INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Ronan BEBIN
Robin GUILL

Notes de conception
Diaballik

Sommaire

A. Diagramme de classe

Le diagramme de classe est disponible à la fin de la partie A ainsi qu'en pdf et en fichier .asta dans l'archive.

A.1. Choix d'implémentation

Nous allons ici justifier nos choix d'implémentation et mettre en avant les patrons de conception utilisés.

A.1.1 Création d'une partie

Il existe deux constructeurs à la classe Game afin de pouvoir créer une partie Player vs. Player et une partie Player vs. AI. La classe Game est le point central de notre implémentation.

A.1.2. Joueurs

Il existe deux types de joueurs, les joueurs humains et les joueurs gérés par une intelligence artificielle. Nous avons donc décidé d'utiliser une classe abstraite « Player » car les deux types sont très ressemblants, la seule différence étant que les joueurs IA possèdent un algorithme appelé stratégie.

Il existe trois algorithmes, représentant les trois niveaux d'IA, qui s'exécute de la même manière. Nous avons donc utilisé le patron de conception « **Strategy** ».

A.1.3 Plateau de jeu

La classe Board définit le plateau et sa taille, composé de 49 Tiles. Elle permet d'associer les pieces au tiles avec les fonction add() et move(), ainsi nous savons ou est chaque pièce.

A.1.4 Pièces et balles

Chaque joueur possède une balle qui est placée sur une des pièces du joueur. Une méthode hasBall() permettra de savoir depuis la pièce si une balle est présente sur celle-ci ou non afin de savoir si elle peut se déplacer ou non.

