

Rapport Final - AZUL A

Par Arthur Dauphin, Lucas Puissant-Baramboit, Jonathan Samuel,
Jérémy Berge et Nicolas Preaux

Sommaire :

Sommaire :	1
1. Point de vue général de l'architecture et des fonctionnalités	2
Glossaire	2
Représentation générale de la solution (diagramme d'activité)	3
Point sur les fonctionnalités traitées	3
2. Modélisation de l'application	4
Analyse des besoins (exigences) : Cas d'utilisation	4
Acteurs	4
Diagrammes de Cas d'utilisation	4
Scénarios	4
Conception logicielle	4
Diagrammes de Classes	4
Diagrammes de Séquence	10
3. Conclusion	10
Analyse de votre solution : points forts et points faibles	10
Comment prendre la suite du développement de votre projet	10

Le symbole Δ met en valeur les changements effectués depuis le rapport intermédiaire.

1. Point de vue général de l'architecture et des fonctionnalités

Glossaire

Δ : Le glossaire était vide lors du rapport intermédiaire.

Utilisateur : Dans le cas de ce projet, le client.

Random : Aléatoire en anglais.

Fabrique : Pioche aléatoirement 4 tuiles à chaque tour, les bots piochent dedans.

Centre de la table : Contient toutes les tuiles non choisies par les bots, le premier bot qui choisit de piocher dans le centre de la table obtient un malus.

Plancher : Agit comme un accumulateur de malus. Toutes les tuiles ne peuvent pas être jouées par le bots tombent dans le plancher et génèrent un malus selon un paterne.

Ligne motif : Lignes sur lesquelles les bots placent les tuiles choisies depuis une fabrique ou le centre de la table. Ils ne peuvent y placer que le bon nombre (sinon elles tombent dans le plancher) de tuiles et de la bonne couleur par rapport à celles précédemment placées.

Mur : Point central du jeu ou les joueurs (dans notre cas les bots) pose les tuiles lorsqu'ils ont complété une ligne motif. Une tuile posée permet d'accumuler des points et lorsqu'une ligne horizontale est créée sur le mur la partie est finie.

Plateau : Mode de jeu qui permet de jouer de plusieurs manières différentes. Dans le mode par défaut les scores et malus sont basiques et les joueurs doivent placer les tuiles selon un paterne précis.

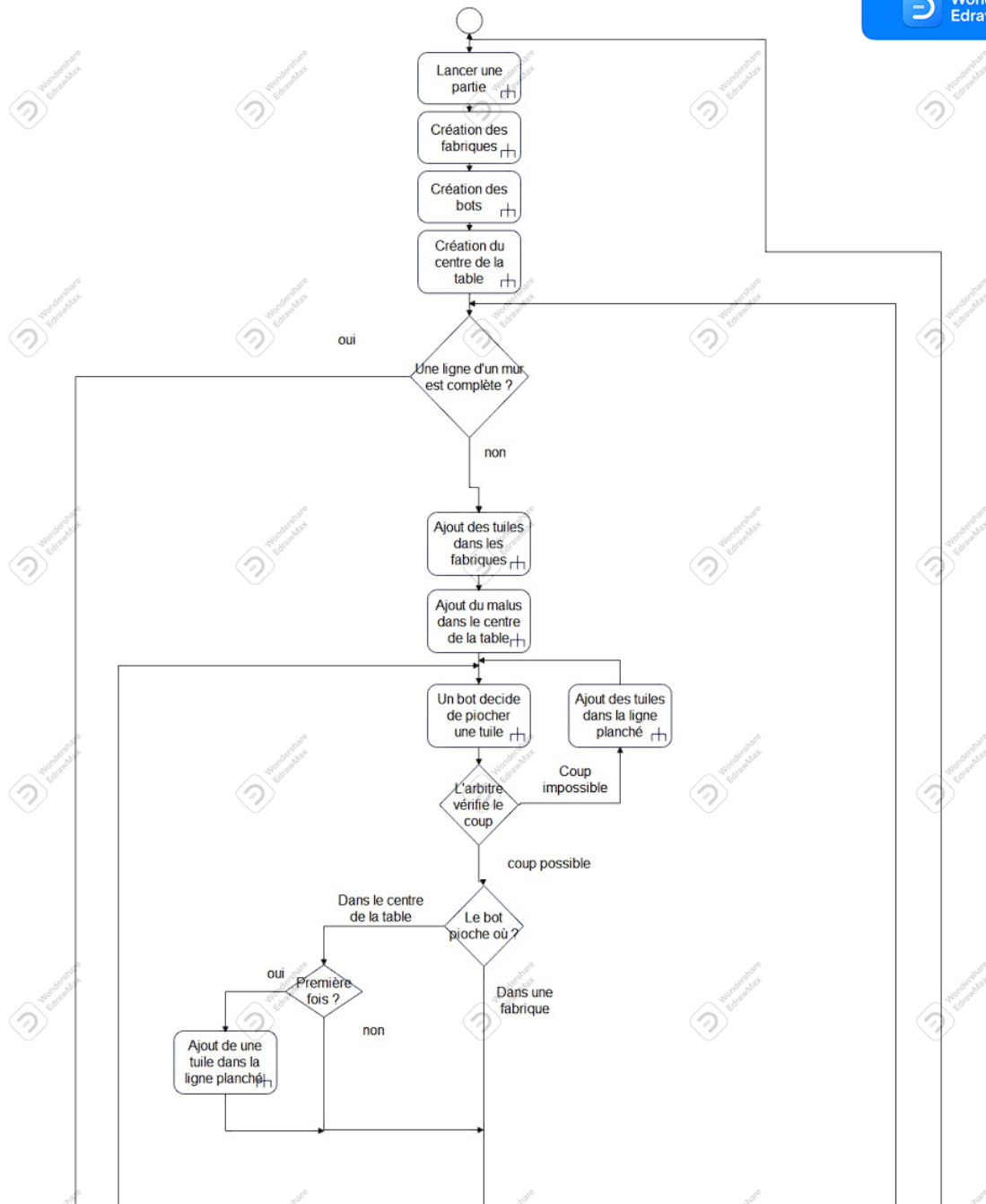
Mur gris : Mode de jeu qui diffère du par défaut et permet aux bots de placer leurs tuiles à n'importe quel endroit si il n'y a pas déjà une tuile de cette couleur sur la ligne et la colonne accueillante.

