BUT2 Informatique
Groupe 3

RAPPORT
Paysage

Clément JANNAIRE
Clémence DUCREUX
Aurélien AMARY

2024-2025

SOMMAIRE

INTRODUCTION

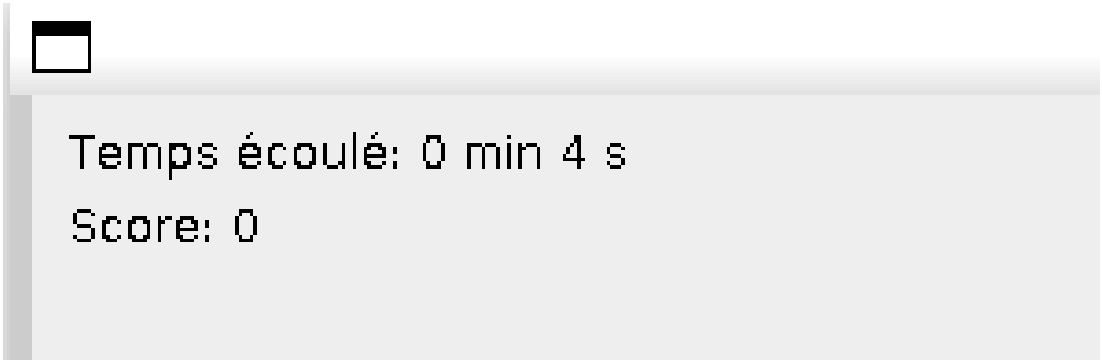
Le projet "Paysages" est un jeu de stratégie où le joueur construit un paysage en plaçant des tuiles hexagonales. Chaque tuile représente un ou plusieurs types de terrains (mer, champ, pré, forêt, montagne), et le but est d'assembler ces tuiles de manière à créer des zones cohérentes, appelées poches de terrains, pour obtenir le meilleur score possible. Le jeu s'inspire librement du concept de Dorfromantik.

Le développement a été réalisé en Java, en appliquant les principes de la programmation orientée objet et en suivant une structure claire du code. Chaque classe est documentée avec Javadoc et un Makefile est utilisé pour gérer la compilation et la génération d'un fichier jar exécutable. En fin de partie, les scores des joueurs sont enregistrés dans une base de données pour permettre leur comparaison.

FONCTIONNALITÉS

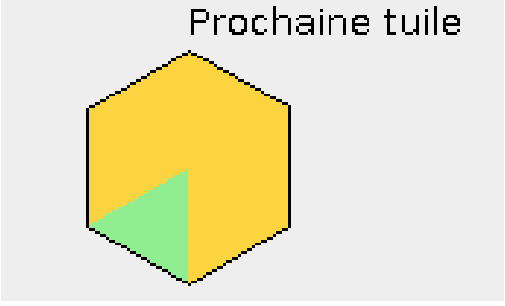
Timer et Score

Le jeu utilise un **Timer** pour suivre et afficher le temps écoulé depuis le début de la partie, en actualisant l'affichage chaque seconde. Lorsqu'une nouvelle partie démarre, le moment de début est enregistré. Ensuite, un **Timer Swing** déclenche la méthode **repaint()** toutes les secondes, permettant à l'interface de rafraîchir le temps affiché. Dans le code d'affichage, le temps écoulé est calculé en soustrayant l'heure actuelle à celle du début, et il est affiché en minutes et secondes. Si le joueur termine ou redémarre la partie, le timer se réinitialise, repartant de zéro.
La fonctionnalité de score enregistre les points obtenus par les joueurs, les trie, et les affiche dans un classement. Elle motive les joueurs, facilite la comparaison des performances, et permet de suivre les progrès dans le jeu.



