

TSURO

2E SEMESTRE



Rapport

Projet de programmation

GROUP VP2 B

Rayane Arkam
Naïm Cherchour
Sami Chioukh
Dejan Kecman
Sifaks Lounici



SOMMAIRE

I. INTRODUCTION	03 - 06
• <i>Objectif du rapport</i>	
• <i>Présentation du jeu</i>	
II. ARCHITECTURE DU PROJET	07 - 13
• <i>Projet MVC</i>	
• <i>Diagramme des classes</i>	
• <i>Rôles des classes</i>	
III. SPRINTS ET DÉVELOPPEMENT	13 - 15
• <i>Détails des sprints</i>	
IV. PROJET SUR GITLAB	16 - 18
• <i>Gestion des branches</i>	
V. FONCTIONNALITÉS	19 - 20
• <i>Fonctionnalités achevées</i>	
• <i>Fonctionnalités inachevées</i>	
VI. CONCLUSION	21
<i>Ce qu'on a tiré du projet</i>	

INTRODUCTION

I. Objectif du rapport

Ce rapport offre une analyse détaillée de notre projet collaboratif et met en lumière les défis que nous avons rencontrés en intégrant plusieurs éléments clés dans l'architecture de notre logiciel. Nous avons adopté une approche structurée pour orchestrer efficacement les interactions entre les différentes composantes de notre application, en utilisant les principes du Modèle-Vue-Contrôleur (MVC) pour assurer une séparation claire des responsabilités.

Dans ce document, nous explorons en profondeur l'architecture de notre système, montrant comment le MVC a été appliqué pour favoriser une gestion efficace des interactions entre les composants. Nous abordons également notre approche de développement agile, marquée par des sprints successifs et l'utilisation de GitLab pour faciliter la collaboration et l'intégration continue.

Le jeu était conçu pour offrir une expérience utilisateur riche et interactive. Nous avons intégré un bot intelligent pour enrichir le gameplay et développer une interface graphique attrayante pour améliorer l'engagement des utilisateurs. L'efficacité des algorithmes a été une priorité pour assurer des performances optimales.

Nous avions également l'ambition d'incorporer un mode réseau permettant aux joueurs de se connecter et de s'affronter en ligne, enrichissant ainsi l'expérience sociale et compétitive du jeu. Bien que nous ayons réussi à créer un serveur de jeu et à connecter des clients, nous n'avons pas pu finaliser la transmission des données avant la fin du projet.

L'objectif de ce rapport est de présenter de manière transparente et exhaustive notre progression, les enseignements tirés et comment ces expériences influenceront nos futurs engagements dans le domaine informatique. En partageant ces connaissances, nous espérons non seulement documenter notre propre voyage d'apprentissage mais aussi fournir des repères utiles pour de futurs projets de développement logiciel.