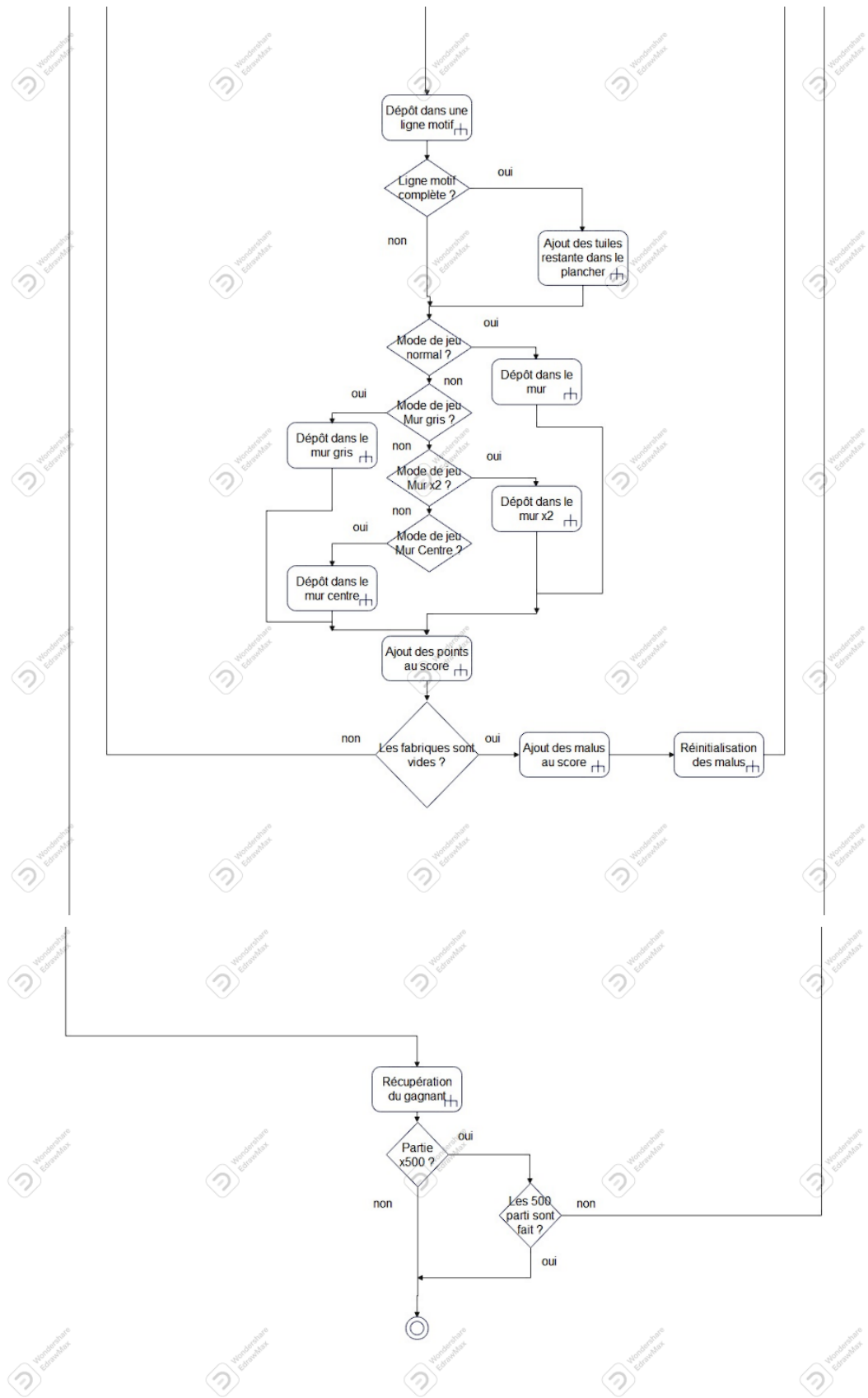
Face 1 : Mode de jeu qui diffère du par défaut et permet aux bots de placer leurs tuiles à des endroits mixant mur par défaut et mur gris. Cette face change le score en effet si une tuile est placée sur un paterne alors le score créé par cette tuile sera doublé.

