



Tic Tac Toe - Morpion

APO - 3A Informatique - Polytech Lyon

ABBATE Titouan
LAURENT Clément
RANDRIANTSOA Matthieu

2022 - 2023

Méthodologie et organisation du travail

Le projet à été réalisé en Java 11 à l'aide de l'IDE IntelliJ. Nous nous sommes répartie le travail de façon équitable en fonction des compétences des chacun.

Après réflexion sur la structure de notre projet, Matthieu s'est chargé de concevoir les diagrammes UML permettant le développement du jeu.

Titouan et Clément ont ensuite travaillé ensemble sur le développement du code. Chacun de nous a apporté ses compétences pour mener à bien le projet.

Néanmoins, les diagrammes ont subi quelques ajustement au cours de l'avancement du projet car des fonctionnalités et des contraintes ont vu le jour.

De plus, nous avons créé un dépôt **Github** afin de pouvoir facilement collaborer sur le projet. Dont voici le lien : <https://github.com/Matsew-uwu/TicTacToe3D>.

Nous avons décidé d'implémenter les **extensions** suivantes :

- Taille de grille varié pour les Morpions
- Affichage en couleur
- Affichage de la combinaison gagnante

Par ailleurs, voici les **points techniques** ajoutés au projet :

- Compilation automatisée avec Maven
- Versionnement du code avec Git
- Les tests unitaires avec JUnit 5

Voici un tableau récapitulatif de la répartition des tâches :

Titouan	Clément	Matthieu
Développement de l'interface utilisateur.	Implémentation des méthodes essentielles pour Morpion2D et Morpion3D.	Diagramme UML
Développement des méthodes de Morpion2d et Morpion3d.	Développement algorithme pour l'alignement dans les classes Morpion2D et Morpion3D.	Automatisation et tests unitaires
Ajout de fonction de vérification de fin de partie dans Morpion3d et Morpion2d.		GitMaster et gestion des conflits lors des merges Git.
		Optimisation des méthodes
		Migration du projet sous
		Maven

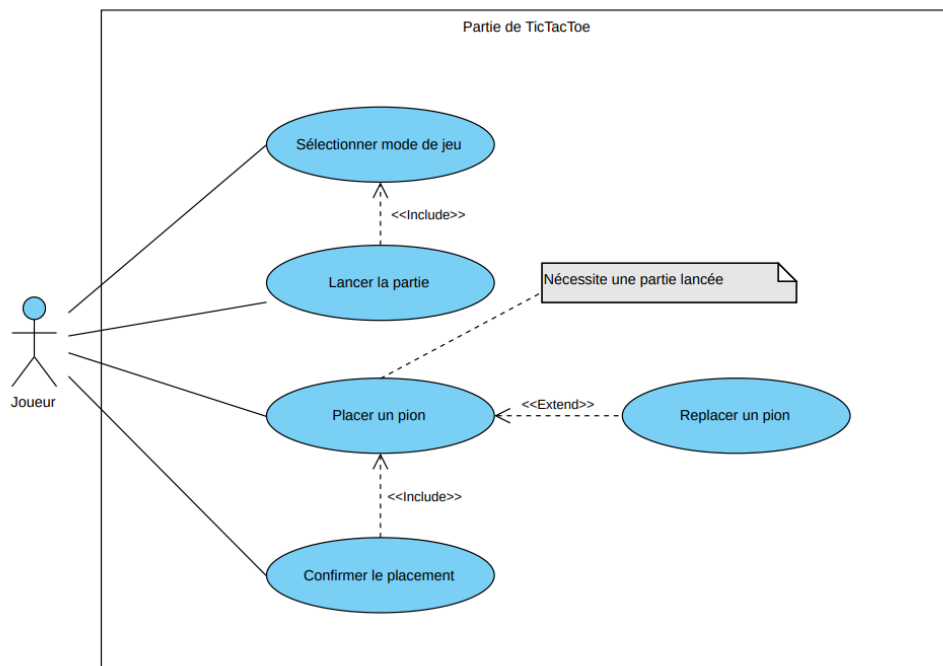
Structure du projet

Voici la structure du projet :

```
Projet/  
├ Documents/ - Diagrammes UML  
├ TicTacToeApp/ - Le répertoire du projet  
│   ├── src/  
│   │   ├── main/ - contient les fichiers java (classe, packages)  
│   │   ├── test/ - contient les classes de tests unitaires  
│   │   ├── pom.xml - fichier de configuration Maven  
│   │   └ TicTacToeApp.iml  
├ README.md  
└ .gitignore
```