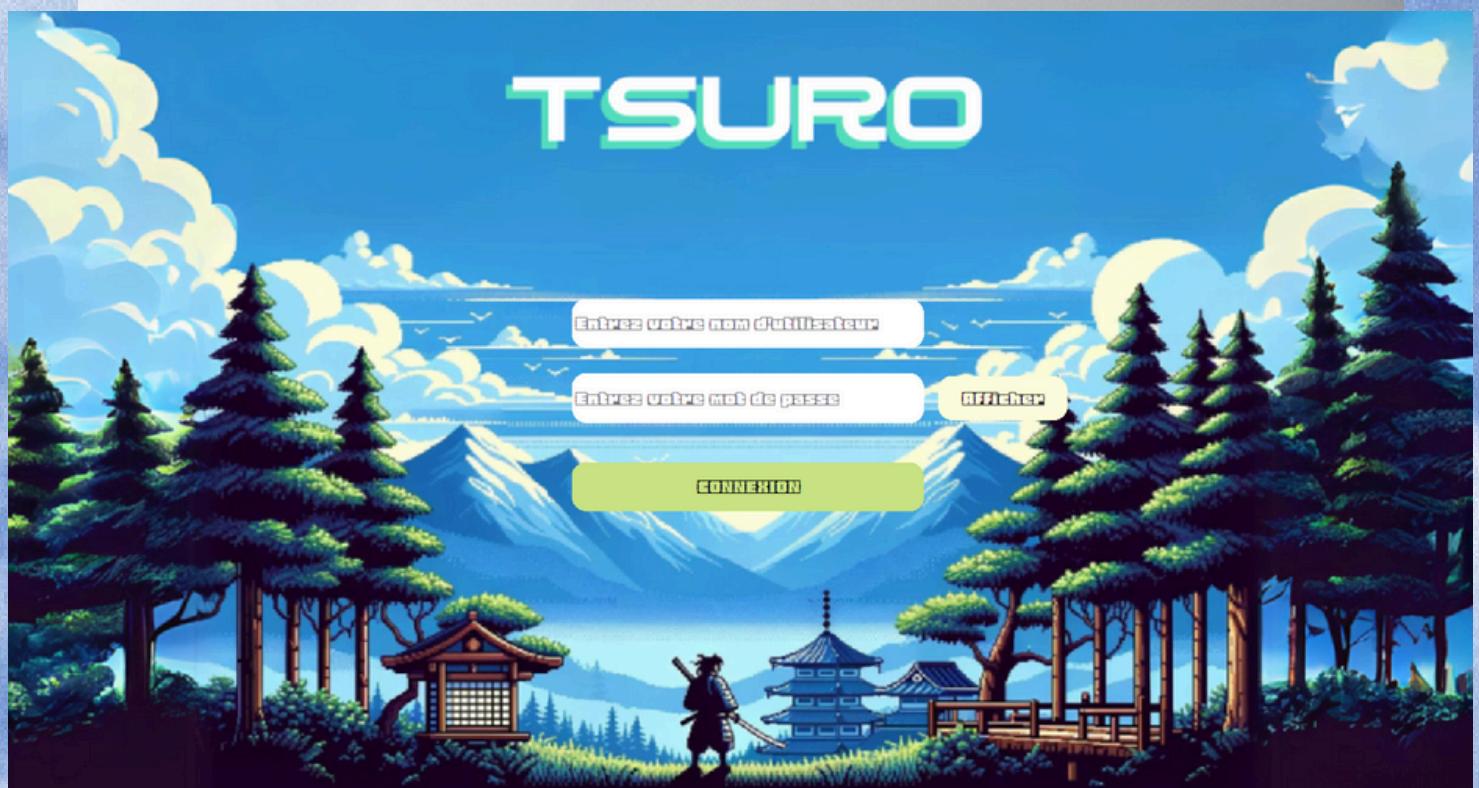
PRÉSENTATION DU JEU

04



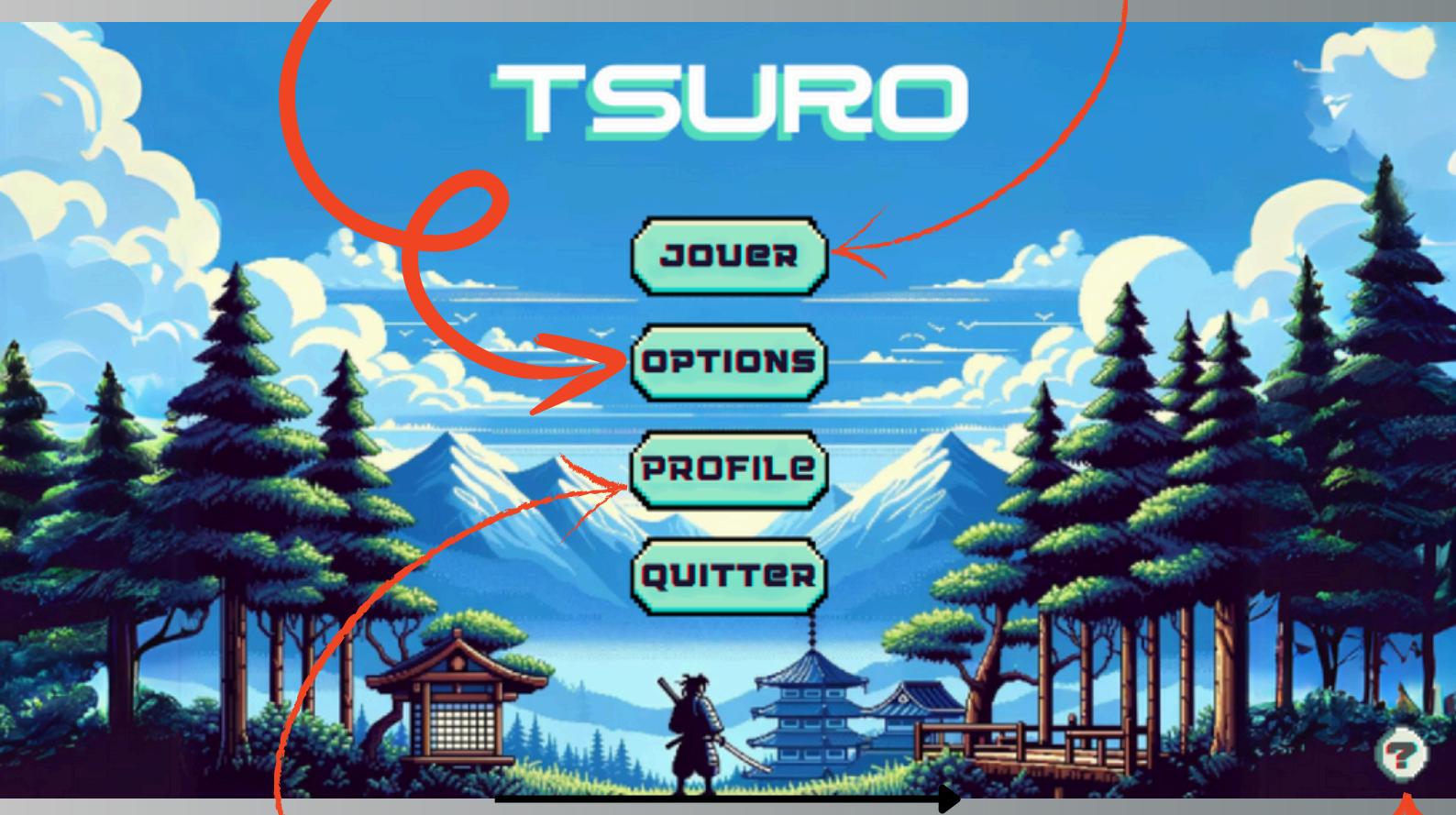
Tsuro est un jeu de plateau pouvant se jouer de 2 à 8 joueurs. Lors du lancement, un écran d'accueil apparaît, invitant les joueurs à appuyer sur la barre d'espace pour accéder à la page de connexion.

Cette page permet aux joueurs de se connecter à un compte existant ou de s'inscrire pour créer un nouveau compte.



Gestion du son du jeu

Jouer en choisissant le mode de jeu et le nombre de joueurs



Possibilité de charger des parties sauvegardées associées au compte

Explication des règles sous forme de pages à tourner

Nous tenions à avoir un jeu fonctionnel évidemment mais nous voulions aussi que le résultat soit esthétiquement plaisant, et c'est ce que nous avons essayé de communiquer avec les différents menus du jeu.



PATEAU DE JEU

Bouton de sauvegarde des parties associées au profil

Temps restant: 7 secondes

Deck de Tuiles du joueur

Plateau de tuiles de taille 6*6

Screen d'une partie multi

Thème inspiré du vrai jeu



Tsuro of the Seas Vétérans d'extension du The Seas - version anglaise

Marque : Calliope

4,6 ★★★★★ 338 évaluations

24³⁷ €

Retours GRATUITS

Les prix des articles vendus sur Amazon incluent la TVA. En fonction de votre adresse de livraison, la TVA peut varier au moment du paiement. Pour plus d'informations, Veuillez voir les détails.

Message promotionnel Economisez 5% sur Docsmagic.de Pro-Player Album lorsque vous... 10 promotions

Style: Tsuro Expansion Veterans of the Seas

Tsuro Expansion Veterans of the Seas
24,37 €

Tsuro Phoenix Rising.

48,75 €

II. Architecture du projet

MVC



Le projet que nous avons développé suit les principes de l'architecture Modèle-Vue-Contrôleur (MVC), qui est essentielle pour créer des applications structurées, maintenables et évolutives. Nous avons mis en œuvre des concepts tels que les listeners, les contrôleurs, le modèle, la vue, et l'utilisation de l'observation (pattern Observer) pour une interaction réactive entre ces composants.

• Modèle

Le **modèle** contient les données de l'application ainsi que la logique métier. Dans notre jeu, cela inclut par exemple les positions des joueurs, et les règles qui dictent la progression du jeu. Le modèle est conçu pour être totalement indépendant de la vue et du contrôleur. La classe Game contient une méthode jouerUnTour(Tuile tuile) qui met à jour le plateau de jeu basé sur l'action du joueur, illustrant clairement son rôle dans la gestion de la logique métier.

• Vue

La vue est responsable de l'affichage des informations, c'est-à-dire qu'elle génère la sortie visuelle basée sur les données qu'elle reçoit du modèle. Par exemple, la classe TuilePanel affiche les tuiles du jeu. Elle accède aux données via des méthodes comme getTuile() de l'interface ReadOnlyGame, assurant une interaction en lecture seule avec le modèle pour préserver l'intégrité de l'état du jeu.

• Contrôleur

Le Controller agit comme le médiateur entre la vue et le modèle. Il implémente des interfaces comme MouseListener et ActionListener, ce qui lui permet de réagir aux interactions de l'utilisateur. Lorsqu'une tuile est cliquée, par exemple, il exécute placerTuileFromDeck() et handleTilePlacement() pour actualiser le modèle. Il gère également la rotation des tuiles avec rotateTile(), traduisant ainsi directement les actions de l'utilisateur en modifications du modèle.