Face 2 : Mode de jeu qui diffère du par défaut et permet aux bots de placer leurs tuiles à des endroits mixant mur par défaut et mur gris. Cette face change le score en effet à la fin d'une partie les bonus sont augmentés par rapport au autre mode de jeu.

Représentation générale de la solution (diagramme d'activité)

Δ : Changement radical du diagramme d'activité.





Point sur les fonctionnalités traitées

Δ : Ajout d'un grand nombre de nouvelles fonctionnalités.

Tout d'abord au niveau des fonctionnalité basique :

- La solution peut lancer une partie normale avec un nombre de bots variables choisi par l'utilisateur.
- L'utilisateur peut jouer avec deux types de bots (random et intelligent) qui respectent toutes les règles.
- Il y a un affichage complet des murs, des lignes motifs, du score, des égalités et du gagnant. (De plus l'affichage est prévu pour des consoles avec des arrière-plan noir et blanc)
- La solution comporte aussi le plateau, les fabriques, le centre de la table et le plancher avec gestion du score (des bonus et des malus).
- Le mur gris a aussi été ajouté et est jouable.

En ce qui concerne les fonctionnalité bonus :

- Il y a deux nouveaux modes de jeu: la face 1 et la face 2.
- Les nouvelles règles pour ces modes de jeu : Une sorte de mélange entre le mur classique et le mur gris.
- Les nouveaux calculent des scores notamment les scores en jeu pour la face 1 qui sont doublés par moment et les scores de fin de partie qui sont augmenté pour la face 2.
- Les threads ont été implémentés pour permettre au client d'exécuter de nombreuses parties en simultané (500+ par exemple).
- De plus l'affichage a été revu pour que les statistiques des parties soient affichées.

2. Modélisation de l'application

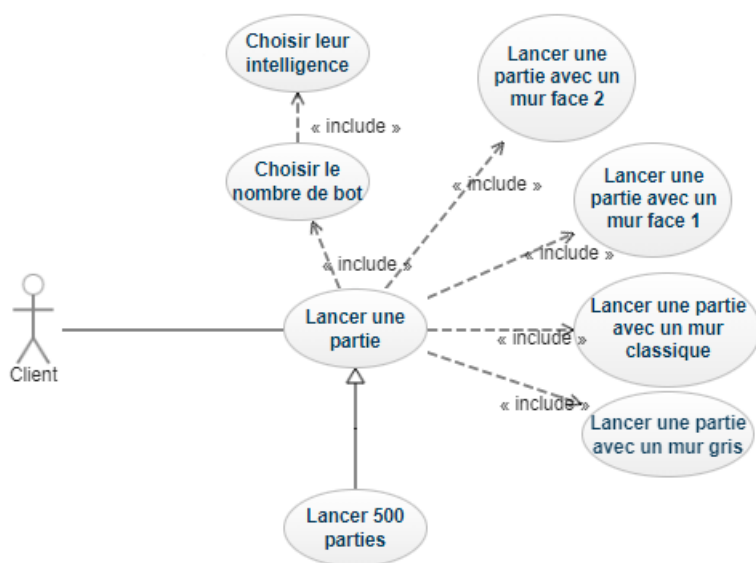
Analyse des besoins (exigences) : Cas d'utilisation

Acteurs

Δ : Le nombre d'acteurs a été réduit à un seul pour être en adéquation avec le projet.
Le seul acteur pour ce projet est le client.

Diagrammes de Cas d'utilisation

Δ : Le diagramme de cas d'utilisation a été refait pour une meilleure compréhension avec seulement les user story permettant à l'acteur d'exécuter une partie.



Scénarios

Δ : Les scénarios ont été refaits pour être en adéquation avec le nouveau diagramme de cas d'utilisation.

Le client (ou l'utilisateur) lance une partie, pour ça il choisit le mode de jeu qu'il veut, le nombre de bot ainsi que leur intelligence.

