Compte rendu SAE31_2023 : Dorfromantik

Auteurs : Amir Kabbouri - Tarehi Zaabay - Athiran Nagathurai

Table de matières

1. Introduction

Dans le cadre de ce projet, un jeu de stratégie inspiré de Dorfromantik a été développé. L'objectif principal est d'offrir au joueur la possibilité d'assembler des tuiles hexagonales représentant divers terrains pour créer un paysage harmonieux. Chaque tuile peut être placée de différentes manières, et le joueur doit décider où et comment la positionner.

Le développement a été effectué en Java, en respectant la contrainte d'utiliser uniquement les outils et bibliothèques fournis par l'API officielle. Les sources du projet sont hébergées sur le serveur Gitea (lien) de notre département, où nous avons également documenté l'avancement du projet ainsi que les contributions individuelles des membres de l'équipe à travers des commits.

Ce rapport présentera dans un premier temps les principales fonctionnalités du jeu ainsi que son fonctionnement. Nous aborderons ensuite l'architecture du programme à l'aide d'un diagramme de classes simplifié, avant d'expliquer l'algorithme utilisé pour calculer les points du joueur selon les zones de terrains. Enfin, nous conclurons par une analyse de l'interface utilisateur et des méthodes employées pour comparer les scores entre joueurs.”

Ce jeu a été développée par :

- Kabbouri Amir
- Nagathurai Athiran
- Zaabay Tarehi

2. Fonctionnalité du programme :

Au demarrage du programme avec la commande make run1, l'écran d'acceuil suivant s'affiche :

L'interface présente un menu principal trois options sont disponibles pour l'utilisateur :

- **Tableau des scores**
- **Comment jouer**
- **Jouer**

Un menu déroulant intitulé **Choisir une série** est également disponible au centre, permettant de sélectionner une série de jeux, "Série 1, Série 2 ou Série 3".

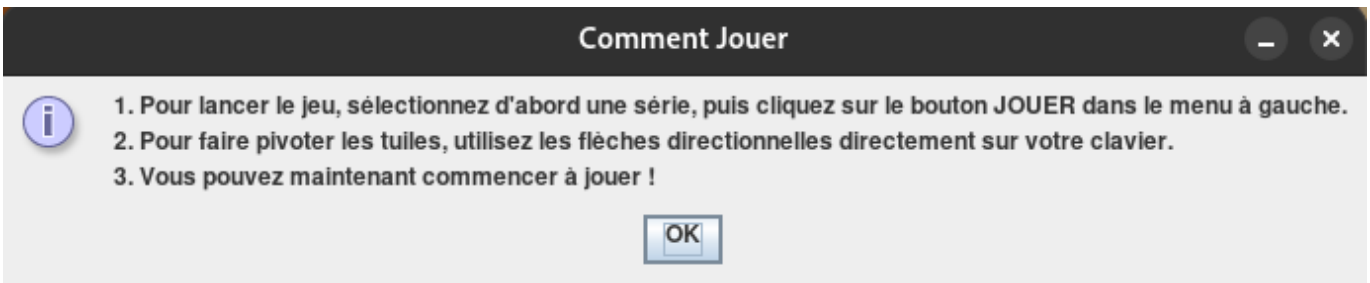
2.1 Tableau des Scores :

Une fenêtre intitulée **Tableau des Scores - Série 1** affiche les meilleurs scores obtenus pour la série en cours. Dans cet exemple, deux scores sont visibles :

Cette fonctionnalité permet aux joueurs de suivre leurs performances et de se comparer aux meilleurs scores enregistrés.