• **Listeners et Observers**

Le code intègre des patterns avancés pour la gestion des événements et les notifications de changement. Les **listeners** dans le contrôleur réagissent aux interactions de l'utilisateur, telles que les clics de souris et les appuis sur les boutons, en modifiant l'état du jeu via des méthodes spécifiques du modèle.

Les **observers** (GameObserver) sont alertés des modifications dans le modèle par des méthodes comme **notifyObservers()**. Suivant le pattern Observer, ils actualisent la vue en fonction des changements du modèle, garantissant que l'interface utilisateur affiche toujours l'état actuel du jeu, comme lors de la rotation ou du placement de tuiles.

• **Organisation des fichiers**

Notre code montre également une bonne organisation des fichiers selon les principes MVC. Les classes de modèle, vue, et contrôleur sont séparées dans différents packages et fichiers, favorisant la modularité et la maintenabilité du code. Cela permet à l'équipe de développeurs de travailler sur des aspects spécifiques du projet sans interférence, améliorant ainsi la gestion et la collaboration.

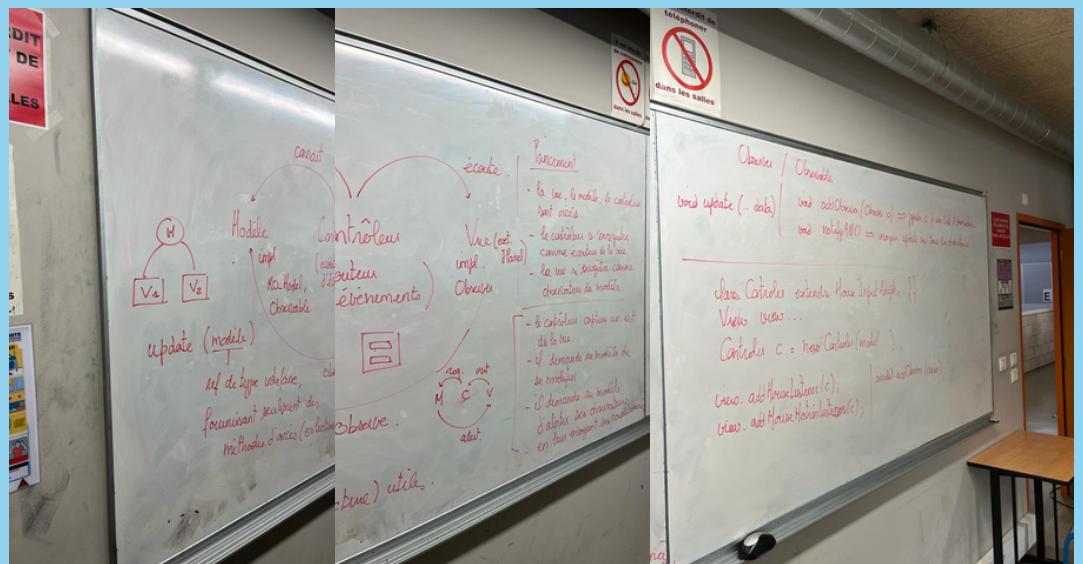
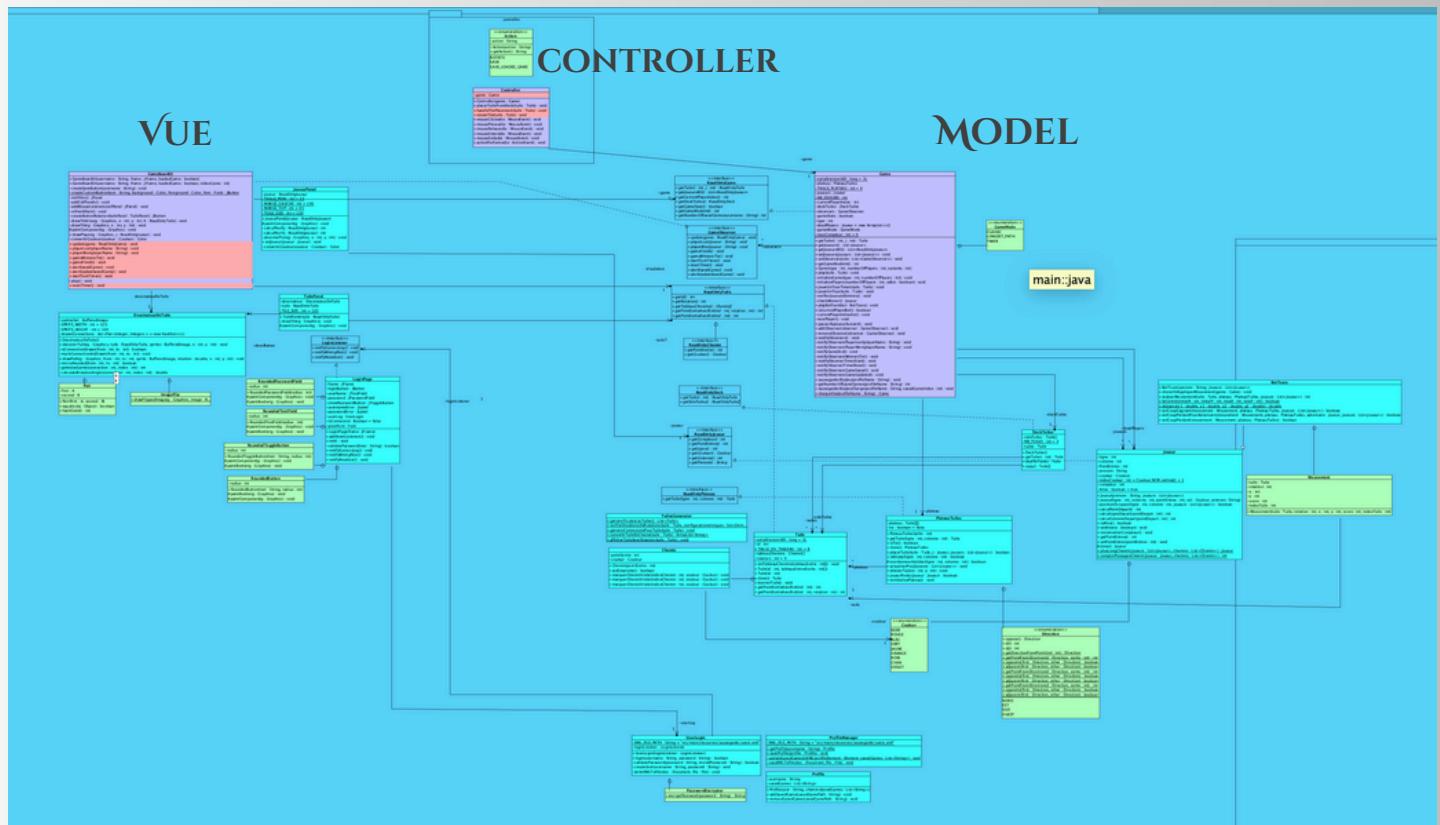


Diagramme des classes

INTERACTION MVC



Les relations entre les classes sont principalement des associations et des compositions :

- **Associations :**

1. La classe Controller est associée à la classe Game, ce qui lui permet de manipuler l'état du jeu.
2. La classe Game utilise des instances de Joueur, DeckTuiles, et Plateau.
3. Les Joueur possèdent des Tuile.