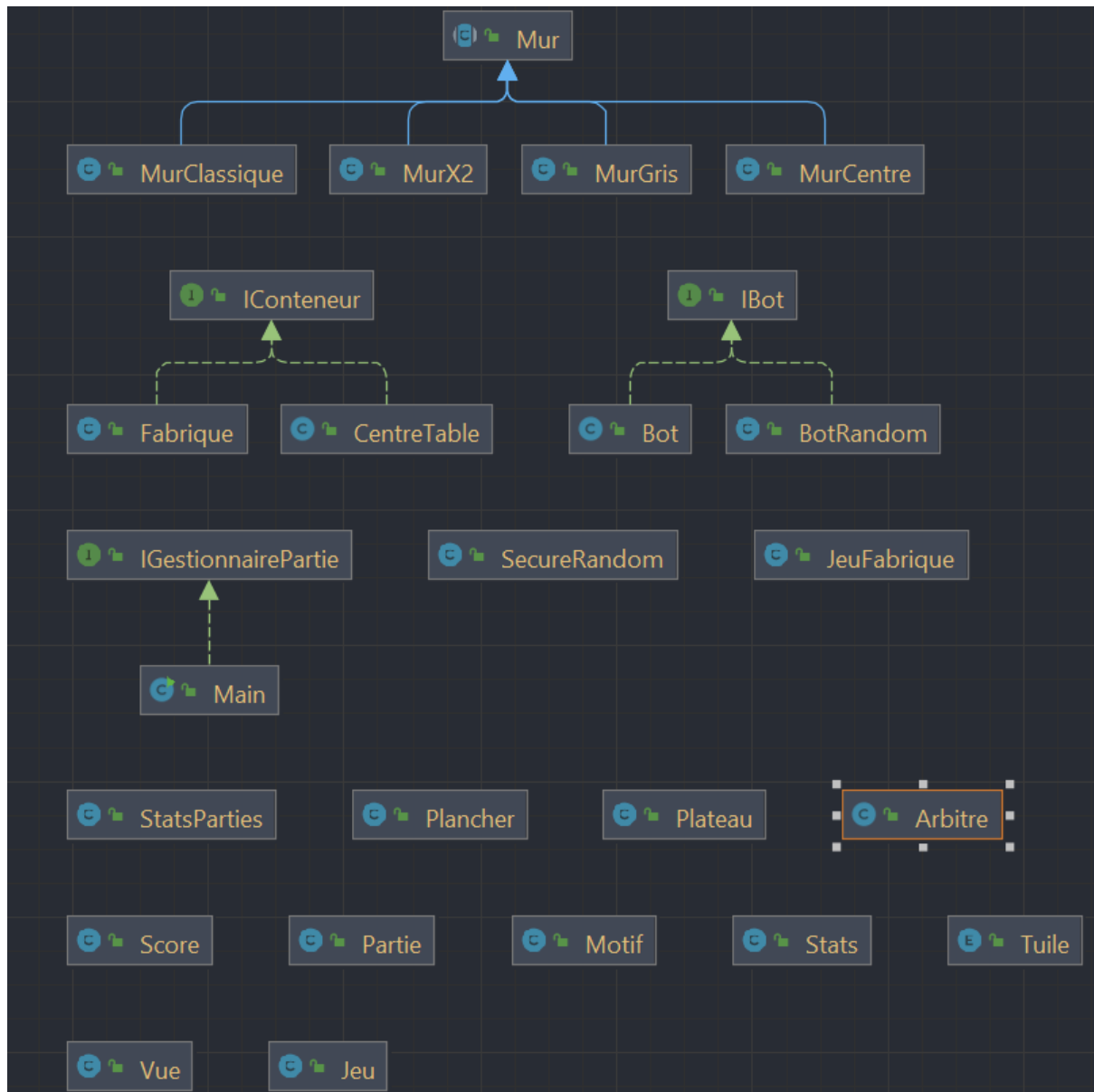
Cela va de paire avec le scénario de lancer 500 parties. En effet le client lance 500 parties, pour cela il choisit le mode de jeu, le nombre de bot ainsi que leur intelligence.