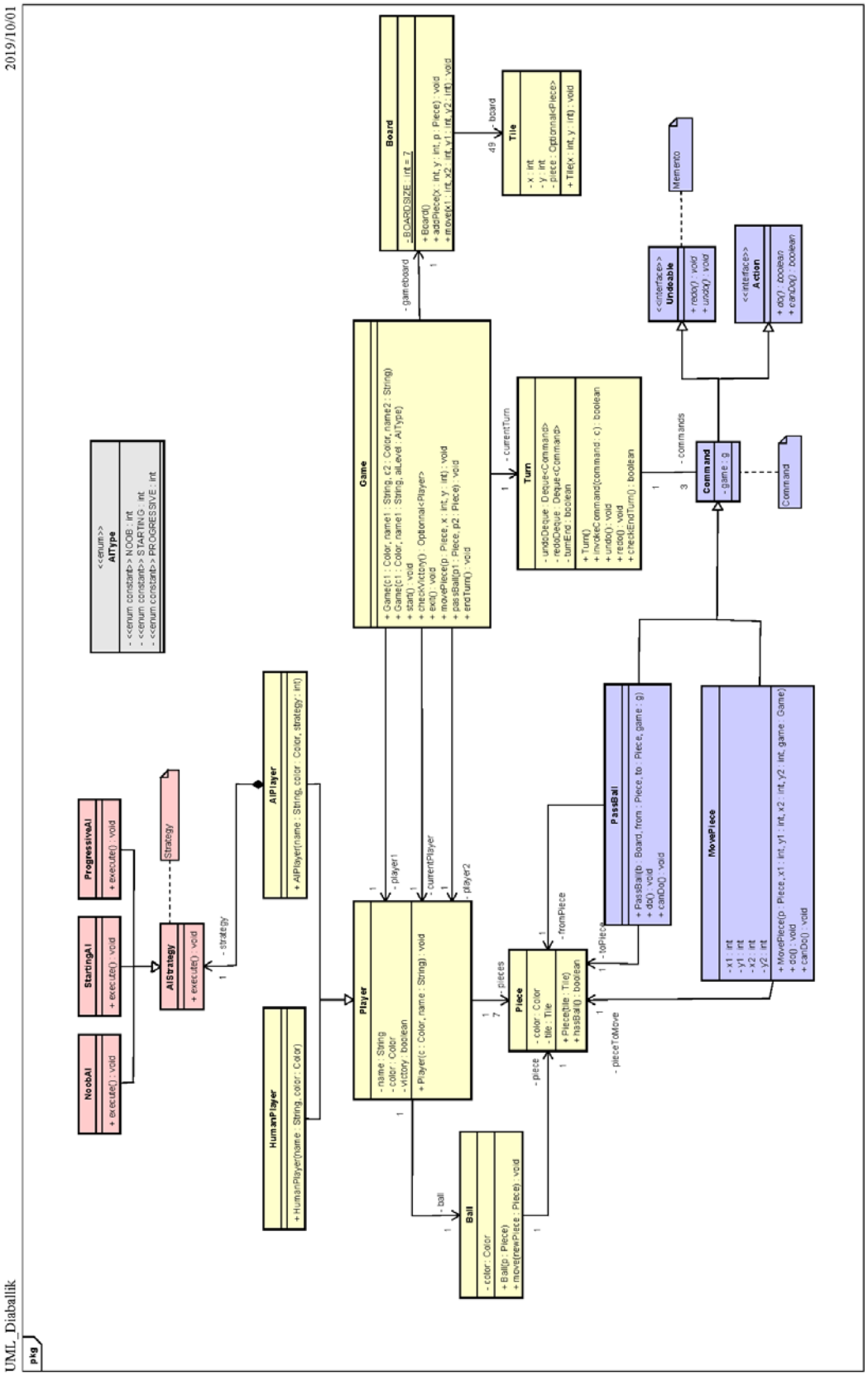
A.1.5 Passe et déplacements

Deux actions étant possible, nous utilisons une classe abstraite Command, héritée par MovePiece et PassBall. Notre classe abstraite implémente l'interface Action et Undoable et nos classes concrètes utilisent le patron de conception "Commande". Ainsi nous stockons les objets « Command » dans notre classe Turn. Chaque Turn possède les 3 actions effectuées par un joueur pendant son tour.

A.1.6 Fin de partie

Le joueur ne pouvant gagner que par une passe jusqu'au bord du terrain adverse, nous vérifions pour chaque passe si la balle à atteint le fond du terrain adverse avec la fonction « checkVictory() ». Elle rend un objet Optionnal<Player> et nous stoppons la partie si il n'est pas vide.

A.2 Diagramme de Classe



1 / 1

B. API REST

Voici les requête REST de modification/création de ressources, lors de l'envoi de ces requêtes au backend il renvoie un objet game (modifié ou non) à chaque fois.

Si les requêtes sont interdites par les règles du jeu, le game est rendu non-modifié par le backend.

Créer une nouvelle partie

Création d'un nouveau jeu avec 2 humains :

POST /game/newPvP/{name1}/{color1}/{name2}/{color2}

name1 (String) : Nom du 1er joueur
color1 (int) : Couleur du 1er joueur (0 : Noir ; 1 : Blanc)
name2 (String) : Nom du 2ème joueur
color2 (int) : Couleur du 2ème joueur (0 : Noir ; 1 : Blanc)

Création d'un nouveau jeu avec 1 humain 1 IA :

POST /game/newPvAI/{name}/{color}/{strategy}

name (String) : Nom du joueur
color (int) : Couleur du joueur (0 : Noir ; 1 : Blanc)
strategy (int) : Niveau de l'IA choisi (0 : NoobAI ; 1 : StartingAI ; 2 : ProgressiveAI)

Action et Commands dans une partie

Faire une passe

PUT /game/action/{playerID}/passBall/{pieceID1}/{pieceID2}

playerID (int) : Identifiant du joueur voulant effectuer le déplacement
pieceID1 (int) : Pièce lanceuse
pieceID2 (int) : Pièce receveuse

Bouger une pièce

PUT /game/action/{playerID}/movePiece/{pieceID}/{x}/{y}

playerID (int) : Identifiant du joueur voulant effectuer le déplacement
pieceID (int) : Pièce à déplacer
x (int) : Nouvelle position selon l'axe des abscisses
y (int) : Nouvelle Position selon l'axe des ordonnées