2.2 Comment Jouer :

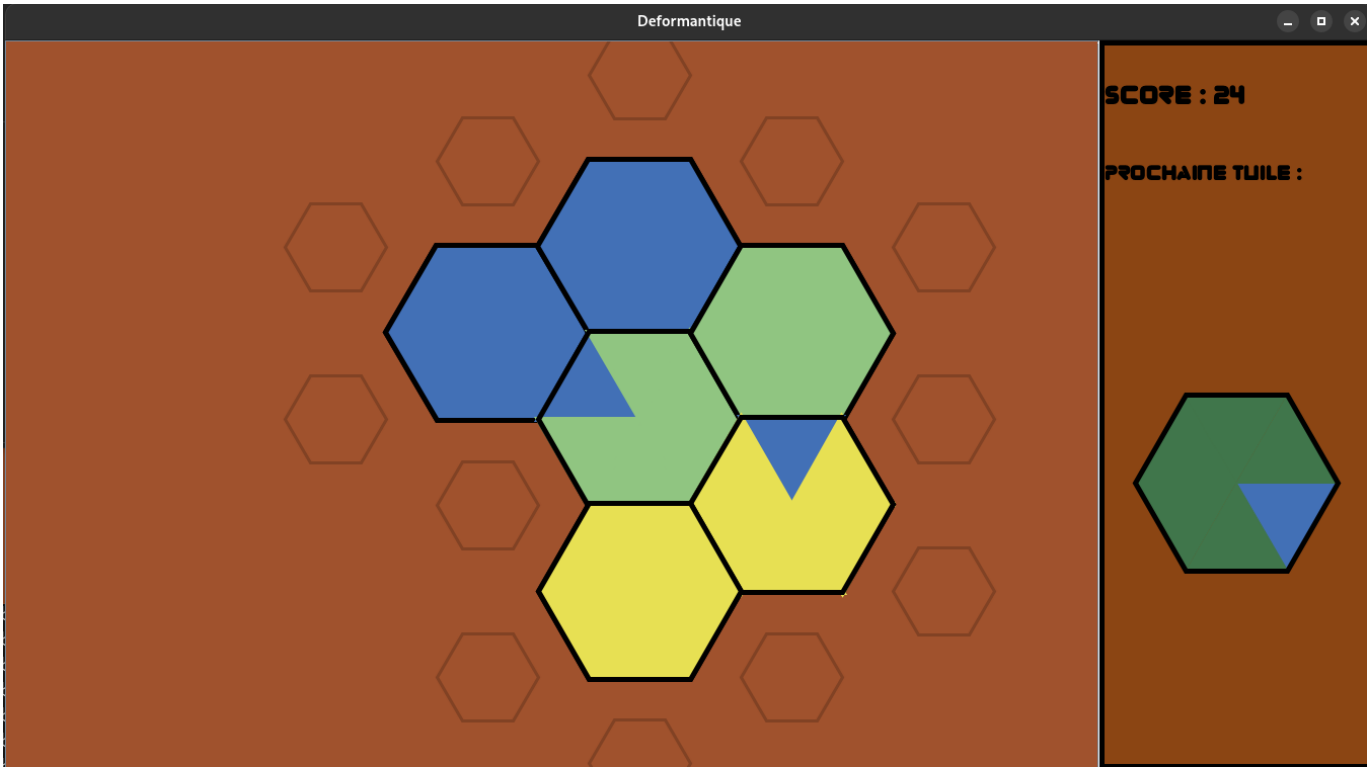
Une fenêtre d'instructions, intitulée **Comment Jouer**, est également accessible. Elle fournit des directives claires sur le fonctionnement du jeu :



2.3 Jeu :

Le déroulement d’une partie se présente ainsi :

1. **Début de la partie** : La première tuile est placée automatiquement.
2. **Tour du joueur** : À chaque tour, une nouvelle tuile est révélée. Le joueur doit alors choisir où la placer et peut la faire pivoter pour optimiser l’agencement du paysage. La seule contrainte est que la nouvelle tuile doit être adjacente à une tuile déjà posée.
3. **Calcul des points** : Le nombre de points pour une poche est calculé en élevant au carré le nombre de tuiles dans cette poche. Par exemple, si une poche contient 2 tuiles, elle vaut $2^2=4$ points. Si une poche contient 3 tuiles, elle vaut $3^2=9$ points, et ainsi de suite.