Conception logicielle





















































Diagrammes de Classes

Δ : Le diagramme de classes n'était pas présent lors du rendu intermédiaire

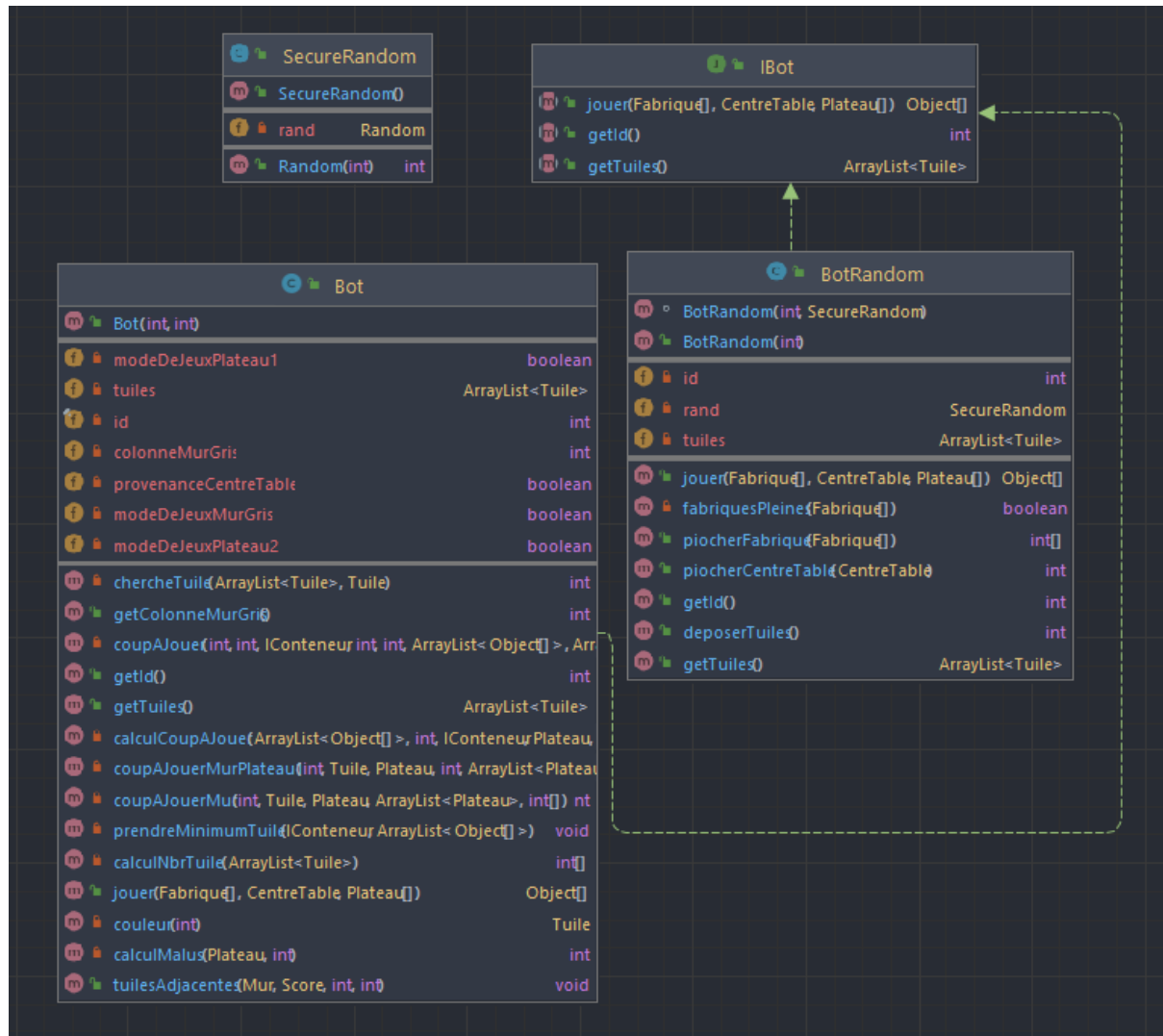
Vue d'ensemble :



Package arbitre :