Retour arrière

PUT /game/action/undo

Rétablir commande

PUT /game/action/redo

Fin de tour

PUT /game/action/endTurn

Commandes générales

Sortir du jeu
DELETE /game

Commandes GET

Get current player
GET /game/player/current

Get ball (id)
GET /game/ball/{id}

id (int) : Identifiant de la balle à récupérer

Get ball (player)
GET /game/ball/{player}

player (int) : Identifiant du joueur dont on veut récupérer la balle

Get piece (id)
GET /game/piece/{id}

id (int) : Identifiant de la pièce à récupérer

Get pieces (player)
GET /game/piece/{player}

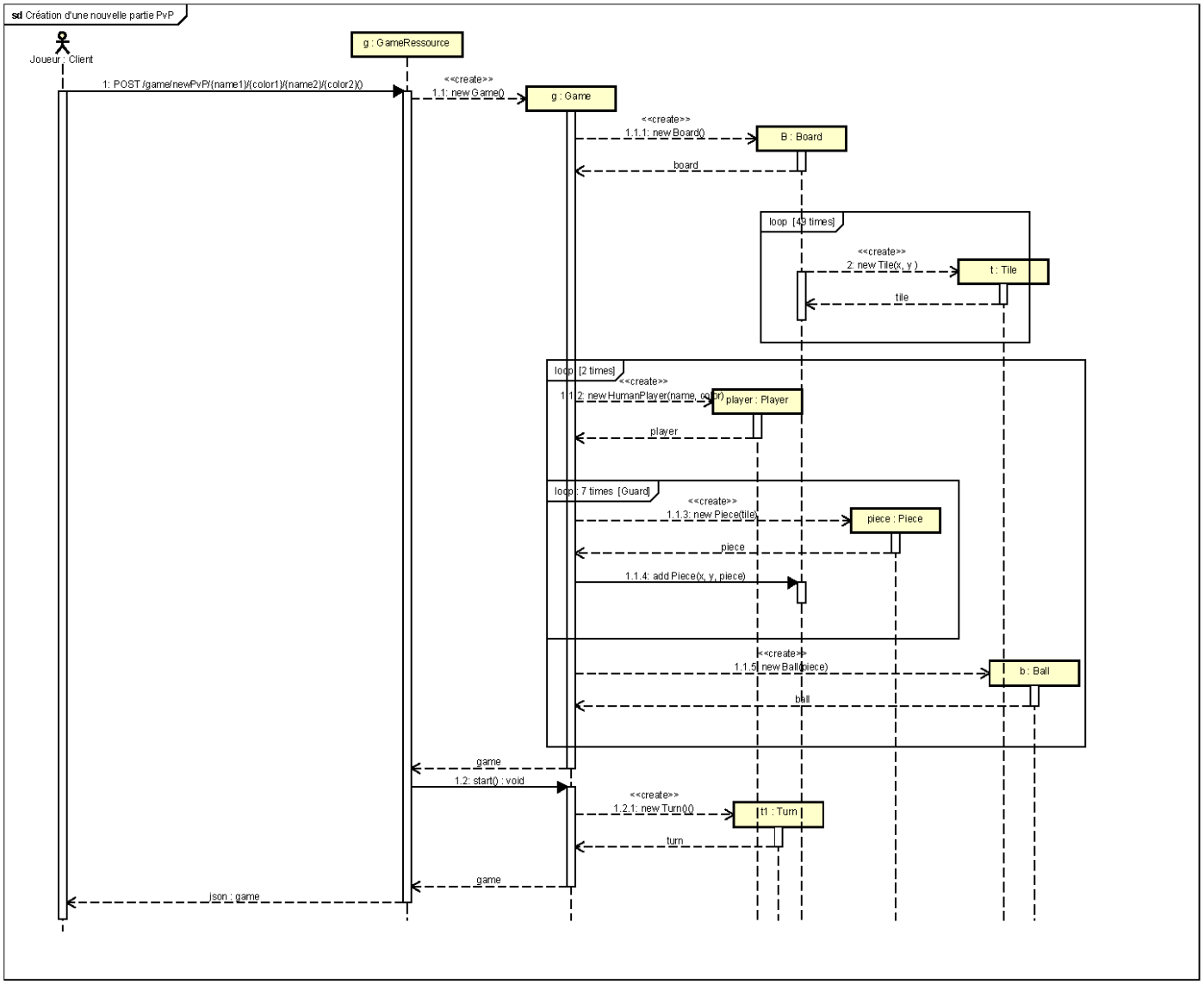
player (int) : Identifiant du joueur dont on veut récupérer toutes les pièces

