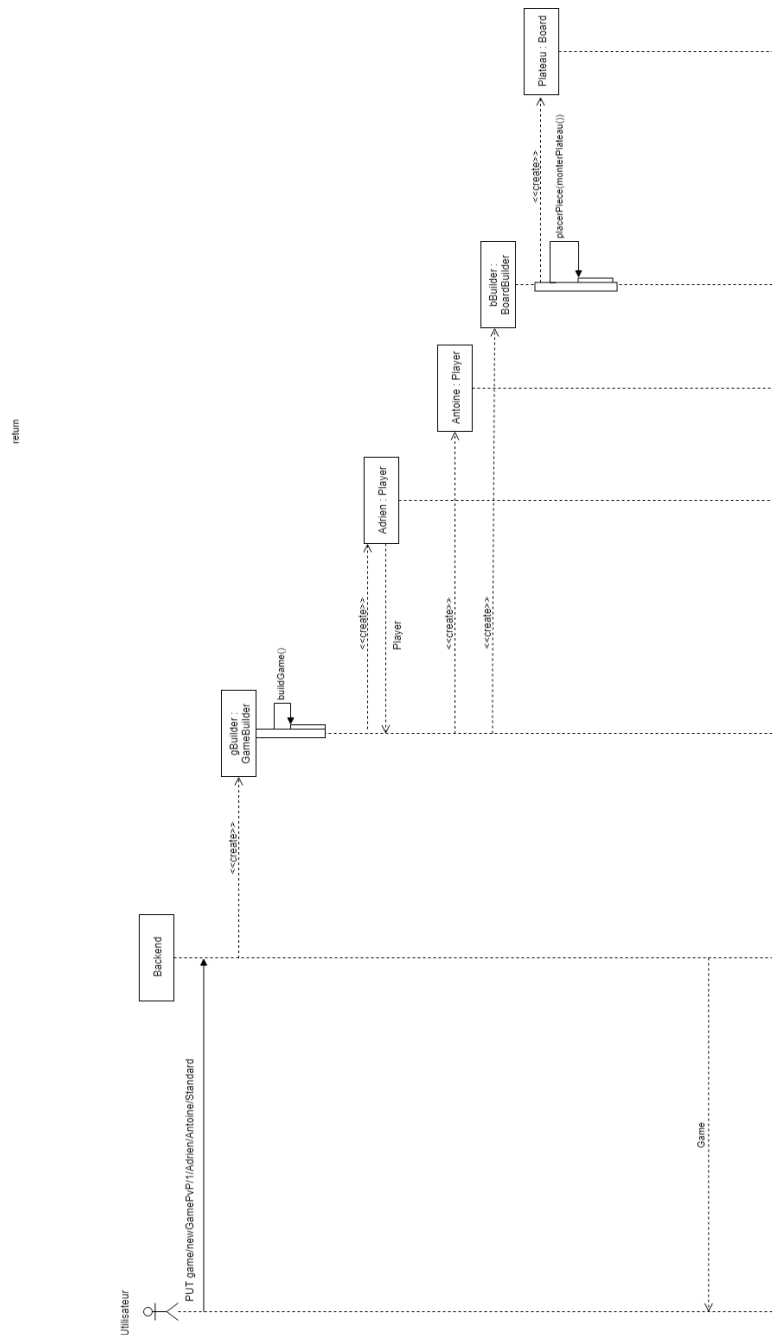


FIGURE 7 – Création d'une partie

Lorsque l'utilisateur souhaite créer une partie, c'est le GameBuilder qui va construire la partie c'est à dire créer les joueurs et construire le plateau via le BoardBuilder.



