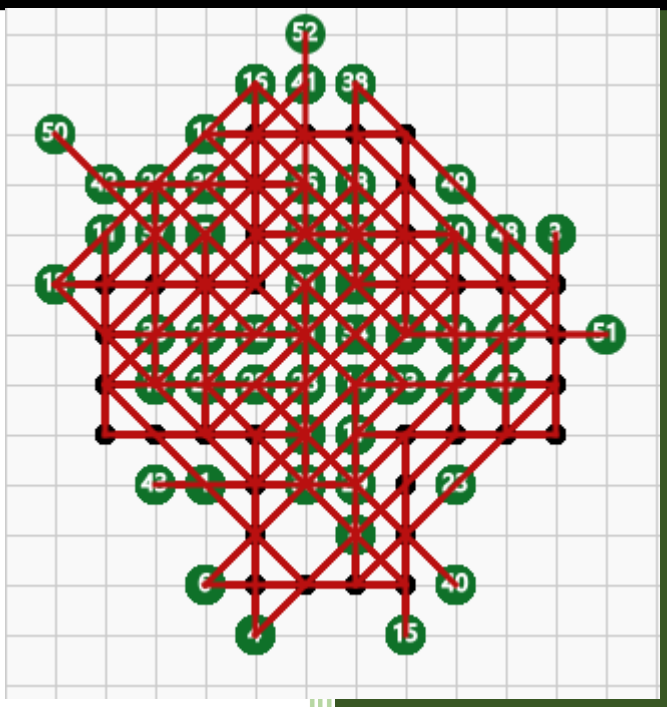
Groupe :
freenestislepa

Documentation développeur
Morpion Solitaire



Mustafa Caglayan
Ayoub Moqrich
2023-2024

Table des matières

I. Introduction

Ce programme a été élaboré dans le cadre du projet de Java avancé, une composante essentielle du cursus de Master 1 en Méthodes Informatiques Appliquées à la Gestion des Entreprises (Miage). L'objectif central de ce projet consiste à concevoir une interface permettant aux utilisateurs de s'immerger dans le Morpion Solitaire tout en intégrant des algorithmes sophistiqués pour résoudre ce jeu captivant.

II. Fonctionnalités offertes par le jeu

Principales fonctionnalités :