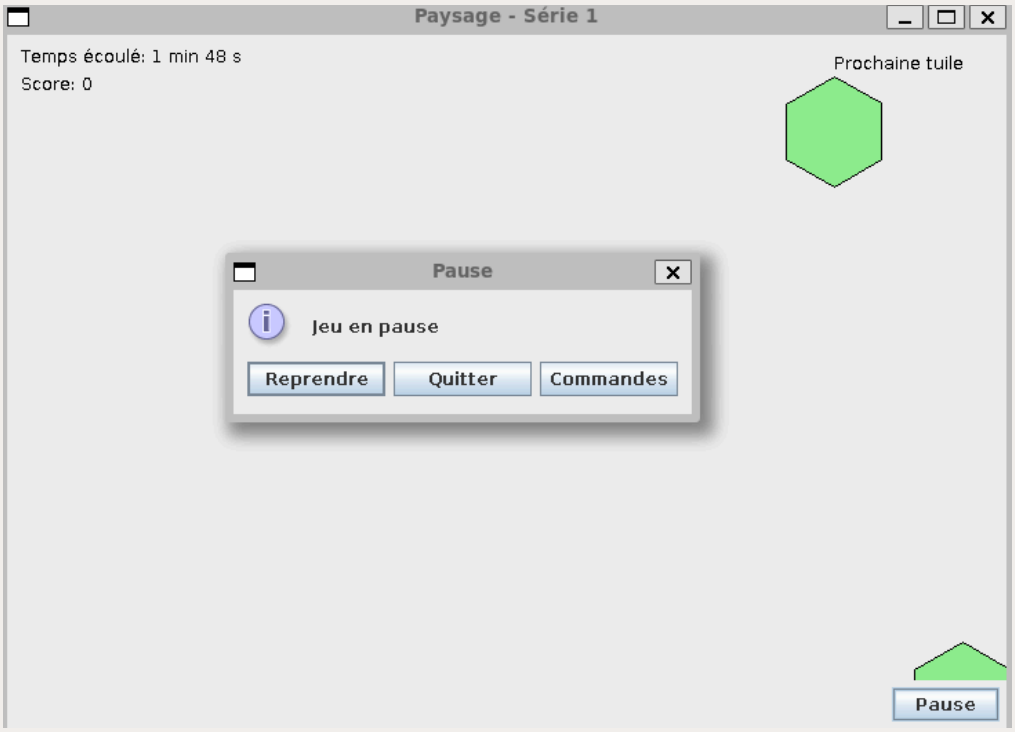
Voir la Prochaine Tuile

La fonctionnalité d'aperçu permet au joueur de voir la prochaine tuile avant de pouvoir la poser. Dans la classe Partie, la tuile suivante est gardée en réserve et affichée dans un coin de l'écran via **GamePanel**, où elle est dessinée à chaque tour. Cela permet au joueur d'anticiper ses actions en tenant compte de la tuile à venir, renforçant ainsi la dimension stratégique du jeu.



Pause :

La fonctionnalité "pause" permet de suspendre le déroulement de la partie en arrêtant le compteur de temps et en bloquant les actions du joueur. Dans la classe Partie, un attribut **enPause** vérifie si la partie est en pause, et les heures de début et de fin de pause sont utilisées pour ajuster le temps écoulé sans inclure la durée de la pause. Du côté de l'affichage, la classe **GamePanel** affiche un écran indiquant que le jeu est en pause et ignore les interactions jusqu'à ce que la pause soit désactivée, garantissant que le score et le temps de jeu ne sont pas impactés. Depuis l'écran de pause, le joueur a également la possibilité de quitter la partie ou de consulter les commandes, offrant ainsi plus de flexibilité et d'informations sans affecter le déroulement du jeu.