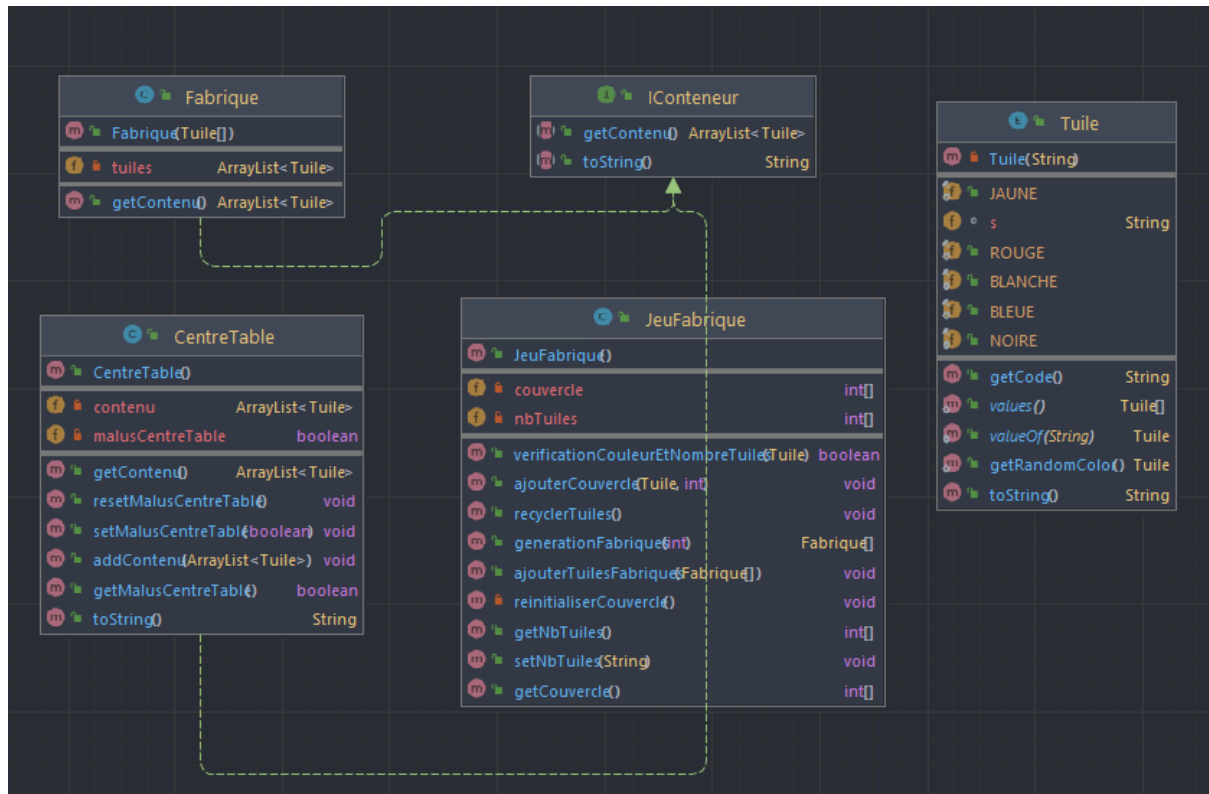
Arbitre		
	 Arbitre (int, IBot[])	
	 joueurs	IBot[]
	 numBot	int
	 typeDePartie	int
	 plateauJoueurs	Plateau[]
	 ajouterAuMurCentre (StatsParties, JeuFabrique)	void
	 ◦ verifierPoseTuiles (int, IBot, Plateau, Vue, int, StatsParties, JeuFabrique)	
	 ◦ ajouterAuMurNormal (StatsParties, JeuFabrique)	void
	 recupererPossiblesGagnants (StatsParties)	ArrayList< IBot >
	 aucuneTuiles (JeuFabrique)	boolean
	 tuilesAdjacentesX2 (Mur, Score, int, int)	void
	 estUneLigneComplete (IBot[])	boolean
	 recupererGagnantFinal (ArrayList< IBot >)	ArrayList< IBot >
	 ◦ ajouterAuMurX2 (StatsParties, JeuFabrique)	void
	 fabriquesPleinesEtCentreTablePlein (Fabrique [], CentreTable)	boolean
	 ajouterAuMur (StatsParties, JeuFabrique)	void
	 getPlateauJoueurs ()	Plateau[]
	 getPlateauJoueurs (int)	Plateau
	 ◦ verifierLigneMotif (Plateau, int, Tuile)	boolean
	 ◦ ajouterAuMurGris (StatsParties, JeuFabrique)	void
	 verifierCoupBot (Fabrique [], CentreTable, Vue, int, StatsParties, JeuFabrique)	
	 tuilesAdjacentes (Mur, Score, int, int)	void
	 getJoueurs ()	IBot[]
	 recupererNbLigneComplete (IBot)	int
	 recupererNbColonneComplete (IBot)	int
	 recupererNbCouleurComplete (IBot)	int

Package bot :



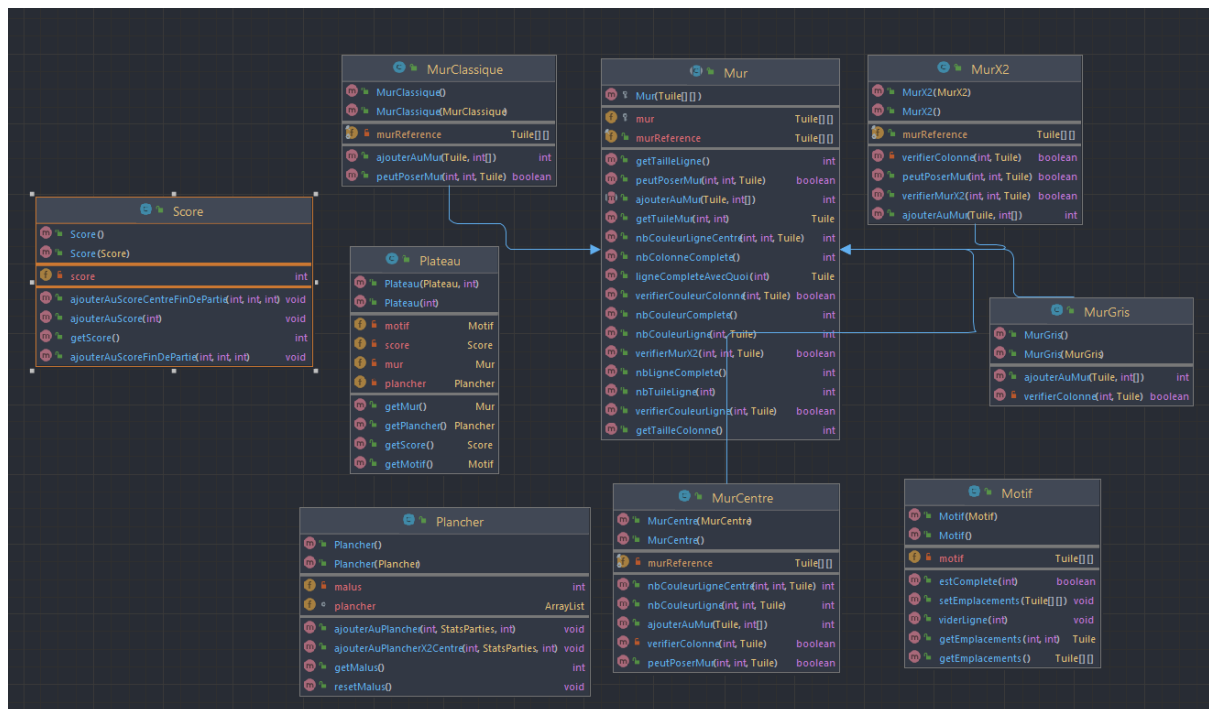
Les classes Bot (Bot intelligent) et BotRandom (Bot sans intelligence) implémentent l'interface IBot.