Get game
GET /game

C. Diagrammes de séquences

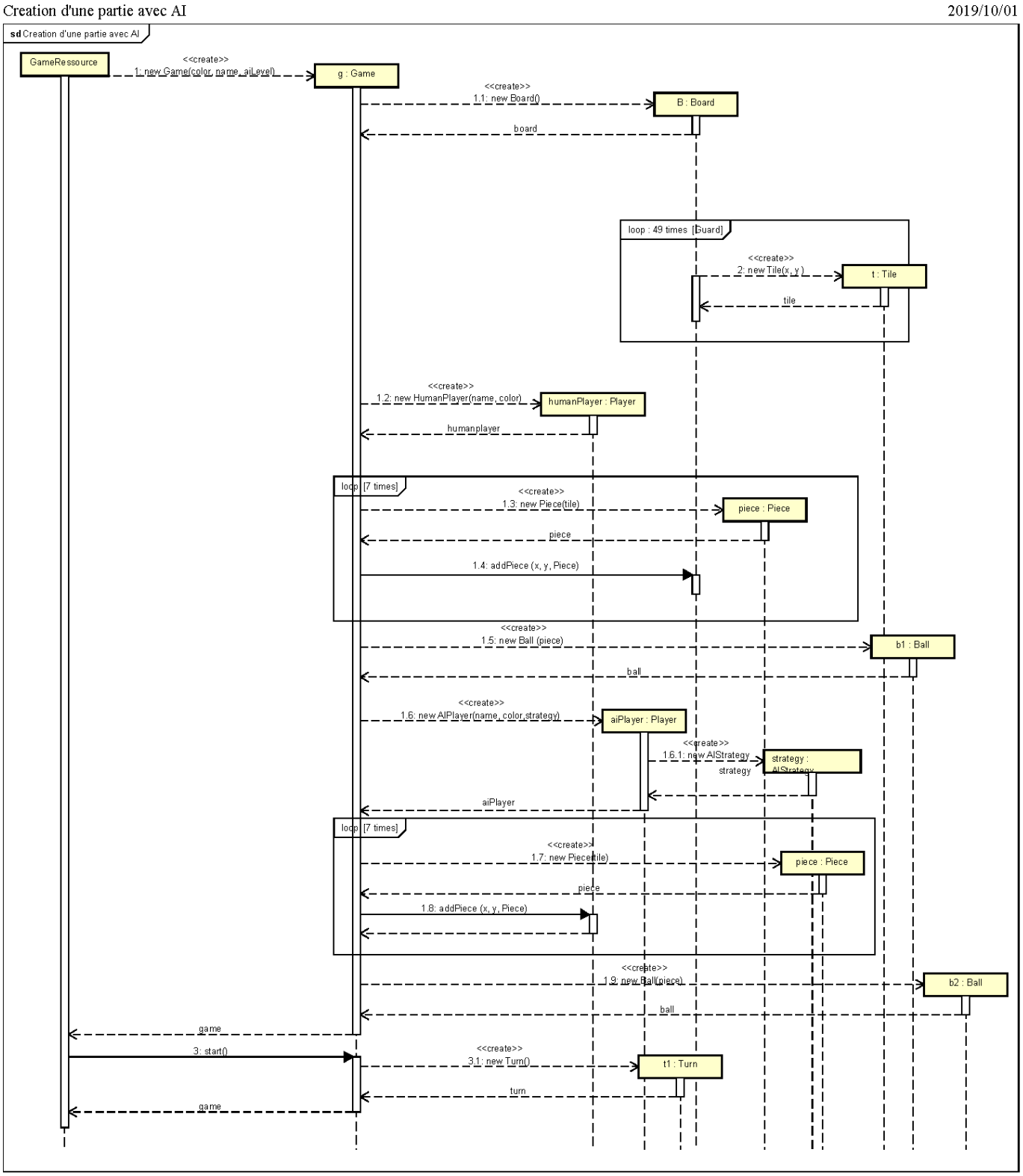
C.1. Création d'une partie PvP

Création d'une nouvelle partie PvP 2019/10/01