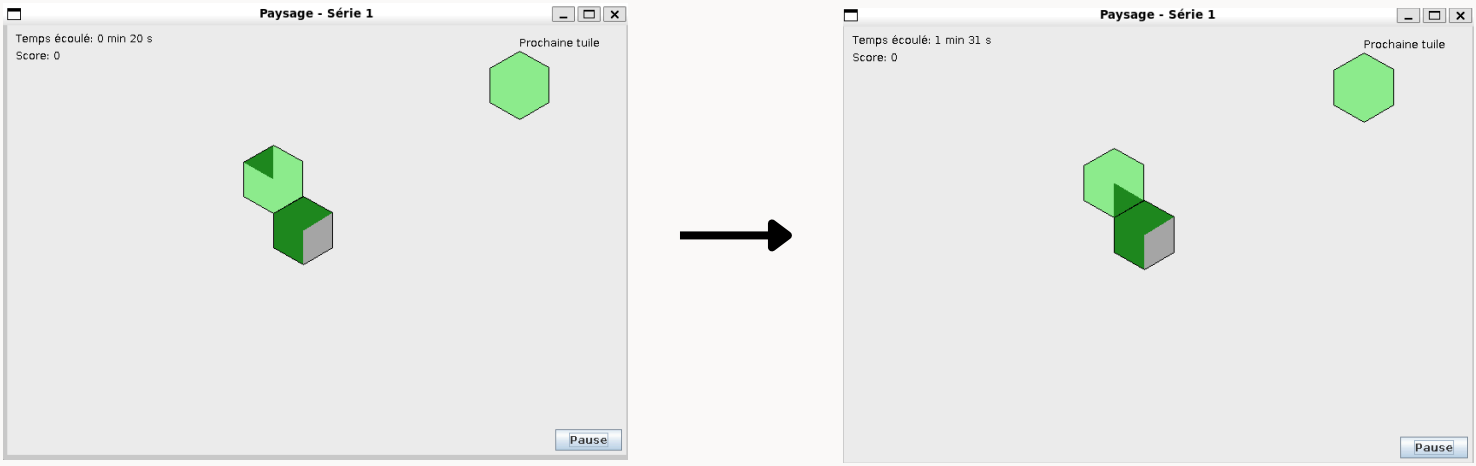


Poser des tuiles

Le placement d'une tuile fonctionne en vérifiant d'abord la validité de la position grâce à une méthode comme **estPlacementValide()**, qui contrôle si la tuile respecte les règles de voisinage et les types de terrains adjacents. Si le placement est correct, la méthode **ajouterTuile()** enregistre la tuile sur le terrain, en mettant à jour sa position, orientation et les liens possibles avec les tuiles voisines. Ensuite, **mettreAJourConnexions()** relie la nouvelle tuile aux autres tuiles environnantes, influençant ainsi les effets de terrain et la progression du jeu.

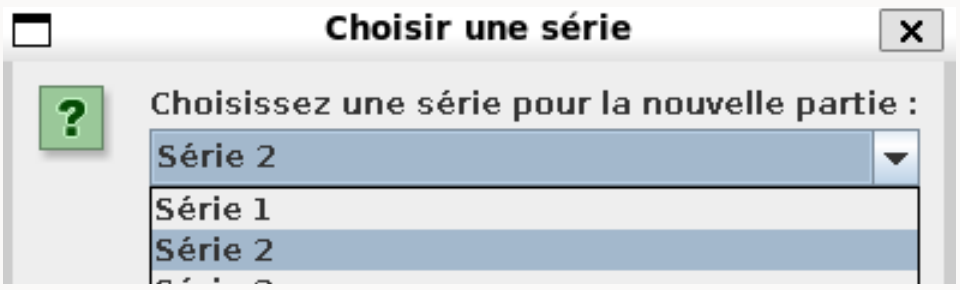
Tourner les tuiles

La rotation des tuiles fonctionne grâce à un écouteur de la molette de la souris dans la classe **GamePanel.** Lorsqu'un joueur tourne la molette, l'écouteur **MouseWheelListener** détecte ce mouvement et ajuste l'angle d'orientation de la tuile en cours de 60° : un tour vers le haut augmente l'angle, et un tour vers le bas le diminue. L'orientation de la tuile est alors mise à jour et limitée à une valeur entre 0° et 360° pour rester cohérente. Ensuite, la méthode **repaint()** est appelée, rafraîchissant l'affichage du jeu pour montrer la tuile avec sa nouvelle orientation.



Choisir des série

La fonctionnalité de choix de série fonctionne en proposant au joueur une liste de séries disponibles à l'écran d'accueil (**PageAccueil**). Lorsqu'une série est sélectionnée, elle est transmise à une nouvelle instance de Partie, qui appelle ensuite la classe Serveur pour charger les tuiles spécifiques de cette série depuis la base de données. Ces tuiles définissent le contenu et les défis de la partie, offrant une expérience de jeu personnalisée en fonction de la série choisie.