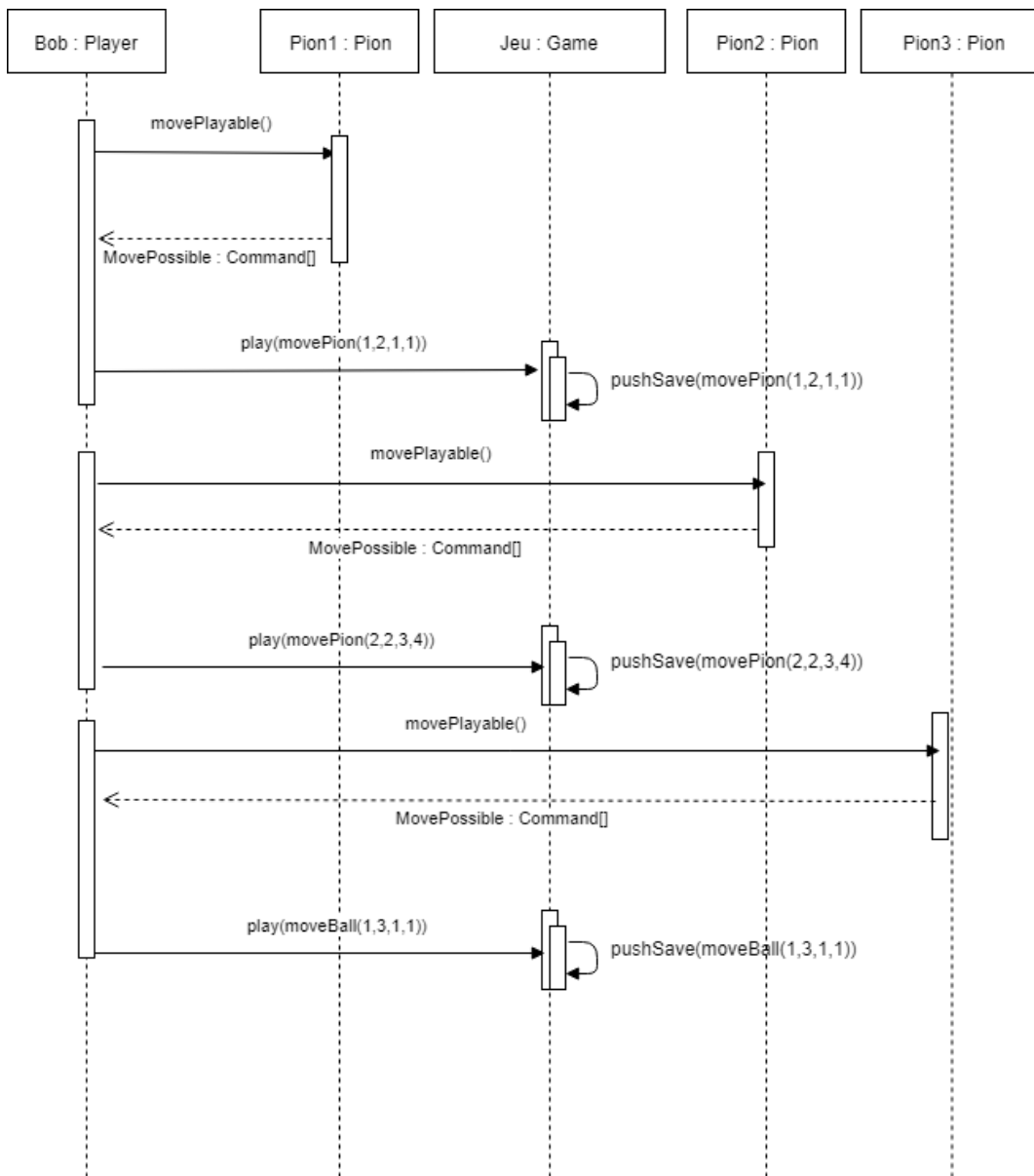


FIGURE 8 – Déplacement d'un pion et de la balle

Lorsqu'un joueur souhaite faire une action lors de son tour, comme déplacer un pion par exemple, il peut demander au pion les différents mouvements possibles à réaliser à partir de ce pion. Ainsi, le pion renvoie au joueur les différentes actions réalisables et le joueur peut choisir d'en réaliser une. Chaque action est ensuite sauvegardée via `pushSave()` pour être visionnée.