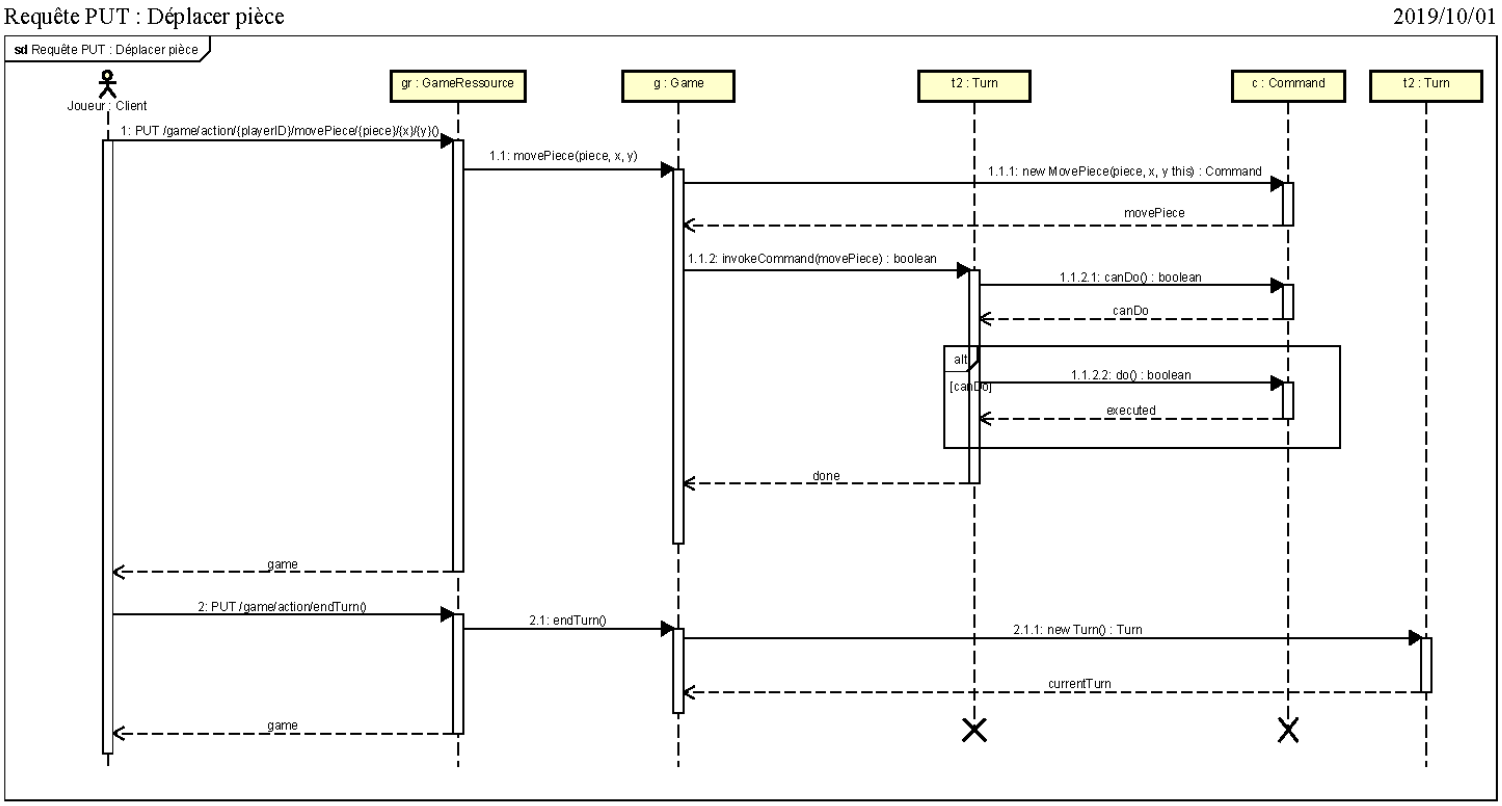
Ici nous explicitons la création d'une partie Joueur contre joueur avec une requête POST venant du client.