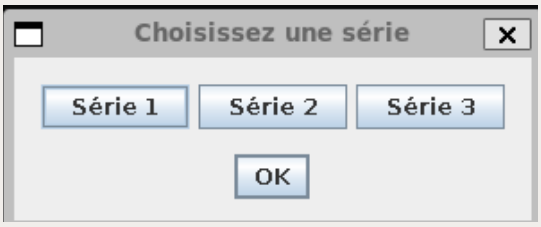


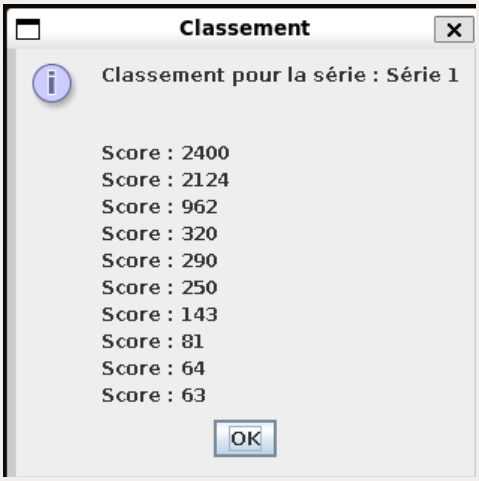
Se déplacer sur la map

La méthode **mouseDragged** dans **GamePanel**.java permet de déplacer la vue de la carte en glissant la souris. Elle calcule le décalage de position et ajuste les coordonnées de la caméra, puis appelle **repaint**() pour actualiser l'affichage en temps réel, rendant le déplacement visible immédiatement.

Classement

La fonctionnalité de classement permet d'afficher les meilleurs scores des joueurs. Une méthode comme **getClassement**() récupère les scores depuis la base de données, les trie par série et par ordre décroissant, puis les affiche. Ce classement permet aux joueurs de comparer leurs performances et de voir les meilleurs résultats en un coup d'œil.





Commande

Dans le fichier **PageAccueil.java**, la méthode **afficherCommandes()** gère l'affichage des commandes du jeu. Elle crée un texte décrivant les actions de la souris (comme le clic gauche pour poser une tuile ou le clic droit pour déplacer la vue) et la légende des couleurs des terrains. Ce texte est affiché dans une boîte de dialogue en utilisant **JOptionPane.showMessageDialog,** permettant au joueur de consulter facilement les commandes depuis le menu principal

BASE DE DONNÉE

La base de données est organisée autour de quatre tables clés :

- **Séries** : Elle contient les différentes séries de jeux, chaque série ayant un identifiant unique **NumSérie (clé primaire)** et un **Nom** descriptif. Elle sert de point central pour relier les séries aux tuiles et aux scores.

- **Tuiles** : Représente les tuiles appartenant aux séries de jeux. Chaque tuile est identifiée par **NumTuile (clé primaire)** et possède un **CodeTuile** (identifiant unique) ainsi qu'une Orientation par défaut. Elle est liée à une série spécifique par **NumSérie**, qui agit comme une clé étrangère. Si une série est supprimée, toutes ses tuiles associées le sont également, grâce à la contrainte **ON DELETE CASCADE**.

- **CompositionTuiles** : Cette table détaille les types de terrains présents sur chaque tuile. Elle inclut **TerrainType** (type de terrain, comme forêt ou montagne) et **NombreTriangles** (quantité de triangles pour ce terrain). Elle est liée aux tuiles via **NumTuile**, permettant de définir la composition spécifique de chaque tuile.

- **Score** : Enregistre les scores des joueurs pour chaque série. Elle inclut Score (le score obtenu), **TempsJoué** (durée de la partie), et **DateJeu** (date de la partie), ainsi que **NumSérie** pour relier le score à une série spécifique. Un index sur **NumSérie** améliore l'efficacité des recherches par série.

(Voir fichier sur GIT pour plus de détail.)

STRUCTURE PROGRAMME

Le fichier **Clé.java** stocke les informations de connexion à la base de données via des constantes, servant de référence pour les classes qui accèdent aux données de jeu.

- Cette classe ne contient pas de méthodes spécifiques, seulement des constantes pour la connexion à la base de données (**LIEN, ID, MP**).

GamePanel est la classe qui gère l'interface graphique du jeu, permettant aux joueurs de manipuler les tuiles et de voir le plateau. Elle utilise des écouteurs pour capter les interactions de la souris et un Timer pour actualiser l'affichage.

