Projet APO Tic-tac-toe 3D

Boissot Aurélien, Laqueuvre Damien, Moulon Florent

janvier 2023

Table des matières

1 Structure de l'archive

- Les classes .java sont dans le dossier */projet_tictactoe/src/main/java/apo/boissot_laqueuvre_moulon*
- La documentation est dans le dossier */Javadoc*
- Les diagrammes UML sont dans le dossier */Rapport/Diagramme UML*

2 Méthodologie

Après avoir pris connaissance du sujet, nous avons décidé de mettre en commun nos idées avant de commencer à programmer.

2.1 Choix des outils

La première étape a été de décider quels outils nous allions utiliser pour la réalisation du projet.

Nous avons rapidement choisi d'utiliser les outils qui nous étaient familiers, à savoir :

- Visual Studio Code *comme IDE*
- Java 8 *comme version de java car c'est la version que nous connaissons le mieux et que nous n'avons pas besoin des nouveautés introduits par les versions postérieures*
- GitHub *pour le travail collaboratif et le versioning*
- StarUML *pour créer les diagrammes UML*

De plus, il nous a semblé intéressant d'essayer un outil de gestion de chaîne de compilation. Notre choix s'est porté sur Maven.

2.2 Choix de l'architecture

Après s'être mis d'accord sur notre espace de travail, nous avons échangé autour des grandes lignes de notre application.

Pour cela, nous avons réalisé un premier diagramme de classe sommaire. C'est à ce moment que nous avons décidé qui les Grille2D et Grille3D seraient remprésentées par 2 classse héritant d'une même classe abstraite Grille.
Nous avons décidé de séparer le lancement du jeu et la partie en elle-même dans 2 classes différentes.
Enfin, nous avons choisi de représenter les joueurs (humain ou IA) par une classe.

Nous avons également choisi dès le départ les extensions que nous souhaitions implémenter afin d'adapter la conception en conséquence, notamment pour les grilles de taille variable.

2.3 Déroulement temporel

Une fois que les bases du projet furent établies, nous nous sommes lancés dans la programmation.

La première étape fut de créer le squelette de la classe Grille et Grille2D, afin d'inplémenter toutes les fonctions élémentaires des grilles. Un test régression a également été créé dans la foulée pour tester les fonctions et s'assurer qu'elle resteront correctes tout au long du projet.
Pour éviter d'avoir à repenser toutes ces fonctions plus tard, nous avons décidé de directement considérer nos grilles comme étant de taille variable.

Une fois ce squelette en place, nous avons créé les classes relatives au coeur du jeu et en parallèle nous avons adapté les fonctions de Grille2D dans Grille3D.
Le principal défi fut la gestion des inputs utilisateur. Entre la vérification des données et la gestion des Scanners java, cette étape fut plus longue que prévu.
Nous avons aussi retravaillé l'affichage pour qu'il soit plus agréable avec les grilles de grande taille.
A la fin de ces étapes, nous avions un jeu fonctionnel où 2 joueurs humains poouvais s'affronter sur une grille de taille n en 2 dimensions. La version en 3 dimensions fut terminée quelques temps après.

Dès que le jeu de base commençait à fonctionner, nous avons commencé à travailler sur les extensions optionelles que nous avions prévues.
Nous avons commencé par implémenter les plus simples en parrallèle, c'est à dire l'affichage de la combinaison gagnante et la sauvagarde.
Après quoi, nous avons codé le minmax et adapté le déroulement du jeu au mode humain/ordinateur.

Pendant que certains finalissaient les détails, d'autres commençaient le rapport et remettaient au propre les diagrammes UML.

2.4 Répartition du travail

Nous avons répartit au mieu les différentes taches en fonction de nos capacités et de nos envies tout au long du projet.
Pour cela, nous avons commencé par lister et se repartir les besoins immédiats. Puis, quand l'un de nous terminait sa tache, nous faisions un bilan pour supprimmer les taches effectuées, ajouter les nouveaux besoins et choisir sa prochaine action.
Voici un bilan des réalisations de chacun :

2.4.1 Aurélien

- affichage Grille3D
- documentation de ses fonctions
- sauvegarde de la partie
- vérification des input pour la Grille3D
- gestion des Scanner et des inputs utilisateurs
- classe InterfaceJeu

2.4.2 Damien

- diagramme de cas d'utilisation
- vérification des victoires Grille2D
- vérification des victoires Grille3D
- documentation de ses fonctions
- mise en place de MinMax

2.4.3 Florent

- diagramme de classe
- méthodes basiques de Grille2D
- méthodes basiques de Grille3D
- affichage Grille2D
- Vérification des inputs pour la Grille2D
- rapport
- documentation de ses fonctions

3 Extensions optionnelles

3.1 Extensions fonctionnelles

3.1.1 Taille variée **1 point**

Comme expliqué plus tôt, dès la conception nous avons pensé les Grilles comme des tableaux de taille n variable. Il a donc été très simple de prendre un paramètre n pour que le nombre de cases d'une Grille2D soit n^2 et celle d'une Grille3D, n^3.