Dans le jeu, l'utilisateur peut placer une tuile en effectuant un clic gauche sur le panneau latéral de gauche. Ce panneau affiche également le score actuel du joueur ainsi que l'aperçu de la prochaine tuile qui sera posée.

L'utilisateur a la possibilité de choisir l'orientation de la tuile en utilisant les flèches directionnelles gauche et droite du clavier (→ ou ←). De plus, pendant le jeu, il est possible de maintenir le clic gauche en bougeant la souris pour ajuster la position ou l'orientation de la tuile de manière fluide.

3. Structure du code :

L'application est divisée en plusieurs packages principaux pour assurer une organisation modulaire, maintenable et évolutive. Ces packages incluent `controller`, `model`, et `vue`, chacun ayant des responsabilités distinctes. Le code est structuré en fonction de l'architecture MVC (Modèle-Vue-Contrôleur), favorisant la séparation des préoccupations et la réutilisabilité.

Package `model`

Le package `model` contient les classes représentant les données et la logique de l'application :

- **Tile** : Modélise une tuile avec différents types de terrain et configurations. Elle contient les attributs et méthodes pour gérer l'état d'une tuile (types de terrain, orientation, etc.).
- **Serie** et **SerieBD** : Gèrent les séries de tuiles, permettant de regrouper des tuiles par série. La classe `SerieBD` communique avec la base de données pour récupérer les informations des séries.
- **ScoreManager** : Assure la persistance des scores en les enregistrant dans la base de données. Ces classes définissent la structure des données et les opérations métiers de l'application, en centralisant les interactions avec la base de données via JDBC.

  Package `controller`

  Le package `controller` contient les classes de contrôle qui manipulent les données du `model` et mettent à jour la `vue` en fonction des interactions de l'utilisateur :

- **TileController** : Gère les interactions et actions liées aux tuiles dans la `TileView`. Il se charge de calculer le score en fonction des règles définies et de mettre à jour les positions disponibles pour placer une nouvelle tuile.

Cette classe agit comme intermédiaire entre le modèle et la vue, assurant que les règles de gestion des tuiles et du score sont appliquées correctement.

Package `vue`

Le package `vue` contient toutes les classes liées à l'affichage et l'interface utilisateur :