- **paintComponent(Graphics g):** Dessine le plateau, les tuiles posées, la tuile en cours et le temps écoulé.
- **mouseWheelMoved(MouseWheelEvent e):** Gère la rotation de la tuile en cours via la molette.
- **mouseClicked(MouseEvent e):** Valide et place la tuile si le joueur clique à un emplacement valide.
- **afficherFinDePartie():** Affiche un message de fin de partie et propose de rejouer ou de retourner à l’accueil.
- **enregistrerScoreBD(String serieNom, int score, int tempsJoue):** Enregistre le score et le temps total dans la base de données.

Main.java contient le point d'entrée du programme. Il initialise l'interface d'accueil et prépare la liste des parties précédentes.

- **main(String[] args):** Initialise l'application en créant PageAccueil.

PageAccueil est l'écran d'accueil où le joueur peut voir les parties précédentes et sélectionner une série pour commencer une nouvelle partie.

- **PageAccueil(List<Partie> partiesPrecedentes):** Initialise l'interface avec la liste des parties.
- **actionPerformed(ActionEvent e):** Gère les interactions utilisateur, comme le choix de série.

Partie.java Cette classe gère une session de jeu, stockant les tuiles posées, les tuiles restantes, le score et le temps écoulé.

- **getProchaineTuile():** Donne la prochaine tuile et la retire de la liste.
- **ajouterTuile(Tuile tuile):** Ajoute une tuile à la liste des tuiles posées.
- **peekProchaineTuile():** Donne un aperçu de la prochaine tuile sans la retirer.
- **getTuilesPosees():** Retourne la liste des tuiles posées.

Score.java gère le score de la partie en cours, avec des méthodes simples d'accès et de modification.

- **getValeur():** Récupère la valeur actuelle du score.
- **setValeur(int valeur):** Met à jour le score.

Serveur est la classe de gestion de la base de données, initialisée avec les informations de Clé.

- **Serveur():** Initialise la connexion à la base de données.
- **getTuilesAsObjects(String serie):** Charge les tuiles d'une série depuis la base de données.
- **ajouterScore(String serieNom, int score, int tempsJoue):** Enregistre le score et le temps de jeu.
- **fermerServeur():** Ferme la connexion.

Terrain.java Une énumération des différents types de terrain possibles pour les tuiles.

- Pas de méthodes spécifiques, seulement les types de terrain (OCEAN, CHAMP, PRE, FORET, MONTAGNE).

Tuile représente chaque pièce de jeu avec des informations de position, d'orientation, et de type de terrain.

- **dessiner(Graphics g, int x, int y):** Dessine la tuile sur le plateau.
- **axialToCartesian():** Convertit les coordonnées hexagonales en coordonnées cartésiennes.
- **setOrientation(int orientation):** Définit l'orientation pour permettre la rotation.
- **getY() et getR():** Fournissent les coordonnées de la tuile pour vérifier sa position.

Le fichier **makefile** fournit une structure pour compiler et exécuter le projet Java en organisant les dépendances et en définissant les étapes nécessaires. Voici une explication de chaque section importante :

Variables

- **JAVAC :** Définit le compilateur Java (javac).
- **JAVA** : Définit l'exécuteur Java (java).
- S**RC, BIN, LIB :** Indiquent respectivement les répertoires source (où se trouvent les fichiers .java), binaire (pour stocker les fichiers .class après compilation), et de bibliothèque (spécifique ici au client mariadb-java).
- Liste des fichiers à compiler :
- **SOURCES :** Utilise la fonction **wildcard** pour récupérer tous les fichiers .**java** dans le dossier **src**, générant automatiquement la liste des fichiers à compiler.
- **CLASSES** : Convertit chaque fichier .**java** de **SRC** en un fichier .**class** correspondant dans **BIN**.

Dépendances explicites

- Les dépendances entre les fichiers sont établies pour s'assurer que les classes nécessaires sont compilées dans le bon ordre. Par exemple :
- **PageAccueil.class** dépend de **Partie.class** et **Serveur.class.**
- **GamePanel.class** dépend de **Partie.class, Serveur.class** et **Tuile.class.**
- Cela garantit qu'une classe est compilée seulement si les classes dont elle dépend sont prêtes.