C.2. Création d'une partie contre une IA



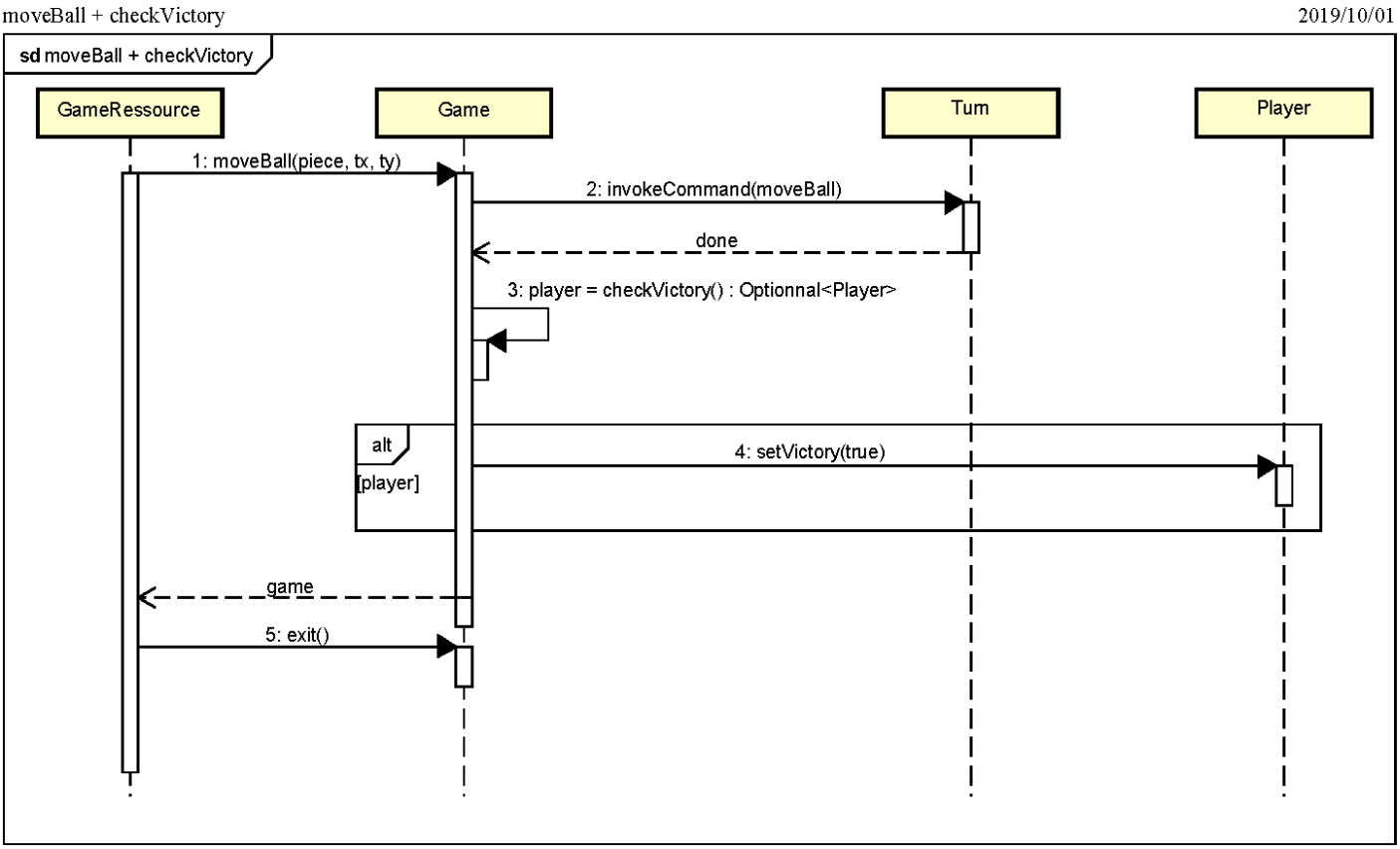
Il nous semblait intéressant de voir comment se créait une partie contre une intelligence artificielle avec la création d'une AIStrategy.