- **Compositions :**

1. Game contient une collection de Plateau, représentant les différents plateaux de jeu.
2. Plateau contient une grille de Tuile.

MODEL

• Game.java :

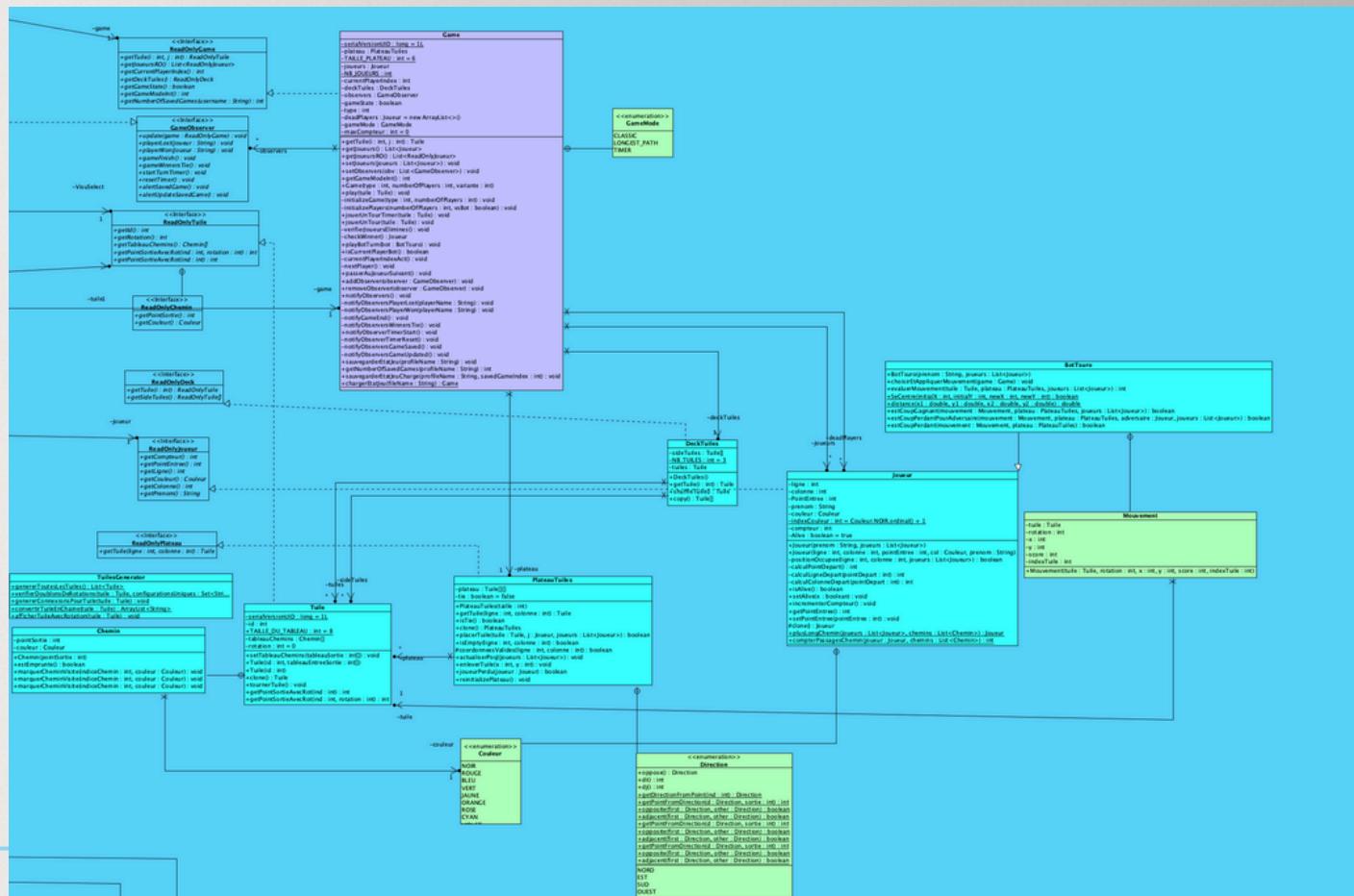
- Responsabilités :
 - Gestion de l'état du jeu.
 - Implémentation des règles du jeu.
 - Interaction avec les joueurs et les tuiles.
 - Communication avec les observateurs pour mettre à jour l'état du jeu.

• BotTsuro.java :

- Responsabilités :
 - Prendre des décisions de jeu automatique.
 - Interagir avec la classe Game pour effectuer des mouvements.

• Joueur.java :

- Responsabilités :
 - Stocker les informations du joueur (nom, couleur, score, etc.).
 - Gérer les actions et les mouvements du joueur.



- **Tuile.java :**

- Responsabilités :
 - Stocker les informations de la tuile (positions de connexion, orientation).
 - Fournir des méthodes pour manipuler et interagir avec les tuiles.

- **TuilesGenerator.java :**

- Responsabilités :
 - Créer et initialiser les tuiles avec les configurations nécessaires.

- **PlateauTuiles.java :**

- Responsabilités :
 - Gérer la disposition et les interactions des tuiles sur le plateau.
 - Vérifier les mouvements et les connexions entre les tuiles.

- **DeckTuiles.java :**

- Responsabilités :
 - Stocker et distribuer les tuiles aux joueurs.

- **ReadOnlyGame.java :**

- Responsabilités :
 - Permet aux vues et autres composants d'accéder à l'état du jeu sans possibilité de modification.

- **ReadOnlyTuile.java, ReadOnlyPlateau.java**

- ReadOnlyDeck.java, ReadOnlyJoueur.java :**

- Responsabilités :
 - Permettent à la vue de s'afficher sans pouvoir modifier leurs valeurs dans le Model.

- **ProfileManager.java :**

- Responsabilités :
 - Stocker et récupérer les informations des profils des utilisateurs.
 - Gérer les connexions et déconnexions des utilisateurs.



DessinateurDeTuile.java

- #### • Responsabilités :

Dessine les tuiles du jeu, et gère les images des tuiles et les manipulations graphiques.

Classe GameBoardUI.java

- #### • Responsabilités :

Elle contient des méthodes pour initialiser, mettre à jour et afficher le plateau de jeu. Elle gère les interactions utilisateur sur le plateau, comme le placement des tuiles et les mouvements des pions. Elle coordonne les composants graphiques du plateau de jeu avec l'état actuel du jeu.