Règles de compilation et exécution

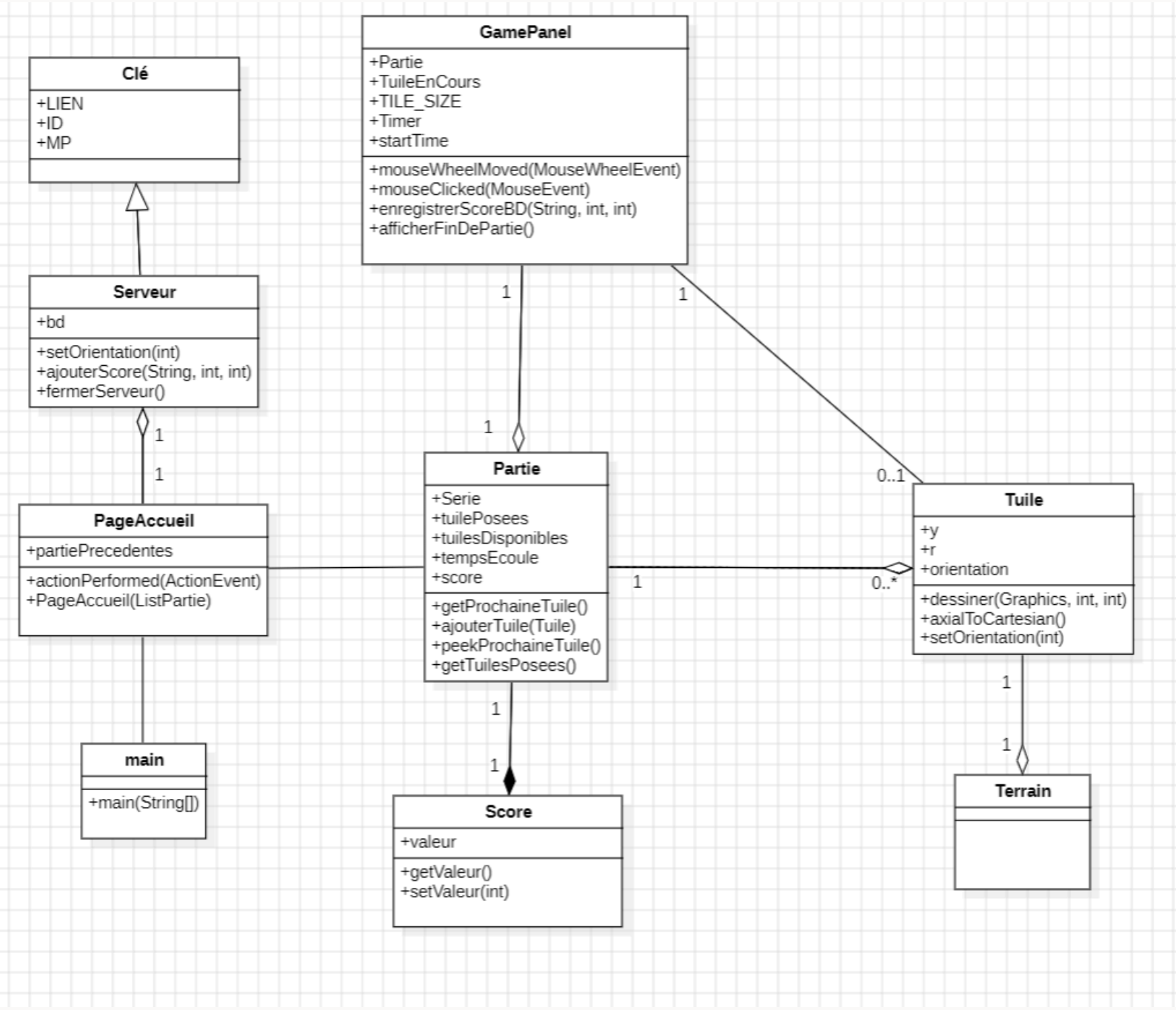
- **all :** Cible principale qui compile tout le projet en créant d'abord le répertoire **BIN** puis en générant tous les .**class**.

- **run** : Exécute le projet en lançant la classe principale (**Main**) avec les bibliothèques nécessaires, en utilisant **java -cp** pour spécifier le chemin de la bibliothèque **mariadb**.

- **clean** : Supprime tous les fichiers .**class** du dossier **BIN**, permettant de nettoyer les fichiers de compilation.