C.3. Requête PUT – Déplacer une pièce



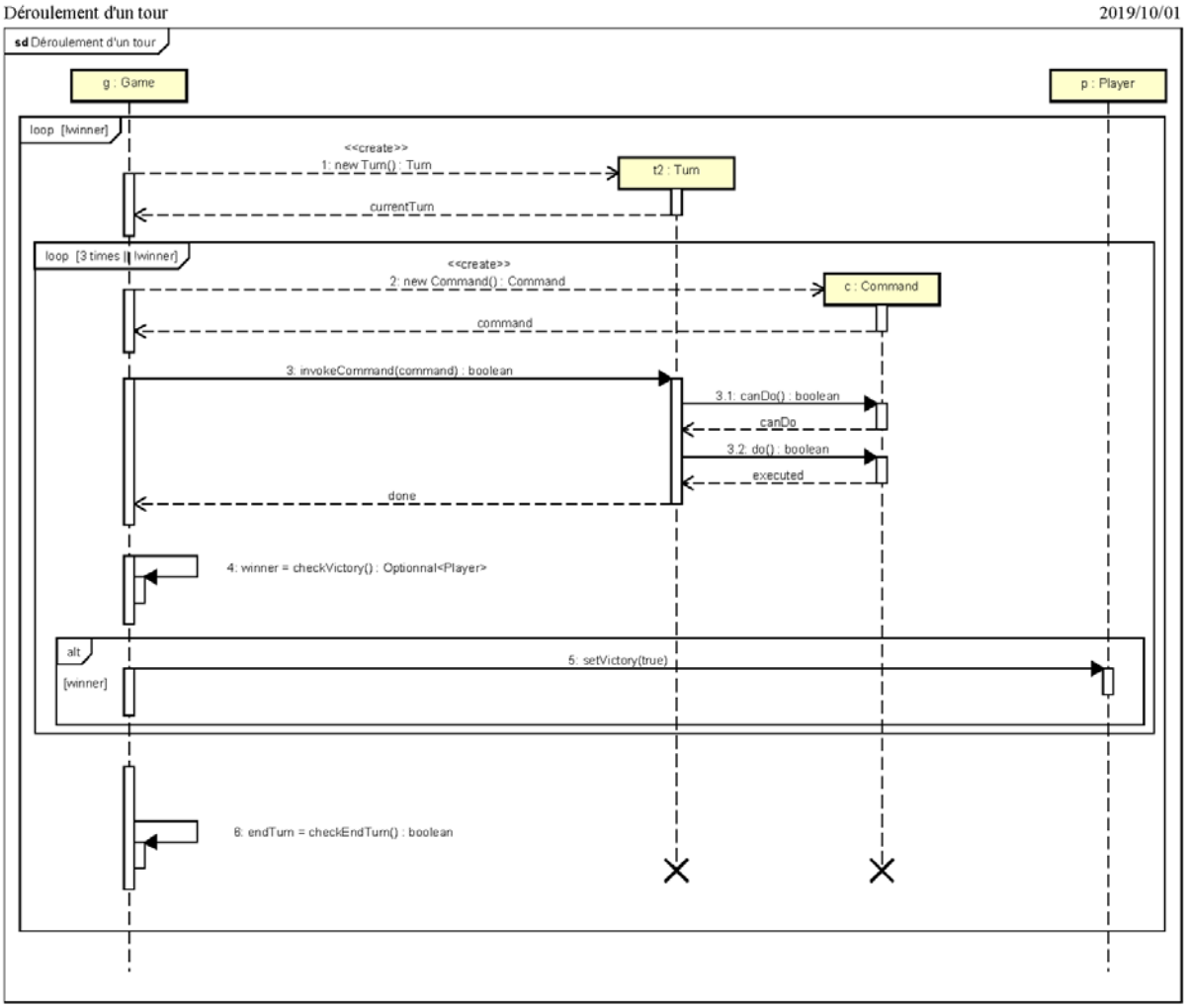
Nous avons décidé de modéliser une requête de déplacement de pièce avec la gestion ainsi que l'utilisation d'une "Command" avec les fonction do() et canDo() hérité de la classe Action.