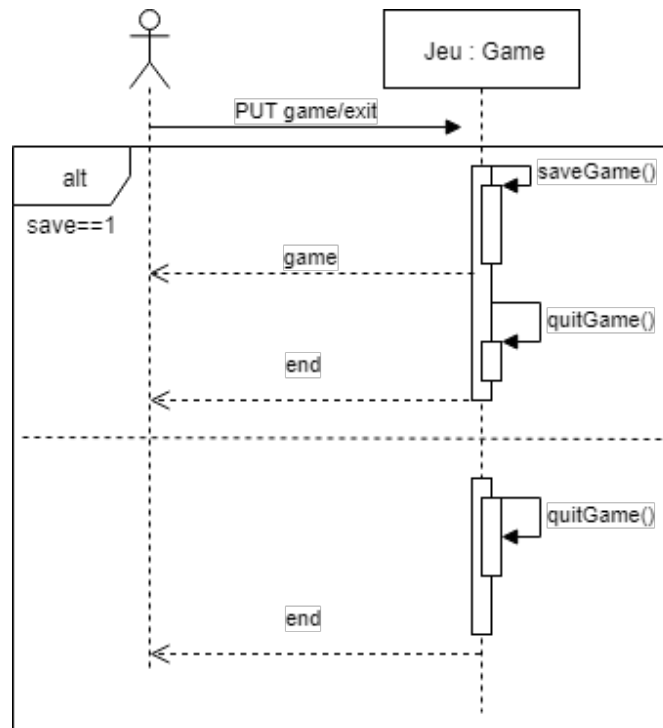


FIGURE 9 – Sauvegarder et quitter une partie

Lorsque l'utilisateur souhaite quitter une partie en cours, il peut soit la sauvegarder puis quitter la partie, soit quitter la partie sans la sauvegarder.

Pour sauvegarder la partie, c'est la méthode `saveGame()` de la classe `Game` qui est exécutée. Toutes les informations nécessaires à la sauvegarde sont stockées dans un fichier Json. Pour quitter la partie, c'est la méthode `quitGame()` qui est exécutée.

### 3.3 API Rest

Les API Rest sont les commandes que l'utilisateur va, sans le savoir, envoyer au serveur du jeu pour pouvoir créer une partie, la sauvegarder, déplacer des pions ou la balle etc. Ces requêtes vont servir à faire le pont entre le Front End et le Back End.

1. Création d'une partie PvP :  
PUT game/newGamePvP/{idGame}/{nom1}/{nom2}/{typeplateau}
2. Création d'une partie PvIA :  
PUT game/newGamePvIA/{idGame}/{nom1}/{typeplateau}/{stratégieIA}/{nomIA}
3. Commande MoveBall :  
PUT game/moveball/{x1}/{x2}/{y1}/{y2}
4. Commande MovePiece :  
PUT game/movepiece/{x1}/{x2}/{y1}/{y2}
5. Récupération Plateau  
GET game/board
6. Sauvegarde d'une partie  
GET game
7. Visualisation d'une partie  
POST game/replay/{id}  
POST game/replay/forward  
POST game/replay/backward
8. Reprendre une partie  
POST/game/{id}
9. Supprimer une partie :  
DELETE game/{id}
10. Quitter la partie :  
PUT game/exit