Diagrammes UML

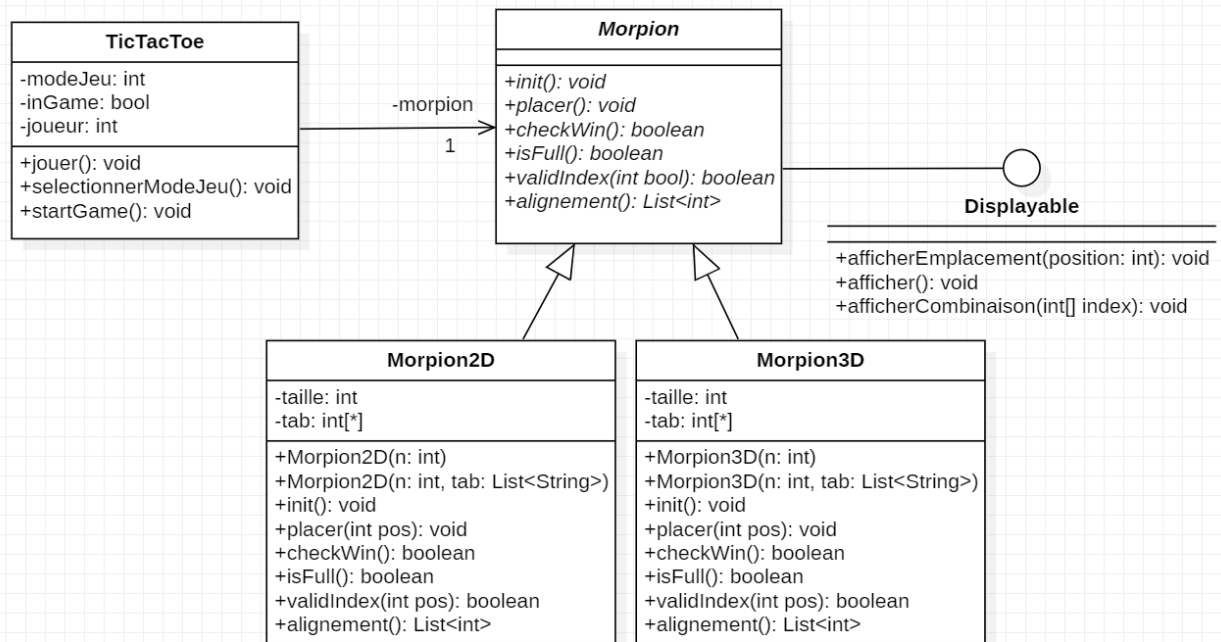
1. Diagramme de CU



L'application doit permettre à un joueur de sélectionner un mode de jeu, et de lancer une partie. Ainsi aura la possibilité de placer un pion sur la grille, de confirmer ce placement, ou de remplacer ce pion.

Les descriptions textuelles se trouvent en annexe.