3.1.2 Affichage de la combinaison gagnante **1 point**

Lorsqu'un joueur gagne, nous affichons la grille une dernière fois. Les cases de la combinaison gagnante sont encadrés par des chevrons :

```
|>O< X  3 |
|>O< X  6 |
|>O< O  X |
```

3.1.3 Sauvegarde du jeu **1 point**

Lorsque l'on demande au joueur de saisir une case, il peut saisir "save" à la place de son choix de case et cela sauvegardera sa partie dans un fichier.
Lors de sa prochaine partie, un message s'affichera lui demandant s'il souhaite charger sa partie enregistrée. S'il accepte, sa partie précédente sera rechargée et il pourra continuer à jouer sur celle-ci.

Structure de la sauvegarde :

- La première ligne contient toutes les informations caractérisant la partie : le type de grille (2D ou 3D), le types de joueurs (joueur humain ou IA), qui commence la partie, et la taille de la grille.
- Les lignes suivantes contiennent les coup joués par les joueurs avant d'avoir sauvegardé : chaque ligne précise le symbole du joueur ("X" ou "O") et le coup joué.

3.1.4 Intelligence artificielle 2D et 3D **2 points**

En plus du mode humain contre humain, notre programme propose un mode humain contre ordinateur.
Afin de faire une IA capable de jouer, nous avons implémenté l'algorithme minimax pour des parties en 2D et en 3D.

3.2 Extensions techniques

3.2.1 Compilation automatisée — 1 point

Nous avons utilisé Maven pour automatiser la compilation et le lancement de notre programme.
Dans une console, placez vous dans le dossier */projet_tictactoe/*

- Pour compiler le code utiliser les commandes :
  mvn compile
  mvn package
  mvn install
- Pour lancer le programme, utiliser la commande :
  java -cp projet_tictactoe-1.0-SNAPSHOT.jar apo.boissot_laqueuvre_moulon.App

3.2.2 Versioning du code — 1 point

Comme mentionné au début, nous avons utilisé GitHub pour le versioning de notre code.
le répertoire git utilisé est public et accessible ici :
https://github.com/FlorentMoulon/Projet-APO-Tic-tac-toe-3D

3.2.3 Tests unitaires — 1 point

Des tests unitaires vérifiant la bonne exécution des méthodes principales de nos classes on été inmplémentés sous la forme de fonction testRegression().
Chaque classe possède une telle fonction qui effectue et affiche des test de toutes les méthodes. Ces fonctions sont tenues à jour tout au long du développement de l'application pour s'assurer que les nouvelles fonctions ne "cassent" pas les anciennes.