- **jar :** Crée une archive JAR contenant tous les fichiers .**class** compilés. Cette règle utilise jar pour créer un fichier JAR à partir du contenu du répertoire **BIN**. Cela vous permet de distribuer facilement votre application en une seule archive exécutable.

DIAGRAME DE CLASSE



ERGONOMIE

Écran d'Accueil (PageAccueil.java)

- **Interface simplifiée et intuitive** : Les éléments essentiels (boutons, liste des séries de tuiles, etc.) sont disposés de manière à guider le regard du joueur, avec des étiquettes claires pour chaque action.

- **Retour visuel sur les actions** : Les boutons changent de couleur ou affichent un effet visuel (légère surbrillance) au passage de la souris pour signaler l'interactivité.

- **Confirmation des actions importantes :** Lorsqu'un joueur sélectionne une série et démarre une partie, une confirmation s'affiche pour valider le choix. Cela permet de réduire les erreurs et garantit que le joueur commence bien avec la série souhaitée.

Écran de jeu (GamePanel)

- **Accessibilité des commandes** : Les commandes de jeu et les informations essentielles (score, temps restant, etc.) sont affichées de manière constante mais discrète autour de la zone de jeu pour que le joueur puisse y accéder facilement sans surcharger l’écran.

- **Fonctionnalité pause bien intégrée :** Un bouton de pause, facilement accessible, permet de suspendre la partie, et un écran semi-transparent informe l'utilisateur que le jeu est en pause tout en laissant une vue partielle de la situation de jeu. Pendant la pause, un rappel des touches de commande est affiché, permettant au joueur de les consulter au besoin avant de reprendre. Depuis cet écran de pause, le joueur peut également choisir de quitter la partie ou de consulter les commandes en détail, offrant ainsi plus de flexibilité sans affecter le déroulement du jeu.

Écran de score (Score)

- **Classement et accessibilité :** Le classement des scores est bien structuré avec les meilleurs scores en haut, et les détails comme le nom de la série et la date du score sont affichés pour contextualiser les performances.

- **Navigation facile :** Boutons pour revenir rapidement à l'écran de jeu ou à l'accueil, améliorant la fluidité de navigation pour les utilisateurs qui consultent régulièrement leurs performances.

Écran de sélection de série (Partie lié à Serveur) :

- **Filtrage et tri des séries :** Les séries disponibles peuvent être triées ou filtrées par popularité, date ou type, permettant aux utilisateurs de trouver facilement une série spécifique.

- **Indicateurs visuels :** Les séries avec des scores élevés ou fréquemment jouées sont marquées, donnant une indication rapide aux utilisateurs sur les tendances.

ALGORITHMES

Algorithme d'identification des poches

L'algorithme comporte trois méthodes principales

- **getPocheDeTuile(Tuile tuile)** : Détecte une poche de tuiles connectées à partir d'une tuile donnée.

- **explorePoche(Tuile tuile, List<Tuile> poche, Set<Tuile> visitees) :** Fonction récursive pour l'exploration en profondeur, ajoutant les tuiles connectées de même type à la poche.

- **sontTerrainsCompatibles(Tuile tuile1, Tuile tuile2) :** Vérifie si deux tuiles ont des terrains compatibles pour être dans la même poche.

Étapes de l'algorithme

- **Initialisation :** La méthode **getPocheDeTuile** commence en créant une liste poche pour stocker les tuiles connectées et un ensemble **visitees** pour enregistrer les tuiles déjà explorées.