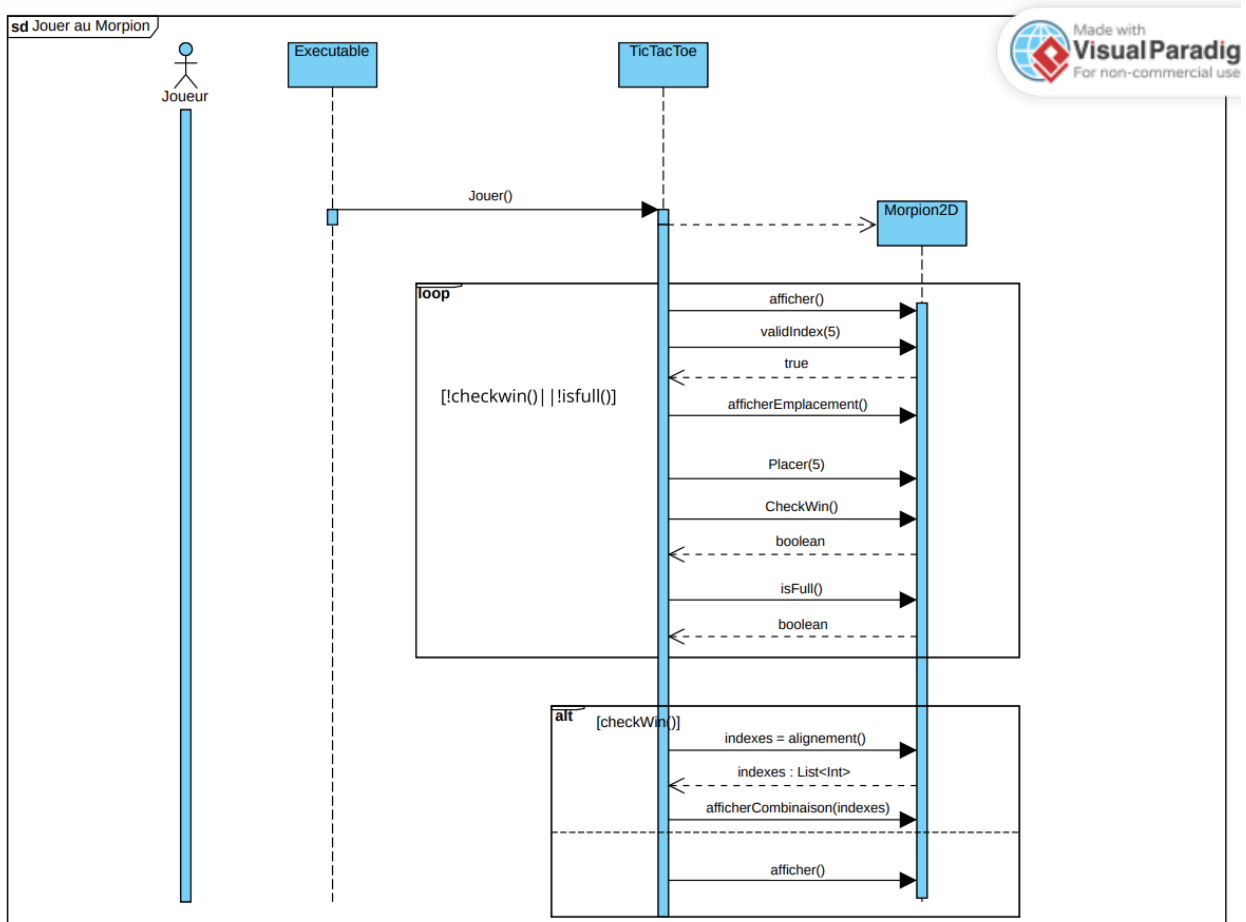
2. Diagramme de classe



Nous avons initialement prévu d'implémenter une interface pour les méthodes d'affichage, mais finalement toutes les fonctions se sont retrouvées dans la classe *Morpion*.

Les classes *Morpion*, *Morpion2D* et *Morpion3D* sont contenues dans un package, ce qui les rend réutilisables dans d'autres projets.

3. Diagramme de séquence



Choix de conception

Nous avons commencé à travailler sous java 17. Cependant nous avons rencontré quelques problèmes lorsque nous voulions automatiser la chaine de compilation. Nous nous sommes donc restreint à **Java 11**.

En créant le diagramme de classe nous avons décidé de créer une classe *Morpion3D* pour le jeu du morpion en 3D et une classe *Morpion2D* pour le jeu du morpion en 2D. Ces deux classes héritent d'une classe abstraite *Morpion* afin de pouvoir facilement instancier l'une des deux classe dans l'exécutable. Les méthodes restent les mêmes, mais celles de la classe correspondante seront appelées.

Classe Morpion2D

Nous avons décidé de travailler avec un tableau de String représentant la grille du morpion. L'affichage du tableau se faire à partir de 1 jusque la taille du tableau.

```
--- Affichage du Morpion2D ---  
1  2  3  
4  5  6  
7  8  9
```

Initialement nous avons fait le choix de travailler avec un tableau et ensuite de le transformer en grille, ce qui nous permettait d'avoir un accès plus simple aux indices.