### 3.4 Maquettes du projet

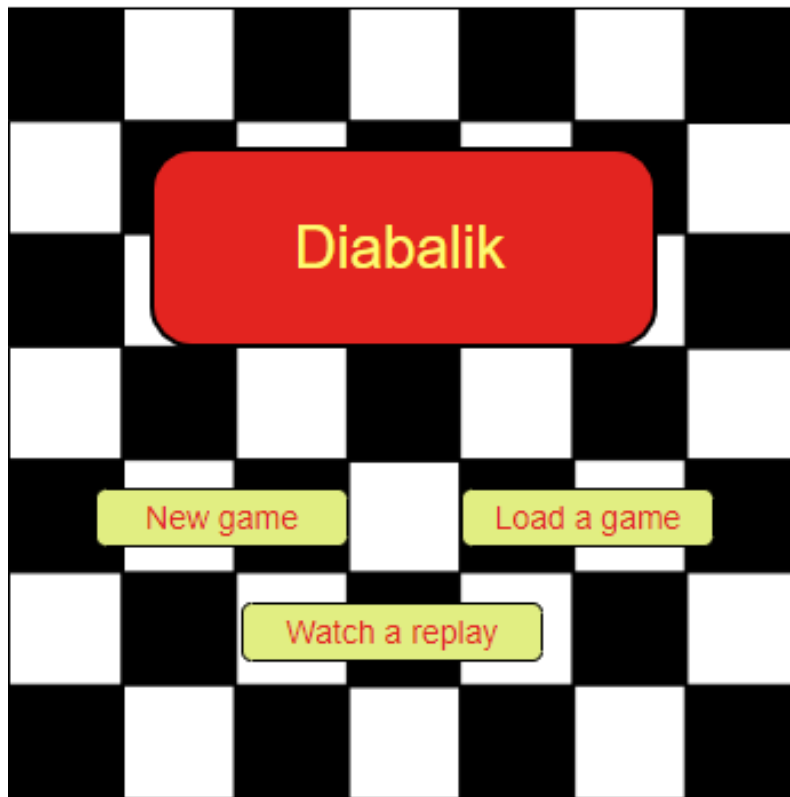


FIGURE 10 – Ecran d'accueil

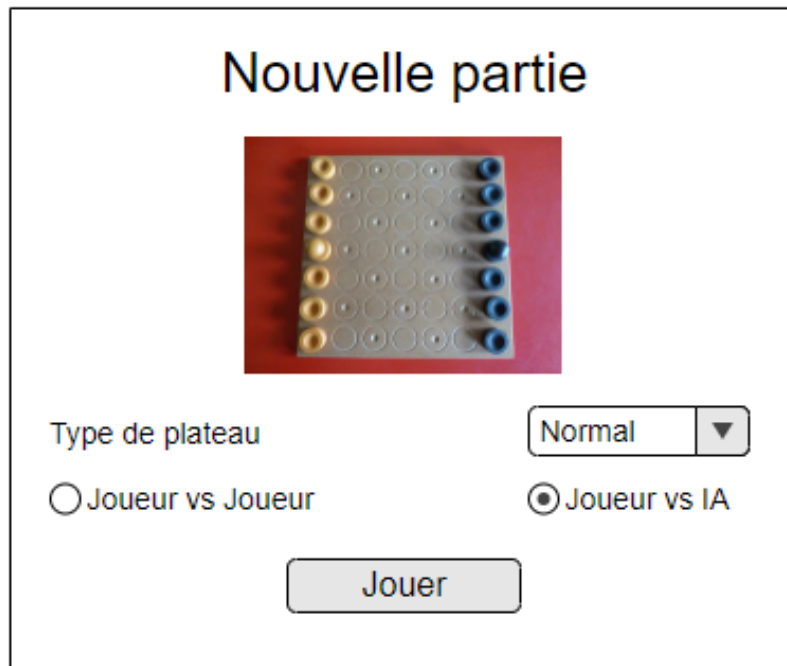


FIGURE 11 – Création d'une nouvelle partie

## Player 1

Nom :


Couleur : 

FIGURE 12 – Création d'un joueur

## Intelligence artificielle

Nom de l'IA

Level de l'IA

▼

FIGURE 13 – Création d'une IA



FIGURE 14 – Interface du jeu