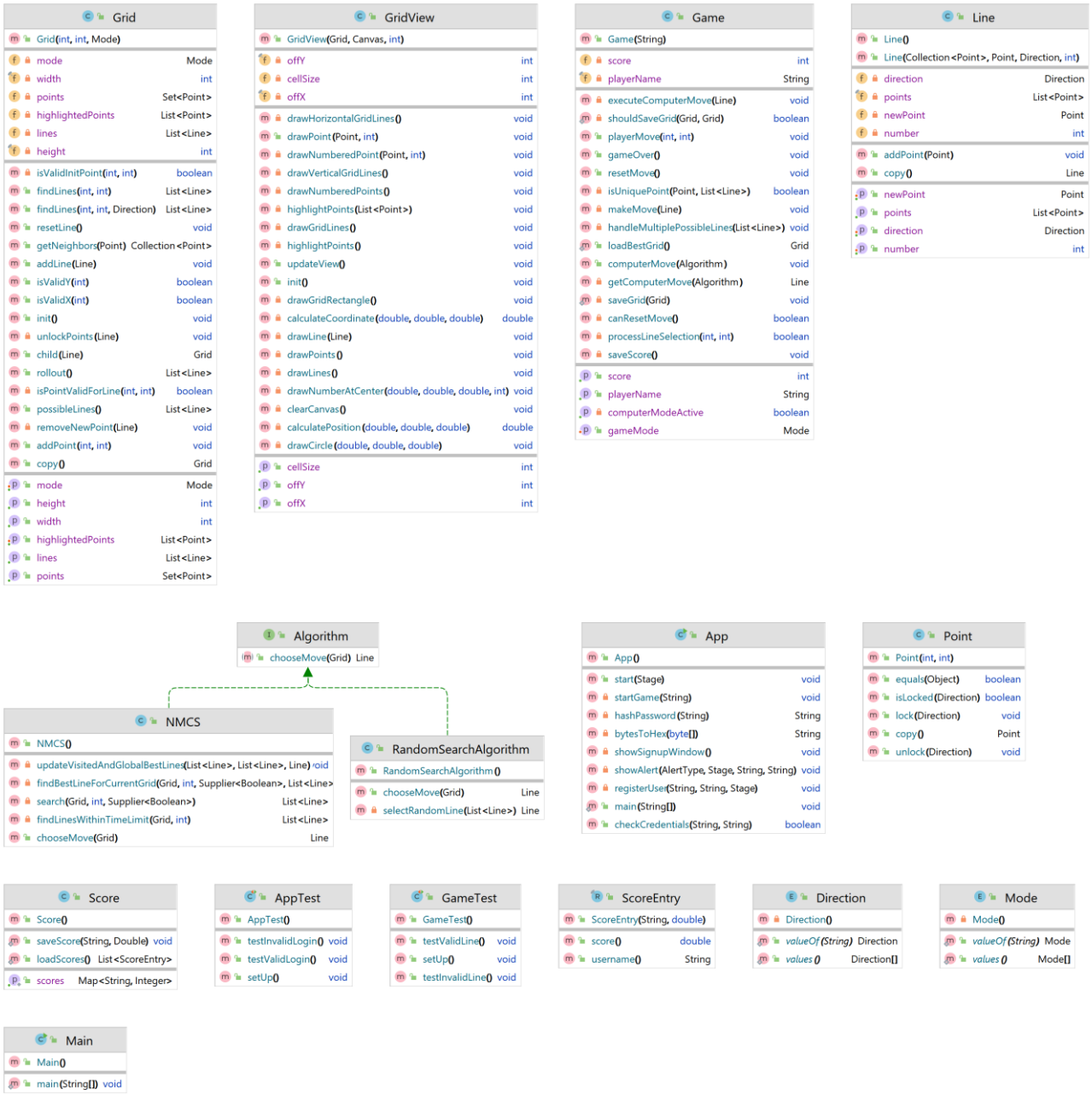
1. **Implémentation d'un moteur de jeu 5D et 5T :** Ce programme propose une implémentation robuste d'un moteur de jeu permettant aux utilisateurs de jouer au Morpion Solitaire dans les versions 5D (avec recouvrement) et 5T (sans recouvrement). La flexibilité du moteur offre une expérience de jeu variée et stimulante.
2. **Méthode de recherche de solution automatique aléatoire :** Pour enrichir la jouabilité et évaluer différentes approches algorithmiques, une méthode de recherche de solution automatique aléatoire a été créée. Cela offre aux joueurs la possibilité de découvrir des stratégies alternatives et d'améliorer leurs compétences.
3. **Interface graphique intuitive :** Une interface graphique intuitive a été développée pour faciliter l'interaction et l'observation du jeu. Cette interface offre une expérience utilisateur fluide et esthétique, rendant le Morpion Solitaire accessible à un large public.

Extensions :

1. **Tableau des Scores :** Intégration d'un tableau des scores pour la partie interactive, permettant d'enregistrer les meilleures grilles dans des fichiers. Cette fonctionnalité enrichit la compétitivité et offre un moyen de suivre les performances des joueurs.
2. **Algorithme NMCS :** L'implémentation de l'algorithme NMCS vise à améliorer les capacités de recherche de solutions du jeu. Cette extension offre une approche avancée pour résoudre le Morpion Solitaire, relevant le défi stratégique.
3. **Diversité d'Algorithmes :** Proposition et mise en œuvre d'autres algorithmes de recherche de solutions. Cette fonctionnalité étend la variété des approches possibles, offrant aux utilisateurs la possibilité d'explorer différentes méthodes de résolution.
4. **Multi-threading :** Utilisation du multi-threading pour accélérer les algorithmes de recherche. Cette extension améliore les performances du programme, garantissant une résolution rapide et efficace du Morpion Solitaire.
5. **Plateaux de Jeu Personnalisés :** Offrir la possibilité de jouer sur des plateaux de jeu différents (5T# ou 5D#). Cette fonctionnalité permet aux joueurs de personnaliser leur expérience de jeu en choisissant des configurations spécifiques, ajoutant une dimension créative au Morpion Solitaire.

III. Architecture

L'architecture globale du projet Morpion Solitaire repose sur le modèle MVC (Modèle-Vue-Contrôleur), une approche de conception qui sépare les différentes responsabilités du logiciel. Cette division permet une meilleure organisation du code, une maintenance facilitée, et favorise la réutilisation des composants. Voici une description plus détaillée de chaque composant de l'architecture :



Classe UserController :

- *Responsabilités principales :*
    - Configuration et initialisation du jeu.
    - Gestion des entrées utilisateur telles que les clics de souris.