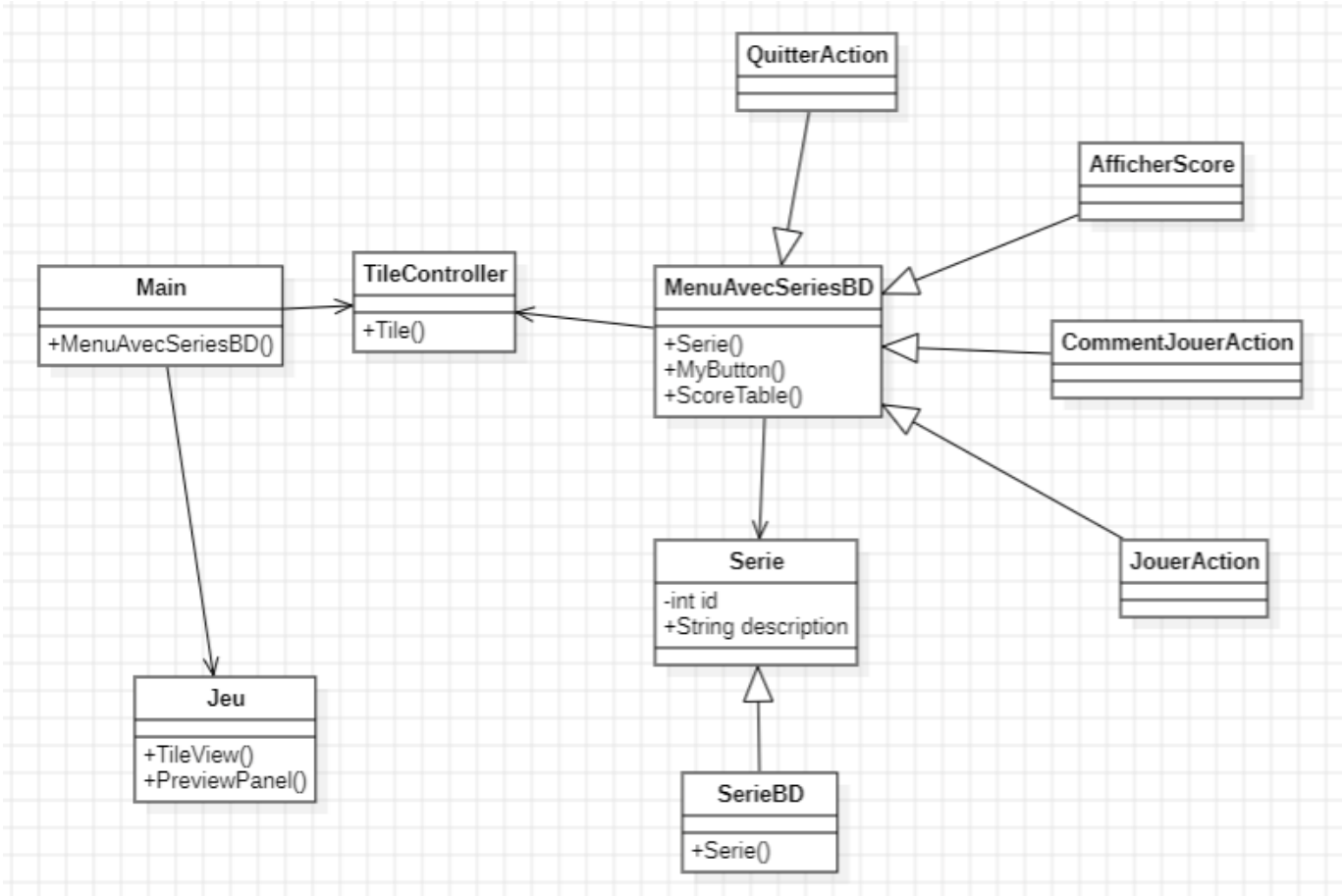
- **TileView** : Responsable de l'affichage de la grille hexagonale des tuiles et de la gestion des emplacements disponibles pour placer de nouvelles tuiles. Cette classe inclut des contrôles pour détecter les interactions de l'utilisateur.
- **MenuAvecSeriesBD** : Interface de sélection de la série de tuiles, permettant à l'utilisateur de démarrer une partie ou de visualiser le tableau des scores.
- **Jeu** : Interface principale du jeu où l'utilisateur peut jouer avec les tuiles et voir son score se mettre à jour. Cette vue inclut des composants comme `PreviewPanel` (prévisualisation de la tuile suivante) et `ScoreTable` (tableau des scores).
- **MyButton** et **BackgroundPanel** : Composants visuels personnalisés pour améliorer l'esthétique et l'expérience utilisateur de l'application. Chaque classe de la `vue` est dédiée à une partie spécifique de l'interface, facilitant ainsi la gestion et la modification de l'apparence sans impacter le modèle ou le contrôleur.

  Architecture MVC

L’application suit l’architecture **Modèle-Vue-Contrôleur (MVC)** :

- Le **Modèle** (`model`) gère les données et la logique métier.
- La **Vue** (`vue`) est responsable de l’affichage et de l’interface utilisateur.
- Le **Contrôleur** (`controller`) traite les interactions utilisateur et applique les règles de gestion en appelant le modèle et en mettant à jour la vue en conséquence. Cette architecture permet de séparer clairement les responsabilités, de rendre le code plus modulable et de simplifier les futures modifications.