4 Diagrammes UML

4.1 Diagramme de cas d'utilisation

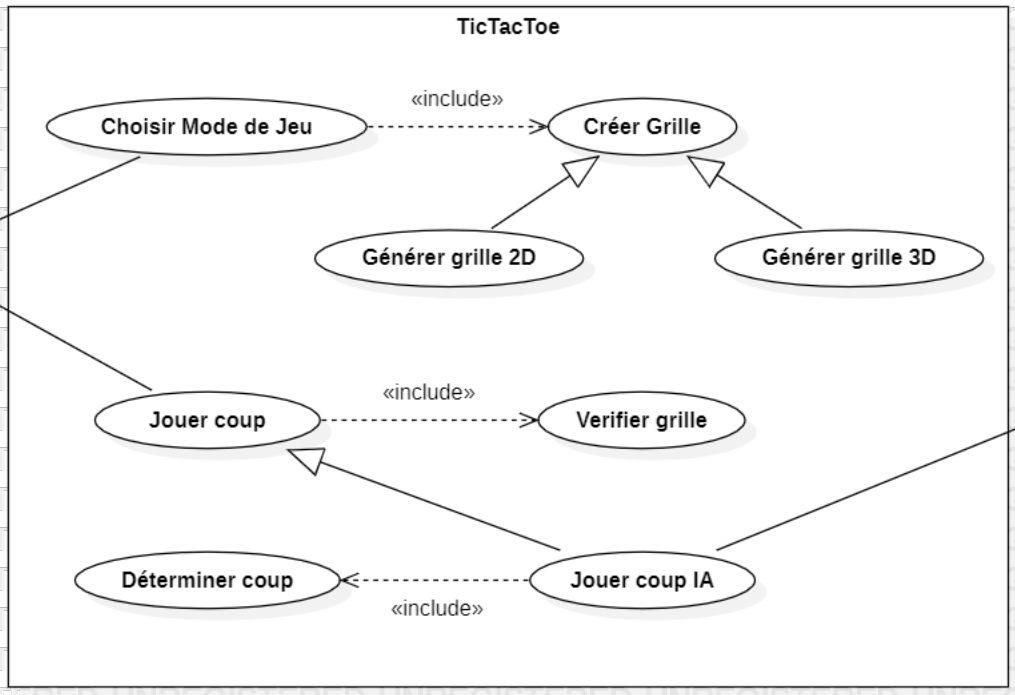

Figure 1: Diagramme de cas d'utilisation

4.2 Diagramme de classe