- Réaction aux événements de l'interface utilisateur tels que la réinitialisation ou le changement de mode de jeu.
- Contrôle de la simulation automatisée.
- Affichage de l'historique des scores et de la meilleure grille.
- Mise à jour de la vue GridView pour refléter les changements d'état du jeu.

Classe Main :

- Point d'entrée de l'application JavaFX.
- Définit le package et contient la méthode main() qui est le point d'entrée de l'application Java.
- Lance la classe Application (App.class) en appelant Application.launch().
- Passe les éventuels arguments de ligne de commande à la méthode launch.

Classe App :

- Classe principale qui gère le lancement de l'application Morpion Solitaire.
- Contient les méthodes start(), startGame(), showSignupWindow(), registerUser(), checkCredentials(), hashPassword(), bytesToHex(), et main().
- La méthode start() est appelée par JavaFX pour lancer l'application et configure l'interface de connexion initiale.
- La méthode startGame() lance l'interface utilisateur réelle du jeu après une connexion réussie et charge le fichier FXML et le contrôleur du jeu.
- La méthode showSignupWindow() ouvre une fenêtre contextuelle pour permettre l'inscription de l'utilisateur.
- La méthode registerUser() enregistre un nouvel utilisateur dans le fichier users.txt lors de l'inscription.
- La méthode checkCredentials() vérifie les informations d'identification de connexion par rapport au fichier users.txt.
- Les méthodes hashPassword() et bytesToHex() sont des méthodes d'aide pour le hachage sécurisé des mots de passe.
- La méthode main() est le point d'entrée standard de Java qui lance l'application.

En résumé, le projet Morpion Solitaire suit une architecture MVC (Modèle-Vue-Contrôleur) où la classe UserController agit comme le contrôleur qui relie le modèle de jeu à l'interface utilisateur (vue). La classe App est responsable du lancement de l'application et de la gestion de la connexion et de l'inscription des utilisateurs. Le fichier pom.xml fournit la configuration et les dépendances du projet pour permettre à Maven de construire et gérer le projet.

IV. Algorithme de Recherche de Solution Aléatoire

1) Random Search

L'algorithme de recherche de solutions aléatoires est une approche simple et fondamentale pour résoudre des problèmes. Voici comment il est mis en œuvre :

1. **Génération de Solutions Aléatoires :** L'algorithme commence par générer une solution initiale de manière aléatoire. Cette solution peut représenter une configuration initiale du jeu ou une solution partielle.
2. **Évaluation des Solutions :** Chaque solution générée est évaluée en fonction de critères spécifiques. Dans le cas du jeu Morpion Solitaire, cela peut être le nombre de lignes complétées ou le score obtenu.
3. **Recherche de la Meilleure Solution :** L'algorithme itère sur un certain nombre d'itérations ou jusqu'à ce qu'une condition de terminaison soit atteinte. À chaque itération, une nouvelle solution est générée de manière aléatoire et évaluée. Si cette nouvelle solution est meilleure que la précédente, elle est conservée.

En résumé, l'algorithme de recherche de solutions aléatoires est une méthode simple et efficace pour résoudre le problème du Morpion Solitaire. Il utilise une approche itérative pour générer des solutions aléatoires, évalue les performances de chaque solution, et recherche la meilleure solution possible.

2) NMCS (Nested Monte Carlo Search)

L'algorithme NMCS (Nested Monte Carlo Search) est implémenté dans la classe NMCS, qui respecte l'interface Algorithm. Cet algorithme est utilisé pour prendre des décisions automatiques dans le jeu Morpion Solitaire en évaluant différentes lignes possibles.

a) Fonctionnement Général

L'algorithme NMCS utilise une approche de recherche Monte Carlo avec un niveau de recherche imbriqué. Voici une explication détaillée de son fonctionnement :