Voici le diagramme qui comprend les dependances principales :



3. 2 Structure de la table :

series (id, name, description): Cette table stocke les différentes séries disponibles pour le jeu. Chaque série est identifiée par un id unique, a un name (nom) et une description pour plus de détails. series_tiles (id, series_id, tile_id): Cette table relie chaque série à ses tuiles spécifiques. Chaque entrée représente une tuile dans une série particulière. Le champ series_id est une clé étrangère qui pointe vers la table series, et tile_id est une clé étrangère vers la table tiles.

terrain_types (id, name): Cette table définit les différents types de terrains disponibles dans le jeu. Chaque type de terrain est identifié par un id unique et possède un name pour le type de terrain (ex. : forêt, eau, prairie). tiles (id, terrain1_id, terrain2_id, split_ratio): Cette table contient les tuiles du jeu et leurs caractéristiques. Chaque tuile a un id unique, un terrain1_id et un terrain2_id qui pointent vers les types de terrains dans la table terrain_types. Le champ split_ratio indique le pourcentage de chaque type de terrain présent sur la tuile, utile pour des tuiles mixtes. scores (id, score, date_played, series_id): Cette table enregistre les scores obtenus dans le jeu. Chaque score a un id unique, un score (le nombre de points), et une date_played pour la date de la partie. Le champ series_id est une clé étrangère vers la table series pour indiquer la série à laquelle le score se rapporte.

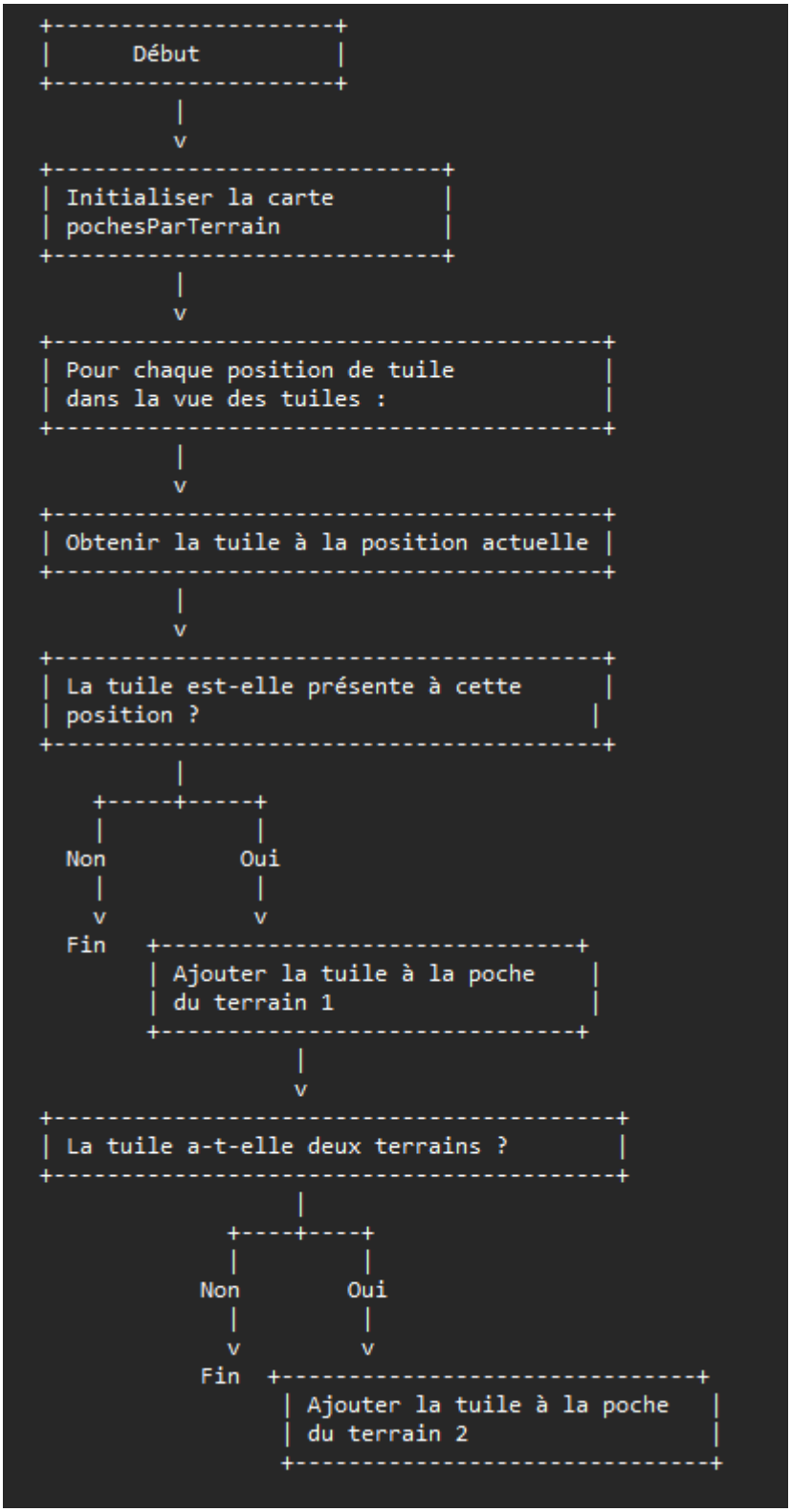
Lisibilité et Simplicité Visuelle :