Package fabrique :



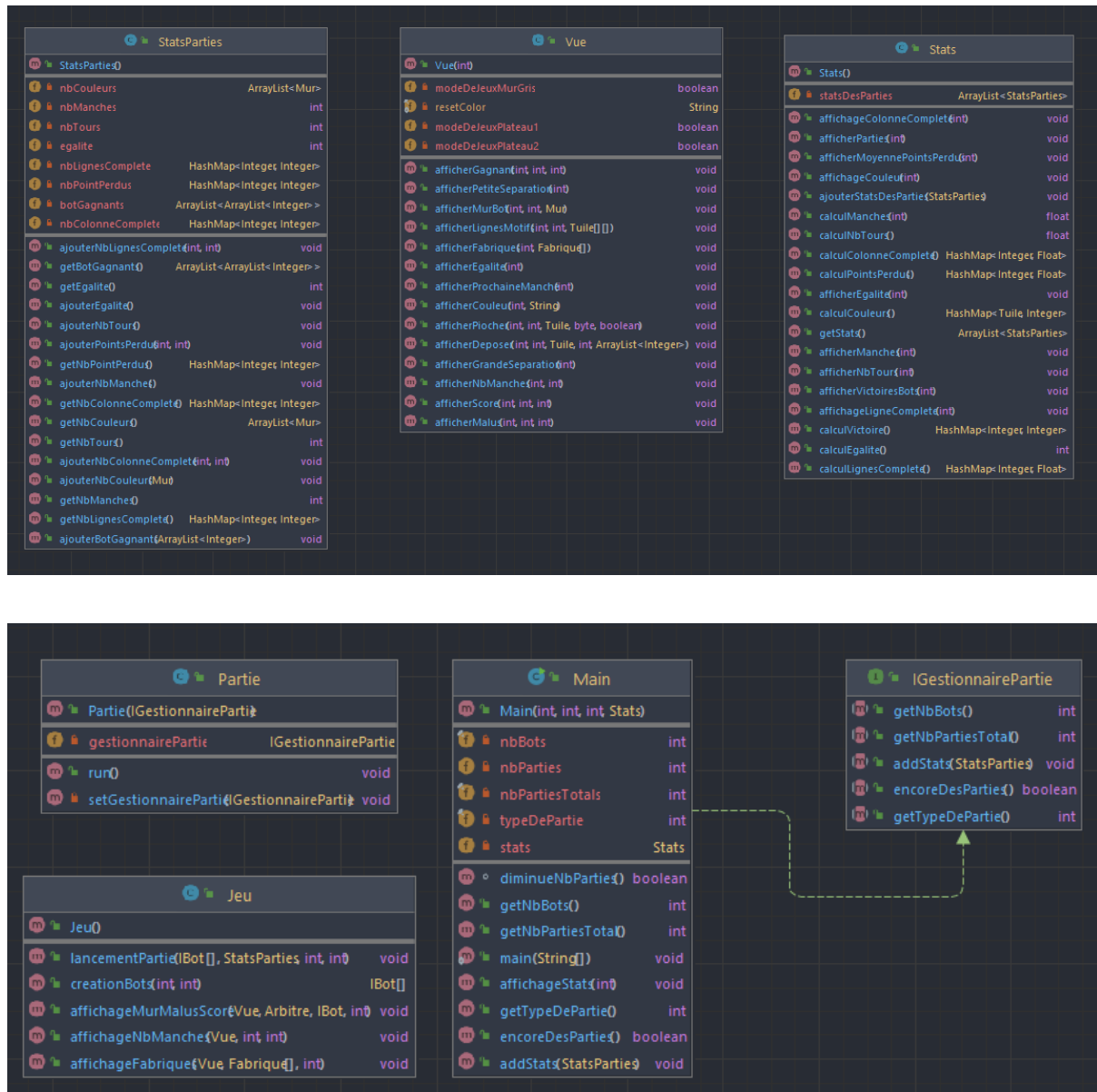
Les classes Fabrique et CentreTable implémentent l'interface IConteneur.

Package plateau :



Les classes MurClassique, MurCentre, MurGris et MurX2 héritent de la classe abstraite Mur.