Elle implémente **GameObserver**, car elle observe le model afin de l'afficher.

Digitized by srujanika@gmail.com

Classe JoueurPanel.java

- #### • Responsabilités :

Elle contient des méthodes pour mettre à jour les informations affichées telles que le nom du joueur, ses points, et ses tuiles en main. Elle permet d'afficher dynamiquement les changements d'état d'un joueur pendant le jeu.

Classe LoginListener.java

- #### • Responsabilités :

Elle contient des méthodes abstraites pour notifier les résultats de la connexion, comme le succès de la connexion, les erreurs de mot de passe, et la création d'un nouvel utilisateur. Elle est implémentée par des classes qui nécessitent des actions spécifiques suite aux événements de connexion.

Classe LoginPage.java

- Responsabilités :

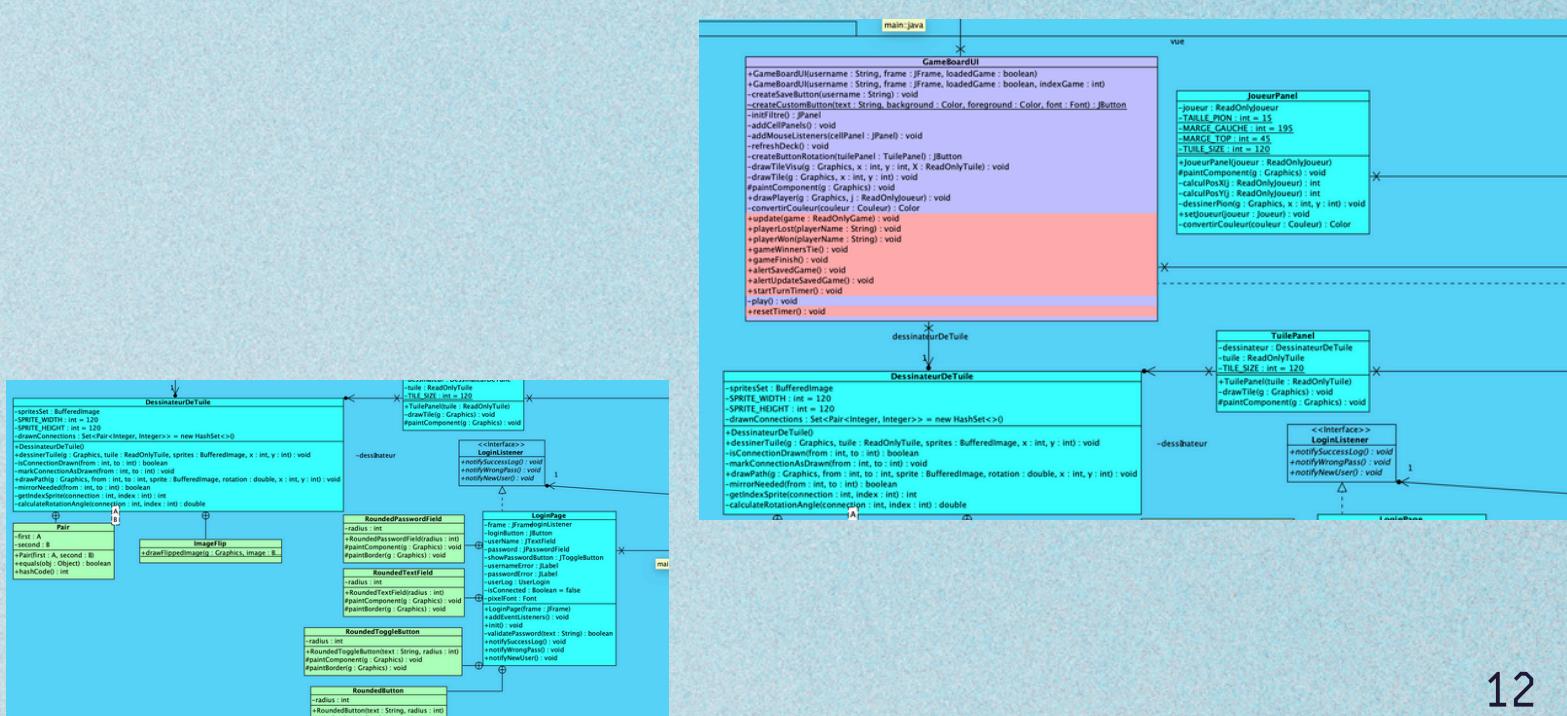
Elle implémente des écouteurs pour gérer les interactions utilisateur comme la saisie du texte et le clic sur les boutons.

Elle coordonne les vérifications des informations de connexion et les réponses appropriées en cas d'erreur ou de succès de la connexion.

Classe TuilePanel.java

- #### • Responsabilités :

Elle contient des méthodes pour dessiner les tuiles de manière interactive, y compris la rotation et le placement des tuiles sur le plateau, met à jour l'affichage des tuiles.



CONTROLLER

• Responsabilités

La classe Controller joue un rôle crucial dans la gestion des interactions entre la vue et le modèle au sein de l'architecture MVC (Modèle-Vue-Contrôleur). Voici ses principales responsabilités :

1. Gestion des Entrées Utilisateur :

- Le contrôleur interprète les actions de l'utilisateur (comme les clics de souris et les frappes au clavier) et met à jour le modèle en conséquence.
- Il gère les événements de souris pour les mouvements et les clics, afin de permettre les actions telles que le placement des tuiles et les déplacements des joueurs.

2. Coordination des Interactions Vue-Modèle :

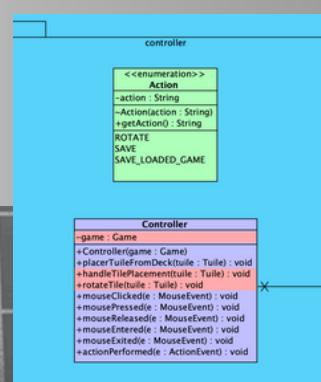
- Le contrôleur transmet les commandes de la vue vers le modèle, et s'assure que la vue est mise à jour lorsque le modèle change.
- Il agit comme un intermédiaire pour synchroniser les modifications entre le modèle de jeu et l'interface utilisateur.

3. Logique de Jeu :

- Il contient la logique qui définit comment les mouvements et les actions des joueurs sont validés et appliqués dans le jeu.
- Il vérifie les règles du jeu pour s'assurer que chaque mouvement est légal et met à jour l'état du jeu en conséquence.

4. Mise à Jour et Notifications :

- Le contrôleur écoute les changements dans le modèle et déclenche des mises à jour dans la vue pour refléter ces changements.
- Il utilise des observateurs pour réagir aux événements de jeu, comme le début ou la fin d'une partie, ou les mises à jour des scores des joueurs.