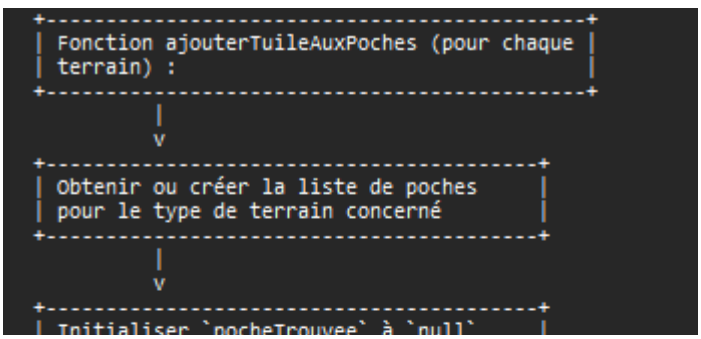
- - Les éléments importants, comme le score et la prochaine tuile, sont clairement visibles sur le côté droit de l’écran, permettant aux joueurs de suivre leur progression et d'anticiper leurs actions sans être distraits par des informations inutiles.
  - La couleur de fond et les contours sombres des tuiles contrastent bien, facilitant la distinction entre les tuiles déjà posées et les emplacements poentiels.
- **Contrôles Intuitifs** :
  - La possibilité de positionner les tuiles avec un clic gauche et d'ajuster leur orientation avec les flèches directionnelles rend les contrôles simples et accessibles, même pour les joueurs novices.
  - Les icônes de navigation en bas (flèches pour rotation, et symboles de zoom) sont facilement identifiables et renforcent l’intuitivité de l’interface, permettant une navigation aisée.
- **Anticipation et Planification** :
  - La section **"Prochaine tuile"** permet au joueur de voir la prochaine pièce à placer, ce qui lui donne l’opportunité de réfléchir à la meilleure stratégie pour optimiser son score.
  - Cette fonctionnalité améliore l’expérience en rendant le jeu plus stratégique et moins basé sur le hasard.
- **Feedback Visuel** :
  - La tuile en surbrillance rouge montre clairement l'emplacement potentiel, indiquant de manière visuelle les actions possibles, ce qui aide les joueurs à comprendre rapidement les choix qui s’offrent à eux.