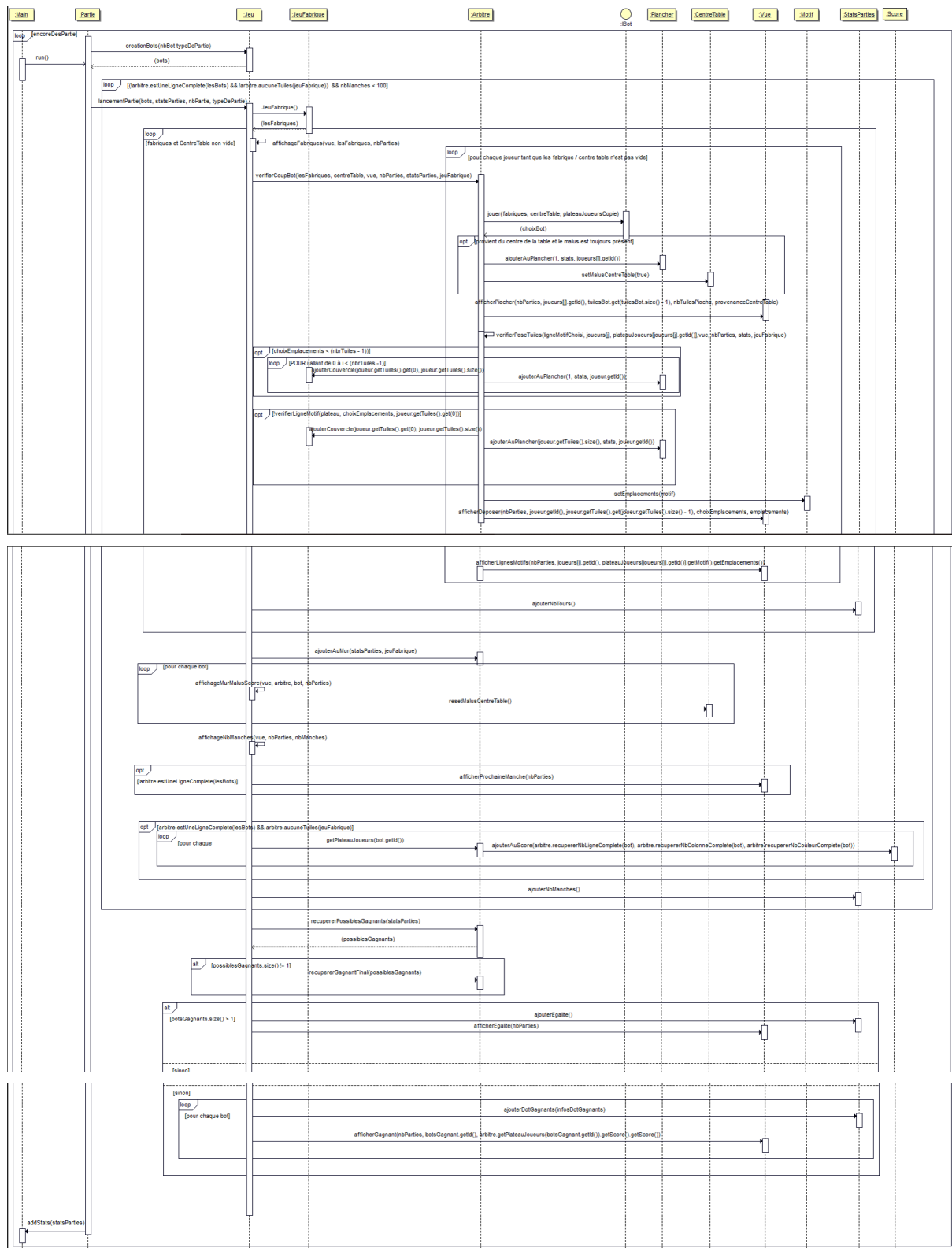
Package visuel :



La classe Main implémente l'interface IGestionnairePartie. Les classes Main, Partie et Jeu permettent de démarrer le jeu.

Diagrammes de Séquence

Δ : Les diagrammes de séquences ont été complètement revus.



3. Conclusion

Analyse de votre solution : points forts et points faibles

Δ : Les points forts et faibles ont été changés pour avoir un point de vue plus général de la solution.

- Points forts :
 - L'arbitre vérifie les coups des bots et l'empêche de « tricher »
 - Réalisation d'un bot intelligent qui fait de bon choix
 - Plusieurs modes de jeux
- Point faible :
 - Manque de documentation
 - Manque de tests
 - Complexité du Bot

Comment prendre la suite du développement de votre projet

Δ : Au précédent rapport nous avons énoncé des changements plutôt dirigés vers la gestion de projet. Ici, les changements ou améliorations futures seront centrés vers la viabilité de notre code.

Il faudrait introduire la notion de réseau avec une architecture client-serveur pour satisfaire toutes les exigences du client, ajouter une intelligence de bot expert pour pouvoir mieux analyser les résultats, améliorer l'affichage des parties en simultanées, faire beaucoup plus de tests unitaires, d'intégration et pourquoi pas fonctionnels. De plus, un refactoring des classes concernant le score devrait être fait pour qu'il soit mieux géré.