C.4. moveBall + checkVictory



Ici, une commande moveBall avec la vérification de victoire.

C.5. Déroulement d'un tour



Pour avoir une idée plus générale de comment va être géré le jeu nous avons ici modélisé le déroulement complet d'un tour avec la vérification de victoire, et l'arrêt de la partie en cas de victoire.

En réalité le checkVictory ne se fera qu'après une passe, comme ici nous avons généralisé en utilisant la classe abstraite « Command », le checkVictory est fait après chaque « Command ».

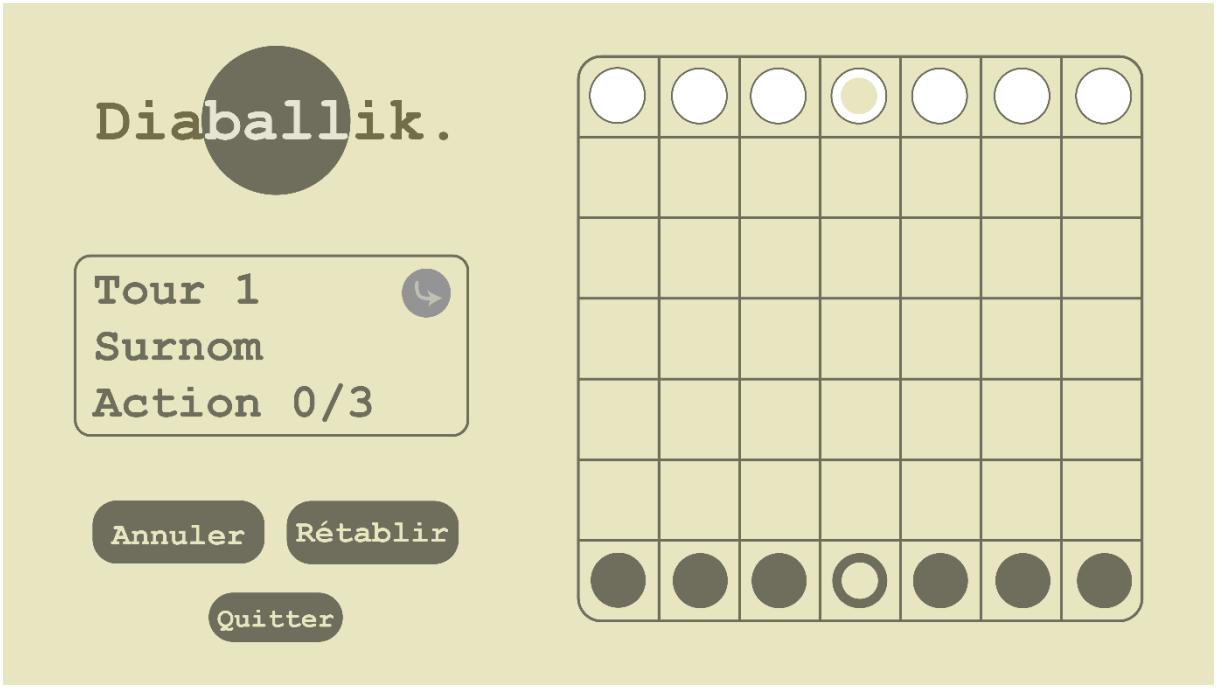
D. Conception graphique

Nous avons décidé d'utiliser une charte graphique simple et efficace avec des couleurs sobres.

D.1. Accueil

D.2. Page de personnalisation

D.3. Partie



D.4. Page de fin de partie