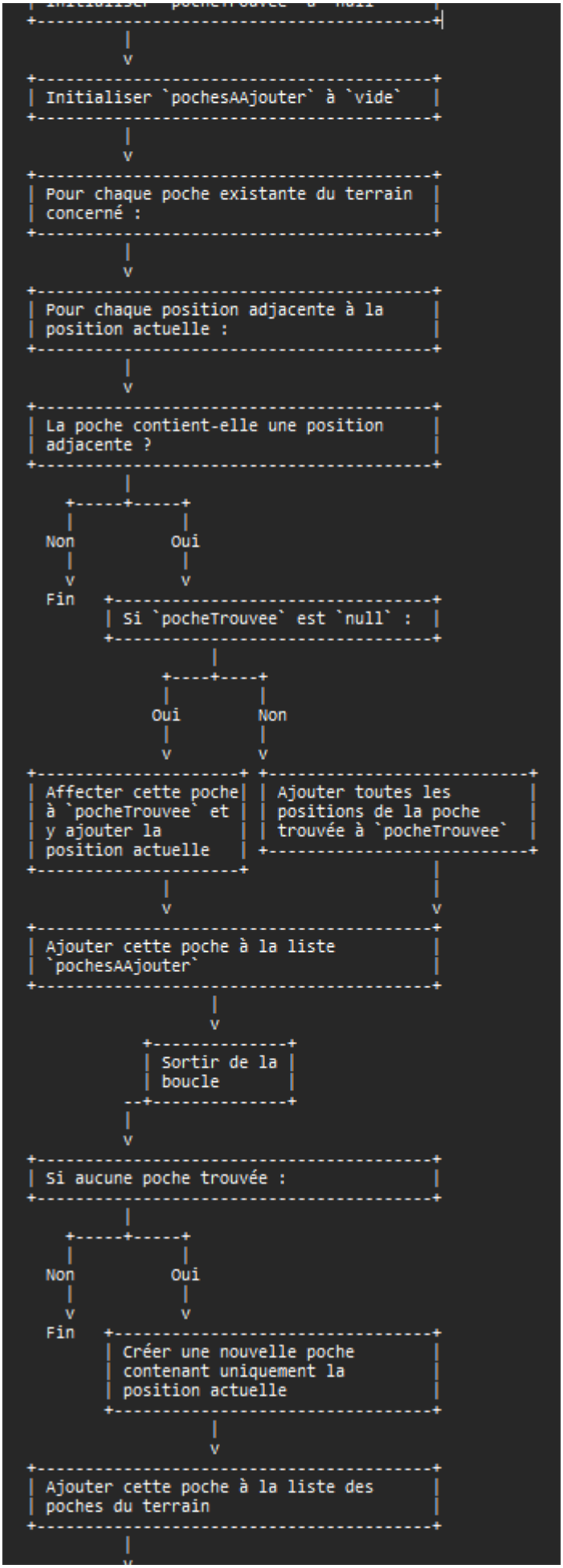
    - Le feedback visuel des tuiles et des zones de placement rend le processus interactif plus fluide et agréable.

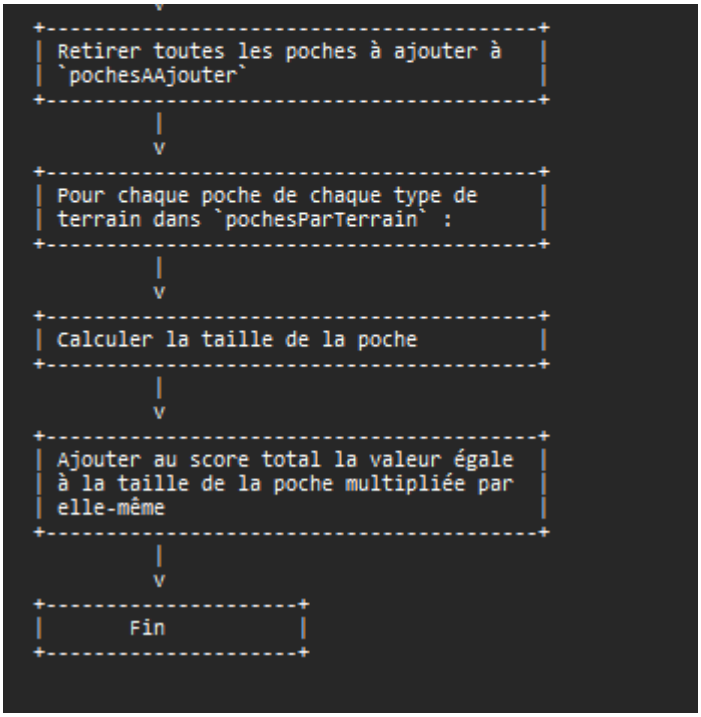
- **Optimisation de l’Espace** :
    - L’espace est utilisé de manière efficace, avec les tuiles bien centrées et les informations annexes regroupées à droite. Cela permet de maintenir un focus sur l'aire de jeu tout en ayant un accès facile aux informations essentielles.