III. Sprints et Développement

• Sprints Retrospective

Dans notre travail, nous nous sommes basés sur le cahier des charges que nous avons élaboré dès le début du projet. Pendant les phases de modélisation et de développement extensif, la planification commune était organisée hebdomadairement sous forme de réunions en distanciel. De plus, les réunions entre les différents membres de l'équipe se faisaient selon la nécessité. Pendant les périodes de « travaux de finition », les réunions étaient plus rares.

Nous avons utilisé GitLab comme système de gestion des versions et les principes de Scrum en tant que framework de développement. En termes d'organisation à l'intérieur de l'équipe, nous avons adopté le principe standard de division entre back-end et front-end.

• Organisation des équipes

Arkam Rayane

Back-end:

- Création et développement de la classe `Tuiles` avec ses méthodes de base, notamment `affichertuile()` .
- Développement des méthodes pour déplacer un joueur sur le plateau en fonction de la tuile actuelle et vérifier si un chemin a été emprunté.
- Implémentation de l'algorithme MinMax pour le bot `BotTsuro` et ajout de méthodes associées.
- Rédaction et mise en page du rapport.

Front-end :

- Conception et développement du menu du jeu, du thème graphique.
- Ajout de l'entièreté des ressources graphiques.
- Finalisation du design global de l'accueil au menu, ajout d'effets visuels pour l'ergonomie.
- Intégration du son dans le jeu et amélioration des ressources graphiques pour les différentes phases du jeu.

Sifaks Lounici

Back-end:

- Création des fonctions `JoueurPerdu()` et `reinitialiserPlateau()` , ainsi que des getters et setters pour les classes `Joueur` et `PlateauTuiles` .
- Simplification de la lisibilité des classes et arguments, ajout de commentaires JavaDoc.
- Implémentation de la logique pour la classe `Profile` du joueur, ajout de superpouvoirs et des fonctionnalités associées.
- Gestion de l'égalité (TIE) et sauvegarde des parties avec serialization.
- Finalisation du modèle et gestion des superpouvoirs des joueurs.

Front-end :

- Organisation du code de la vue avec la création des branches, implémentation des classes `Menu` , `Accueil` , et `TsuroGame` .
- Liaison entre les classes `Accueil` et `MainMenu` , simplification de ces classes et ajout de différentes classes pour les boutons du Menu.
- Développement de la classe `AudioManager` pour gérer l'audio, et ajout des ressources audio.
- Ajout d'une fenêtre `Rules` pour expliquer le jeu et implémentation d'un timer fonctionnel pour le lancement des parties.

III. Sprints et Développement

Dejan Kecman

Back-end :

- Création des attributs et méthodes de PlateauTuiles, génération de tuiles avec une idée de backtracking.
- Développement des méthodes pour trouver le nouveau point de sortie du joueur.
- Tests sur le modèle et correction des méthodes de direction.
- Débogage des placements de tuiles et des rotations des tuiles.
- Implémentation des effets visuels pour améliorer l'apparence du gameboard et du menu.

Front-end :

- Interface graphique du plateau et du deck de cartes, tests de mouvement du pion du joueur.
- Correction de l'interface graphique pour afficher les carrés de Tuile en JPanel.
- Liaison du gameboard avec le menu et amélioration des effets visuels.
- Ajout de la JavaDoc pour les méthodes et les classes.

Naim Cherchour

Back-end :

- Génération des tuiles avec l'idée Random, extraction et rotation des sprites.
- Finalisation de la génération des tuiles avec Random.
- Création de l'énumération Direction et mise à jour de la position du Joueur en utilisant DIRECTION.
- Développement de la relation Controller/View pour les actions principales (poser une tuile et rotation).
- Sauvegarde d'une partie avec serialization et gestion de l'égalité (TIE).
- Tests de la génération des tuiles et des méthodes associées.

Front-end :

- Dessin des tuiles et des pions des joueurs, développement du plateau avec MultiJoueurs.
- Application du MVC avec le pattern Observer.
- Création d'une méthode pour dessiner une tuile à partir d'un set de sprites, ajustement des positions des sprites et rotation des images.
- Amélioration de la classe DessinateurDeTuile en utilisant une formule de calcul au lieu de if..else.
- Finalisation du plateau et ajout de nouvelles scènes pour la vue.

Sami Chioukh

Back-end :

- Développement de la méthode `estCheminValide()` pour la classe `Tuile`, adaptation des fonctions de Tuiles pour les chemins, et ajout de la classe interne `Chemin`.
- Ajout des fonctions permettant de donner le joueur ayant le plus long chemin.
- Tests du modèle pour vérifier le bon déplacement du pion du joueur et tester tous les cas possibles (Hors Bords, Collisions, Rotation).
- Mise en place de la relation `Controller/View` pour les actions principales (poser une tuile et rotation).
- Implémentation du mode réseau et ajout de la gestion des profils de joueurs.

Front-end:

- Création de la vue et des règles du jeu, adaptation des fonctions de Tuiles pour les chemins.
- Ajout d'un attribut `prenom` dans la classe `Joueur` et son getter.
- Développement de la page des règles et des boutons pour le son.
- Ajout d'une nouvelle scène pour sélectionner le mode de jeu et création de la classe `GameBoardUI`

À travers ces sprints, notre équipe a réussi à appliquer les principes du MVC pour structurer le projet efficacement, améliorer continuellement l'interface utilisateur, et développer un bot fonctionnel pour enrichir l'expérience de jeu. Chaque membre a contribué à des aspects spécifiques du projet, permettant une progression collaborative et une résolution des défis techniques. Notre méthodologie agile a favorisé une adaptation rapide aux besoins et aux défis, et l'utilisation de GitLab a facilité une intégration continue et une collaboration efficace.

GESTION DES BRANCHES

Dans notre projet, nous avons utilisé GitLab pour la gestion des versions, ce qui a permis une collaboration efficace et une organisation structurée de notre travail. Voici un résumé de notre stratégie de gestion des branches et des différentes opérations effectuées :

Stratégie de branches

Nous avons adopté une stratégie de branches basée sur Gitflow, en utilisant plusieurs branches pour gérer les différentes étapes du développement et les fonctionnalités spécifiques. Voici un aperçu des principales branches que nous avons utilisées :

- master : Branche principale contenant les versions stables et prêtes pour la production.
- develop : Branche de développement intégrant les nouvelles fonctionnalités avant leur fusion dans master.
- Tuile : Branche dédiée au développement et à la gestion des tuiles du jeu.
- Menu : Branche dédiée au développement de l'interface utilisateur, notamment le menu principal et les différentes pages de l'application.
- Plateau : Branche dédiée au développement du plateau de jeu, y compris la génération et l'affichage des tuiles.
- Reseau : Branche dédiée à l'implémentation des fonctionnalités de réseau pour permettre le jeu en ligne.
- Bot : Branche dédiée à la création et à l'amélioration de l'intelligence artificielle pour le bot du jeu.
- Audio : Branche dédiée à la gestion et à l'intégration des éléments sonores du jeu.