Après réflexion, le choix de prendre un tableau en entrée plutôt que directement une grille n'était pas forcément judicieux, cela a provoqué la nécessité de création de fonction auxiliaire afin de travailler sur la grille et non le tableau.

Néanmoins nous sommes resté sur l'utilisation d'un tableau d'une dimension pour représenter notre grille.

Nous avons également choisi dès le début de pouvoir avoir une grille extensible, c'est pour cela que Morpion2D possède un attribut pour spécifier la taille. Le tableau de Morpion2D est donc de taille $n \times n$.

Afin de vérifier si une grille est pleine ou si une combinaison est gagnante, nous avons au départ développé un algorithme très peu performant et dur à comprendre, ce qui rend le debug compliqué. Par la suite nous avons mis en place un algo plus sobre et optimisé.

Une fonction qui retourne la combinaison à également été faite pour que l'on puisse l'afficher à la fin d'une partie


Classe Morpion3D

La classe Morpion3D héritent également de la classe Morpion, et doit implémenter toutes ses méthodes.

La grille est représenté par un tableau de tableau. Tel plusieurs Morpions superposés.

La méthode alignement fonctionne exactement comme celle de Morpion2D, il faut juste y ajouter des tests d'alignements supplémentaire pour les alignements en trois dimensions.

Méthode vérification d'alignement



La méthode *alignement* sert surtout à récupérer l'indice des éléments de la combinaison gagnante. Elle est grandement similaire à la méthode vérifiant l'existence d'une combinaison gagnante.

La fonction *alignement* vérifie pour Morpion2d tous les alignements possibles :

- Ligne
- Colonne
- Diagonales
- Et ceux sur toutes les dimensions

Nous vous invitons à regarder la documentation du code pour comprendre le fonctionnement de l'algorithme

Classe TicTacToe

C'est cette classe qui se charge de toute la gestion du jeu. C'est à dire :

- Le choix du mode de jeu
- Le choix de la taille de la grille
- La gestion des tours des joueurs
- L'affichage en sortie standard

Les méthodes d'affichage

Le jeu s'affiche entièrement sur la sortie standard, nous avons tout de même tenté de rendre cette sortie esthétique en ajoutant des couleurs, et des séparations. De plus la fonction d'affichage de la grille prends en compte la longueur des éléments qui lui sont donnés et affiche la grille en conséquence, ce qui donne une grille parfaitement alignée en toute circonstance.

Il existe donc quatre fonction d'affichage :

- Un affichage classique
- Un affichage esthétique
- Un affichage avec la position en cours
- Un affichage avec certains éléments surligné (spécifié en paramètre de méthode)

De plus, la méthode *toString* a été réécrite afin de pouvoir directement afficher la classe Morpion.

Une fonction auxiliaire *printspace* crée un string avec le nombre d'espaces spécifié en paramètres. Elle permet de compter le nombre d'espace nécessaire au maintien de l'alignement des chiffres lors de l'affichage. Cela permet de palier au problème d'affichage lorsqu'on atteint des grands nombre.



La méthode de placement

La fonction `placer` prend en paramètre la position, ou le code de placement pour le cas d'un morpion 3D, ainsi que le caractère à placer (en fonction du joueur).

La méthode lève une exception si la saisie est incorrecte, la position est incorrecte, ou déjà occupé.

Au niveau de l'interface on teste on demande préalablement au joueur de confirmer son choix avant d'afficher, sinon on redemande au joueur.

Les extensions

Dès le début nous avons considéré que la grille de jeu était extensible, nous avons codé `Morpion2D` puis `Morpion3D` avec cet objectif en tête.

L'affichage de la combinaison gagnante fait partie intégrante du jeu, tout joueur désire voir avec quel alignement il a gagné.

Extensions techniques

Pour développer en simultané, nous avons utilisé l'outil de versionnage `Git`. Chacun travaille sur une branche dédié et effectue un `Pull Request` qui doit être approuvé par le propriétaire du dépôt avant d'être intégré à la partie principale du projet.