Diagramme d'activités:


Début
Initialiser la carte pochesParTerrain
Pour chaque position de tuile dans la vue des tuiles :
Obtenir la tuile à la position actuelle
La tuile est-elle présente à cette position ?
Non
Fin
Oui
Ajouter la tuile à la poche du terrain 1
La tuile a-t-elle deux terrains ?
Non
Fin
Oui
Ajouter la tuile à la poche du terrain 2


Fonction ajouterTuileAuxPoches (pour chaque terrain) :
Obtenir ou créer la liste de poches pour le type de terrain concerné
Initialiser `pocheTrouvee` à `null`


Initialiser `pochesAAjouter` à `vide`
Pour chaque poche existante du terrain concerné :
Pour chaque position adjacente à la position actuelle :
La poche contient-elle une position adjacente ?
Non
Oui
Fin
Si `pocheTrouvee` est `null` :
Oui
Non
Affecter cette poche à `pocheTrouvee` et y ajouter la position actuelle
Ajouter toutes les positions de la poche trouvée à `pocheTrouvee`
Ajouter cette poche à la liste `pochesAAjouter`
Sortir de la boucle
Si aucune poche trouvée :
Non
Oui
Fin
Créer une nouvelle poche contenant uniquement la position actuelle
Ajouter cette poche à la liste des poches du terrain



Conclusion personelles :

1. Athiran : Cette SAE m’a vraiment permis de mettre en pratique et d'approfondir mes connaissances en Java, tout en touchant à plusieurs aspects du développement logiciel. En plus de mieux comprendre la structure MVC, j’ai appris l'importance de penser à l'interface homme-machine, en concevant des interfaces intuitives et agréables avec `Swing`. Travailler sur l’interface m’a montré que chaque élément doit avoir une logique et être facile à utiliser pour l’utilisateur. En bref, cette SAE m’a donné une vraie expérience de projet : construire quelque chose de bout en bout, en connectant les points entre la logique, l’interface et la structure du code. (modifié)

2. Tarehi : Conclusion personnelle Cette SAE m’a fait avancer sur plein de sujets différents, de la logique de programmation à la conception d’interfaces, et même à la gestion de projet. C’était super de voir comment chaque élément – du code au visuel – doit être cohérent pour que tout fonctionne bien ensemble. Le modèle MVC m’a vraiment aidé à structurer mon code de façon claire, en séparant bien les responsabilités. Sur le côté interface, j’ai mieux compris l’importance de rendre l’application intuitive, pour que l’utilisateur puisse naviguer sans trop réfléchir. La gestion de la base de données a aussi été un vrai plus, avec la connexion à MariaDB pour garder les scores et les données. Ça m’a permis de voir comment lier différents composants dans un projet concret, un peu comme on le ferait dans un vrai environnement de travail. En somme, cette SAE m’a donné une vue d’ensemble du processus de développement, de la planification jusqu’au produit final, en m’apprenant à jongler avec les aspects techniques et ceux de l’expérience utilisateur.

3. Amir :

   Amir : Cette SAE a vraiment été une super expérience pour moi. Elle était bien pensée et m’a permis de mettre en pratique tout ce qu’on a vu en cours et en TP de façon concrète. En bossant sur l’architecture MVC, j’ai mieux compris l’intérêt de bien séparer les différentes parties d’un projet : les données, l’interface. Ça rend le code beaucoup plus clair et facile à maintenir, surtout sur des projets plus gros. J’ai aussi appris à manipuler les bases de données avec JDBC, ce qui m’a aidé à saisir comment sécuriser les requêtes et gérer les erreurs efficacement. Au final, cette SAE m’a permis de prendre confiance en moi et de comprendre comment structurer un projet du début à la fin. C’était concret, enrichissant, et ça m’a permis d’aller plus loin que le simple cadre des TP ! (modifié)

4.