Fusion des branches

Les fusions entre les branches ont été effectuées régulièrement pour intégrer les nouvelles fonctionnalités et assurer la cohérence du code. Voici quelques exemples de fusions importantes :

- Merge Plateau into Tuile : Fusion de la branche Plateau dans Tuile pour intégrer les modifications liées au plateau de jeu.
- Merge Menu into develop : Fusion de la branche Menu dans develop pour intégrer les modifications de l'interface utilisateur.
- Merge Tuile into Reseau : Fusion de la branche Tuile dans Reseau pour intégrer les modifications liées aux tuiles dans la branche de développement réseau.

Gestion des commits

Les commits ont été réalisés de manière régulière et descriptive pour documenter chaque étape du développement.

• Tests

Des tests unitaires ont été effectués pour des fonctions simples en testant d'abord des fonctions très basiques puis des fonctions plus globales qui réutilisent les petites fonctions.

Les tests sont dans leur propre package pour ne pas être inclus dans le .jar final.

Les classes de test sont toujours nommées de la même façon, ClassNameTest, les fonctions sont nommées testFunctionName (+ un numéro à la fin s'il y a plusieurs tests pour 1 seule fonction).

Les tests ne sont pas exhaustifs, mais ils nous ont quand même permis d'identifier plusieurs soucis, parfois rares et donc difficile à détecter autrement. Ils permettent en outre la non régression sur de petites briques de code ainsi que la non modification du modèle qui est fonctionnelle.

Le GUI n'a pas été testé en mode automatique. Par contre, les tests de certaines fonctions dans GUI ont été faits directement dans le code. Nous avons testé ainsi :

- Le temps d'exécution des fonctions d'affichage des points, des murs, des graphiques (tous séparément/indépendamment).
- Fonctionnement des boutons et des menubars.
- La synchronisation des processus dans le model et dans la vue (par exemple pour la propagation de maladie).

La plupart de ces tests ont été supprimés après la vérification, les restes sont commentés dans le code.

GESTION DES BRANCHES GIT ET STRATÉGIE DE VERSIONNEMENT

Gestion des branches Git et stratégie de versionnement.

Dans notre projet, nous avons utilisé GitLab pour la gestion des versions, ce qui a permis une collaboration efficace et une organisation structurée de notre travail. Voici un résumé de notre stratégie de gestion des branches et des différentes opérations effectuées :

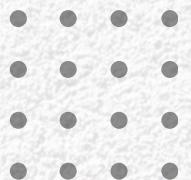
Stratégie de branches

Nous avons utilisé plusieurs branches pour gérer les différentes étapes du développement et les fonctionnalités spécifiques. Voici un aperçu des principales branches que nous avons utilisées :

- master : Branche principale contenant les versions stables et prêtes pour la production.
- develop : Branche de développement intégrant les nouvelles fonctionnalités avant leur fusion dans master.
- Tuile : Branche dédiée au développement et à la gestion des tuiles du jeu.
- Menu : Branche dédiée au développement de l'interface utilisateur, notamment le menu principal et les différentes pages de l'application.
- Plateau : Branche dédiée au développement du plateau de jeu, y compris la génération et l'affichage des tuiles.
- Reseau : Branche dédiée à l'implémentation des fonctionnalités de réseau pour permettre le jeu en ligne.
- Bot : Branche dédiée à la création et à l'amélioration de l'intelligence artificielle pour le bot du jeu.
- Audio : Branche dédiée à la gestion et à l'intégration des éléments sonores du jeu.

Les fusions entre les branches ont été effectuées régulièrement pour intégrer les nouvelles fonctionnalités et assurer la cohérence du code. Voici quelques exemples de fusions importantes :

- Merge Plateau into Tuile : Fusion de la branche Plateau dans Tuile pour intégrer les modifications liées au plateau de jeu.
- Merge Menu into develop : Fusion de la branche Menu dans develop pour intégrer les modifications de l'interface utilisateur.
- Merge Tuile into Reseau : Fusion de la branche Tuile dans Reseau pour intégrer les modifications liées aux tuiles dans la branche de développement réseau.



Gestion des conflits

Des conflits de fusion ont été résolus au fur et à mesure qu'ils se présentaient pour maintenir la stabilité et la cohérence du code. Par exemple, lors de la fusion de Tuile dans develop, certains conflits ont été résolus en ajustant les modifications apportées par les différentes branches.

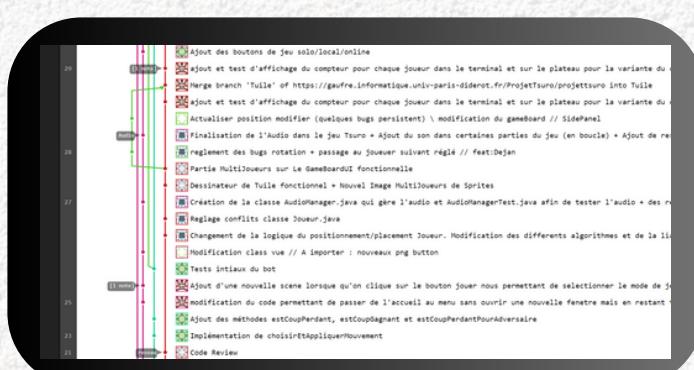
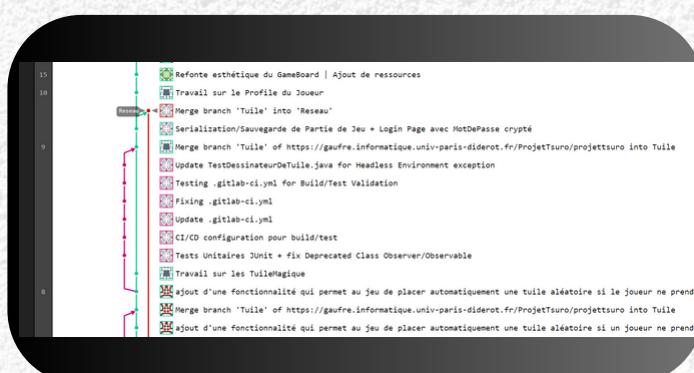
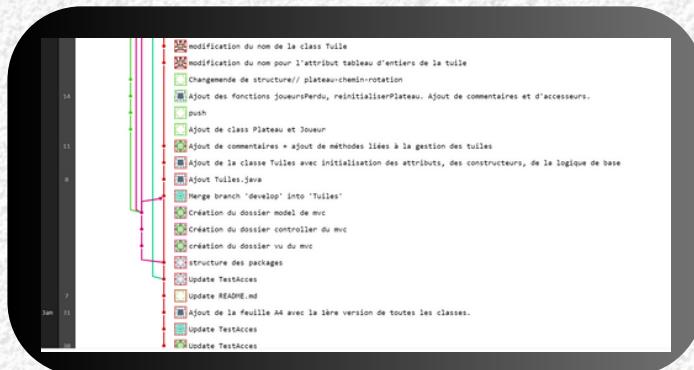
Utilisation de GitLab CI/CD

Nous avons également mis en place une intégration continue (CI/CD) via GitLab pour automatiser les tests et les validations de build. Cela a permis de détecter rapidement les erreurs et d'assurer la qualité du code à chaque étape du développement.