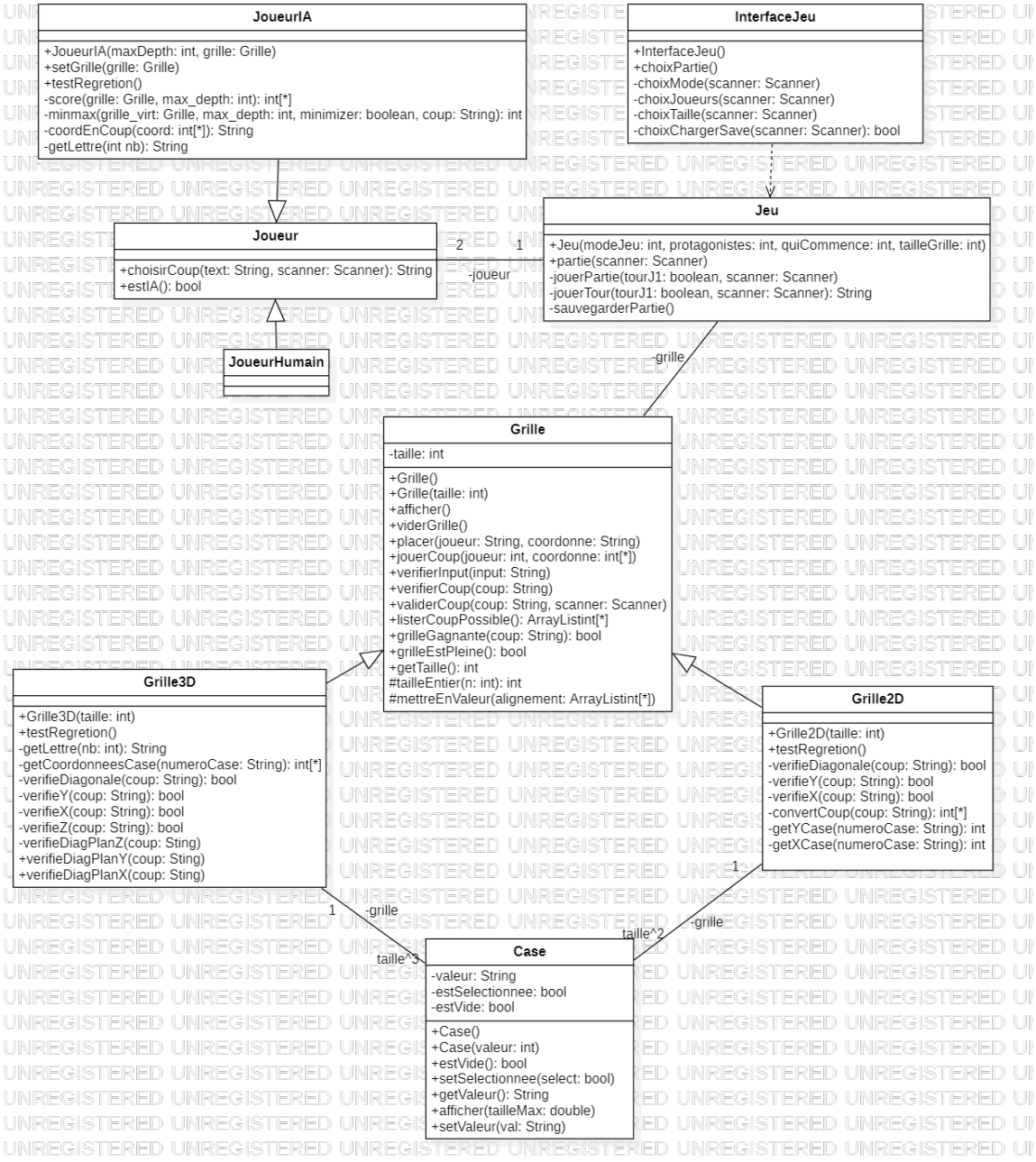

Figure 2: Diagramme de classe

4.3 Diagramme de séquence

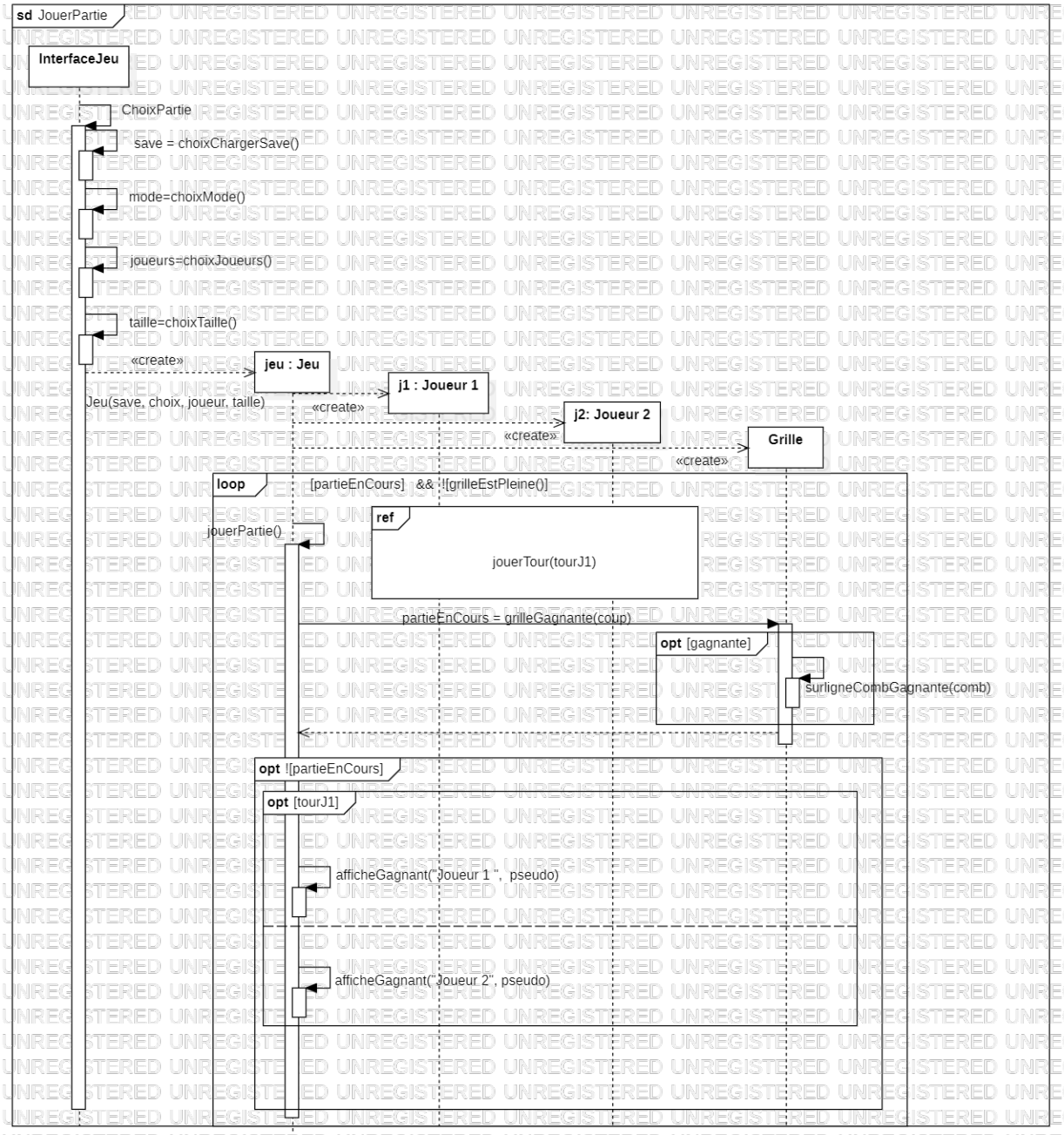

Figure 3: Diagramme de séquence