- **Méthode chooseMove(Grid grid) :** Cette méthode est appelée pour choisir le prochain mouvement à effectuer sur la grille. Un niveau de recherche initial (**searchLevel**) est défini à 2, et un temps maximal d'exécution (**maxRunningTimeMs**) est limité à 1000 millisecondes (1 seconde). L'algorithme recherche les meilleures lignes possibles en utilisant la méthode **findLinesWithinTimeLimit**.
- **Méthode findLinesWithinTimeLimit(Grid grid, int level) :** Limite la recherche dans le temps en utilisant un mécanisme de gestion du temps (**System.currentTimeMillis()**). Appelle la méthode **search** pour effectuer la recherche en profondeur et récupère les lignes résultantes.
- **Méthode search(Grid grid, int depth, Supplier<Boolean> isCanceled) :** La recherche est effectuée récursivement jusqu'à la profondeur spécifiée (**depth**). Pour chaque ligne possible sur la grille, l'algorithme effectue une recherche en profondeur en appelant la méthode **search** avec une profondeur réduite. Les résultats des différentes branches de la recherche sont évalués, et la meilleure ligne est choisie.
- **Méthode findBestLineForCurrentGrid(Grid grid, int depth, Supplier<Boolean> isCanceled, List<Line> globalBestLines, List<Line> visited) :** Pour une grille donnée, cette méthode

recherche la meilleure ligne possible en évaluant les différentes lignes. Elle garde une trace des meilleures lignes rencontrées à ce niveau et les renvoie.

- **Méthode updateVisitedAndGlobalBestLines(List<Line> visited, List<Line> globalBestLines, Line curBestLine) :** Met à jour la liste des lignes visitées et la liste des meilleures lignes globales.

L'algorithme NMCS est utilisé pour prendre des décisions automatiques lorsqu'un mouvement doit être choisi par l'ordinateur dans le jeu Morpion Solitaire. La méthode **chooseMove** retourne la ligne choisie par l'algorithme.

En conclusion, l'algorithme NMCS est une implémentation sophistiquée basée sur la recherche Monte Carlo avec une approche de recherche en profondeur. Il permet au jeu Morpion Solitaire de prendre des décisions intelligentes et offre une expérience de jeu automatisée aux utilisateurs. La flexibilité de l'algorithme permet des ajustements fins en modifiant les paramètres tels que la profondeur de la recherche.

V. Interface Graphique (GUI)

1. Structure du Code

Le code de l'interface utilisateur est organisé autour de deux classes principales : UserController et GridView.

UserController

La classe UserController gère les interactions entre l'utilisateur et le jeu.

GridView

La classe GridView est responsable du rendu graphique de la grille de jeu sur le canvas. Elle fournit des méthodes pour initialiser, mettre à jour, et afficher les éléments graphiques du jeu.

L'interface utilisateur graphique repose sur un canvas où la grille de jeu est rendue visuellement. Voici les composants principaux de l'interface :

- **Canvas:** Utilisé pour afficher la grille de jeu et les éléments graphiques associés.
- **ComboBoxes:** Permettent à l'utilisateur de sélectionner le mode de jeu (5T ou 5D) et l'algorithme pour la simulation automatique.
- **Boutons:** Utilisés pour des actions telles que la simulation automatique, l'annulation de mouvement, l'affichage de l'historique, et l'affichage de la meilleure grille.

2. Mise en Œuvre

L'interaction entre la classe **UserController** et **GridView** est soigneusement orchestrée pour assurer une séparation claire entre la logique du jeu et l'interface utilisateur. Les méthodes de **UserController** déclenchent les actions du jeu, tandis que **GridView** se charge du rendu graphique.

Les coordonnées de la grille sont converties à partir des coordonnées du canvas, permettant une interaction fluide avec l'interface utilisateur. Les méthodes de **GridView** sont appelées pour dessiner les points, les lignes, et d'autres éléments graphiques associés au jeu.

Le fichier FXML, nommé **morpion_solitaire.fxml**, définit la structure de l'interface utilisateur graphique en utilisant le modèle **BorderPane** de JavaFX. Voici une description des éléments clés :

Structure

- **BorderPane:** Conteneur principal de l'interface, organisé en haut, bas, gauche, droite, et centre.
- **Centre:** Une **ScrollPane** contenant un **VBox**, centré et contenant le **Canvas** pour afficher la grille de jeu.
- **Gauche: VBox** aligné à gauche, contenant des éléments pour le menu et les actions du joueur.
- **Droite:** (actuellement vide)

Éléments Graphiques

- **Canvas (fx:id="canvas"):** Utilisé pour afficher visuellement la grille de jeu. Les événements de souris sont associés à la méthode **mousePressed** du contrôleur.
- **Labels, ComboBoxes, et Buttons:** Utilisés pour les options du jeu, le choix de l'algorithme, les actions du joueur (Reset, Undo, Simulate), et l'affichage de l'historique des scores et de la meilleure grille.
- **ScrollPane:** Permet le défilement si la taille du contenu dépasse celle de la fenêtre.