Graphiques de branches

Les graphiques de branches fournissent montrent les différentes branches et leurs interactions tout au long du projet. Ils illustrent les multiples fusions, les créations de branches et les résolutions de conflits. Voici quelques captures d'écran des graphiques de branches pour visualiser notre workflow de gestion des branches :

- Graphique 1 : Début du projet avec les premières fusions et créations de branches
- Graphique 2 : Développement et modifications intermédiaires
- Graphique 3 : Résolution de conflits et finalisation du développement



EXTENSIONS

8 JOUEURS EN LOCAL

L'extension "8 Joueurs en Local" permet à jusqu'à huit joueurs de participer à une même partie de Tsuro sur une seule machine. Cette fonctionnalité enrichit l'expérience de jeu en permettant à un groupe plus large de personnes de s'affronter, favorisant ainsi des parties plus dynamiques et compétitives. Chaque joueur peut contrôler son pion et interagir avec le plateau de jeu, ajoutant de la complexité stratégique et une dimension sociale accrue au jeu.

LONGEST PATH

L'extension "Longest Path" propose un nouveau mode de jeu où l'objectif est de créer le chemin le plus long possible avec son pion. Cette variante met l'accent sur la planification et l'optimisation des mouvements pour maximiser la distance parcourue. Les joueurs doivent naviguer habilement à travers le plateau pour éviter les impasses et utiliser au mieux chaque tuile disponible pour prolonger leur chemin.

TUILLES MAGIQUES

La fonctionnalité des "Tuiles Magiques" introduit une tuile dorée qui peut apparaître dans certains decks de 3 tuiles. Dans le mode de jeu normal, cette tuile permet au joueur de sauter le tour d'un autre joueur, ce qui peut être utilisé pour temporiser et ajuster sa stratégie. Pour le mode "Longest Path", la tuile dorée permet de placer deux tuiles en un seul tour, permettant ainsi de parcourir une plus grande distance. Cette extension ajoute une dimension supplémentaire de hasard et de stratégie, rendant le jeu encore plus captivant et imprévisible.

SAUVEGARDE

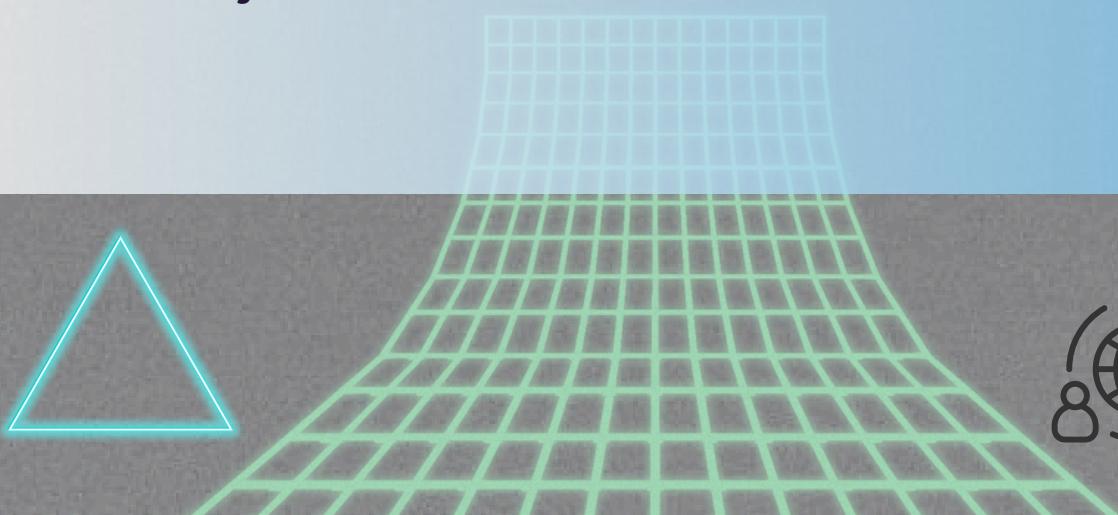
La fonctionnalité de "Sauvegarde" permet aux joueurs de sauvegarder leur progression dans une partie et de la reprendre plus tard. Cette extension est particulièrement utile pour les parties longues ou lorsque les joueurs doivent interrompre le jeu pour une raison quelconque. Les données de la partie, y compris l'état du plateau et les positions des pions, sont enregistrées, permettant une reprise fluide du jeu exactement là où il a été laissé.

RÉSEAU & TUILES TRIANGULAIRES INACHEVÉS

Dès le début du projet, nous avions la volonté d'intégrer un mode réseau pour notre jeu, afin de permettre aux joueurs de se connecter et de jouer ensemble en ligne. Cette fonctionnalité visait à enrichir l'expérience utilisateur en ajoutant une dimension sociale et compétitive au jeu.

Pour structurer ce travail, nous avons créé une branche spécifique, Reseau, dédiée au développement de ces fonctionnalités. Nous avons réussi à créer un serveur fonctionnel et à connecter les clients, ce qui a constitué une étape importante. Cependant, nous n'avons pas pu finaliser la transmission de données entre le serveur et les clients. Malgré notre motivation et nos efforts, plusieurs défis techniques et contraintes de temps nous ont empêchés de finaliser cette fonctionnalité avant la fin du projet.

Nous avions également l'ambition d'introduire des tuiles triangulaires dans le jeu. Cette extension aurait apporté une nouvelle dynamique et complexité au gameplay, en modifiant la manière dont les chemins se connectent et se développent. Cependant, l'implémentation des tuiles triangulaires nécessitait une refonte complète de la structure du jeu, ce qui impliquait de réécrire l'ensemble du code de base. En raison des contraintes de temps et des ressources limitées, nous n'avons pas pu intégrer cette fonctionnalité dans la version actuelle du jeu.



Conclusion



Nous avons fait preuve de patience et de camaraderie, des atouts précieux dans l'élaboration de notre projet. La répartition équilibrée du travail et l'assiduité de chacun aux réunions ont permis un développement dans des conditions optimales. Cette collaboration harmonieuse a été essentielle pour surmonter les défis techniques et respecter les délais, tout en maintenant un environnement de travail positif et productif. Grâce à notre collaboration, nous avons avancé de manière efficace et atteint les objectifs fixés pour ce module. Cela incluait la réalisation de recherches bibliographiques, la conception d'algorithmes, la compréhension de l'architecture et des concepts de la programmation orientée objet, ainsi que la simplification du code. Tout cela a contribué à enrichir notre expérience et nos compétences en développement logiciel.