Pour automatiser la compilation, la gestion des dépendances et d'autres fonctionnalités utiles au cycle de vie d'un projet Java, nous avons choisi d'utiliser `Apache Maven`.

Cela permet également de mettre en place des tests unitaires facilement. Les tests unitaires sont utilisés pour vérifier le bon fonctionnement des méthodes principales de notre programme.

Conclusion

Le jeu de Morpion en 3D était un défi intéressant. Au delà de la conception et le développement de l'application, la méthodologie de développement doivent être soigneusement planifiées pour garantir une production optimale et un code de qualité. Nous avons fait de notre mieux pour garantir une expérience de jeu agréable pour les joueurs. Malheureusement nous aurions aimé implémenter les autres extensions mais nous avons préféré se restreindre à ceux présentés, faute de temps.

Diagramme de Cas d'Utilisation – Description textuelle

Titre : Sélectionner mode de jeu
Description : Un joueur sélectionne le mode de jeu voulu parmi un morpion en 2D ou 3D
Acteur primaire : Le joueur qui lance l'application
Acteurs secondaires : /
Préconditions: Aucune partie n'est lancée
Scenario principal : <ol style="list-style-type: none">1. Le joueur sélectionne le mode de jeu parmi les morpions 2D et 3D2. Le joueur sélectionne la taille de la grille.
Postconditions: Le mode de jeu est sélectionné est la partie est lancée
Scenario alternatif : <ol style="list-style-type: none">1. Le joueur entre ou sélectionne un mode de jeu qui n'est pas correcte.2. Le système redemande au joueur de sélectionner un mode de jeu valide.3. Le joueur sélectionne une taille de grille incorrecte.4. Le système redemande au joueur de sélectionner une taille de grille correcte.

Diagramme de Cas d'Utilisation – Description textuelle

Titre : Placer un pion
Description : L'état du morpion est affiché au joueur et celui-ci sélectionne une case correspondant à la case où il désire poser son pion.
Acteur primaire : Le joueur.
Acteurs secondaires : /
Préconditions: La partie est en cours et la grille est dans un certain état.
Scenario principal : <ol style="list-style-type: none">1. Le joueur entre en entrée la position où il souhaite poser son pion.2. Le joueur valide la position en appuyant sur « Entrer ».
Postconditions: La partie est en cours et la grille reste dans le même état.
Scenario alternatif : <ol style="list-style-type: none">1. Le joueur entre en entrée une position qui ne correspond pas à un placement autorisé (indice invalide ou déjà occupé).2. Le système redemande au joueur de choisir où placer son pion.

Diagramme de Cas d'Utilisation – Description textuelle

Titre : Valide le placement d'un pion
Description : L'état du morpion et la position du pion sur le point d'être placé est affiché au joueur et celui-ci décide de valider le placement ou sélectionne une nouvelle position où placer son pion.
Acteur primaire : Le joueur.
Acteurs secondaires : /
Préconditions: La partie est en cours et la grille est dans un certain état avec la position du pion à placer sur la grille.
Scenario principal : 1. Le joueur valide le placement en appuyant sur « Entrer »
Postconditions: La partie est en cours et le morpion est mise à jour avec le pion du joueur placé
Scenario alternatif : 1. Le joueur entre en entrée une nouvelle position (<i>cf. remplacer un pion</i>). 2. Le système redemande au joueur de valider son placement (<i>cf. Placer un pion</i>).

Diagramme de Cas d'Utilisation – Description textuelle

Titre : Replacer le pion
Description : Cette description textuelle correspond au cas où lors de la validation du placement d'un pion, le joueur décide de sélectionner une autre position.
Acteur primaire : Le joueur.
Acteurs secondaires : /
Préconditions: La partie est en cours et la grille est dans un certain état avec la position du pion à placer sur la grille.
Scenario principal : <ol style="list-style-type: none">1. Lors de la validation du pion le joueur entre une nouvelle position.2. Le joueur valide la position en appuyant sur « Entrer ».
Postconditions: La partie est en cours et la grille reste dans le même état.
Scenario alternatif : (<i>cf. placer un pion</i>).