Contrôleur associé (fx:controller="org.netislepafree.morpion_solitaire.controller.UserController")

La classe **UserController** est le contrôleur associé au fichier FXML. Elle gère les interactions utilisateur-jeu et est responsable de l'initialisation et du contrôle de l'état du jeu.

VI. Améliorations Futures

Dans cette section, nous explorons quelques pistes d'améliorations potentielles pour le projet Morpion Solitaire, ouvrant la voie à des fonctionnalités et optimisations plus avancées.

1. Intégration d'un Système de Tournois

Une évolution intéressante serait d'implémenter un système de tournois qui permettrait aux utilisateurs de s'affronter dans des compétitions Morpion Solitaire. Cela pourrait inclure un tableau des scores en ligne, des classements, et des récompenses pour les performances exceptionnelles.

2. Intelligence Artificielle Évoluée

Affiner et améliorer les algorithmes de recherche de solutions, en particulier l'algorithme NMCS, pourrait considérablement accroître la qualité de jeu de l'intelligence artificielle. Des approches plus avancées, telles que l'apprentissage automatique, pourraient être explorées pour rendre l'IA encore plus compétitive.

3. Plateaux de Jeu Personnalisables

Offrir aux joueurs la possibilité de créer leurs propres plateaux de jeu personnalisés ajouterait une dimension créative au Morpion Solitaire. Cela impliquerait la conception d'une interface conviviale permettant de définir la disposition initiale des points et des traits.

4. Mode Multijoueur en Temps Réel

Intégrer un mode multijoueur en temps réel permettrait aux joueurs de s'affronter en ligne, offrant ainsi une expérience de jeu sociale. Cela nécessiterait la mise en place d'un système de gestion de sessions, de synchronisation des mouvements et d'une interface utilisateur optimisée pour le jeu en réseau.

En explorant ces évolutions potentielles, le projet Morpion Solitaire pourrait élargir son attrait, offrant aux utilisateurs une expérience de jeu enrichie et stimulante. Ces suggestions représentent des pistes prometteuses pour le développement futur du projet.

VII. Difficultés Rencontrées et Répartition des Tâches

1. Difficultés Rencontrées

Le développement du projet Morpion Solitaire a été enrichissant, mais il n'a pas été exempt de défis significatifs. Parmi les difficultés majeures, nous avons rencontré des obstacles lors de l'implémentation de l'interaction entre l'affichage graphique et le modèle du jeu.

L'intégration harmonieuse entre la représentation graphique de la grille de jeu et la logique du jeu s'est avérée être un défi complexe. La synchronisation précise entre les actions de l'utilisateur sur l'interface graphique et les réponses appropriées du modèle du jeu a nécessité une attention particulière. Trouver un équilibre entre la convivialité de l'interface utilisateur et la précision des règles du jeu a été un processus itératif qui a demandé une réflexion approfondie.

2. Répartition des Tâches

Pour surmonter ces défis, une répartition claire des tâches a été établie au sein de l'équipe de développement :

- **A deux :** Logique sous-jacents du jeu, modele MCV et cœur du code des controlers et des fonctions pour construires les lignes.
- **Ayoub : Logique du Jeu et Modèle de Jeu** Ayoub s'est concentré sur l'améloration de la logique du jeu et en améliorant le modèle de jeu. Il a travaillé sur la définition des règles, la gestion des mouvements, et la création d'un modèle pour trouver les lignes en factorisant au maximum les fonctions. Il a aussi fait les tests unitaires et la connexion du joueur.
- **Mustafa : Interface Graphique et Algorithme de Recherche** Mustafa a pris en charge le volet visuel du projet en concevant et mettant en œuvre l'interface graphique. Il a également contribué à l'implémentation de l'algorithme de recherche, en particulier l'algorithme NMCS, pour permettre au jeu de prendre des décisions automatiques intelligentes.

Cette division des responsabilités a permis une concentration efficace sur des domaines spécifiques du projet, exploitant ainsi les forces individuelles de chaque membre de l'équipe. La collaboration étroite entre Ayoub et Mustafa a été cruciale pour assurer la cohérence entre la logique du jeu et son représentation visuelle.

En dépit des difficultés, cette approche collaborative a conduit à la création d'un Morpion Solitaire fonctionnel, combinant avec succès la complexité de la logique de jeu et la convivialité de l'interface graphique. Les leçons tirées de ces défis ont contribué à renforcer l'expertise de l'équipe et ont enrichi l'expérience globale de développement du projet.