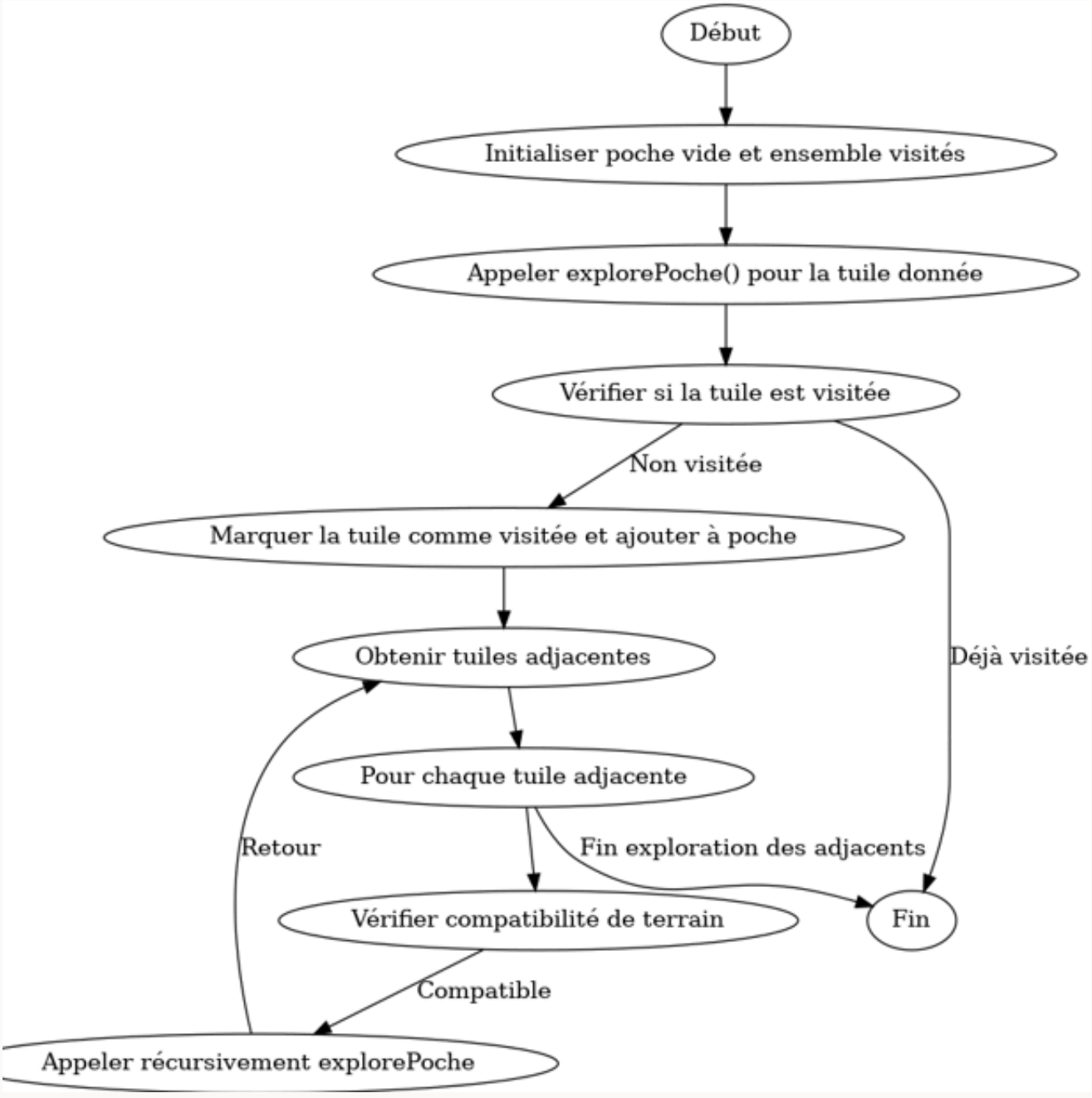
- **Exploration en profondeur : explorePoche** est appelée sur la tuile initiale pour explorer toutes les tuiles adjacentes. Elle ajoute chaque tuile compatible à la poche.

- Vérification des adjacents : La méthode **getTuilesAdjacentes** est utilisée pour obtenir les tuiles adjacentes à la tuile courante.

- **Compatibilité de terrain : sontTerrainsCompatibles** est appelée pour vérifier si la tuile explorée et l'adjacente partagent le même type de terrain.

- **Ajout à la poche :** Si deux tuiles sont compatibles, **explorePoche** est appelée récursivement sur l'adjacente.

- Finalisation : Lorsque toutes les tuiles connectées ont été explorées, **getPocheDeTuile** retourne la liste de tuiles formant une poche.



EXPOSITION SCORE

Comparaison entre les joueurs

- La méthode **getScores** dans la classe Serveur est utilisée pour récupérer tous les scores d'une série spécifique.
- Requête Préparée : Utilisation de requêtes préparées pour éviter les injections SQL. Cela renforce la sécurité lors de la récupération des données.
- Tri des Scores : Les scores sont récupérés et triés par ordre décroissant **(ORDER BY Score DESC)**, ce qui permet de déterminer facilement les meilleurs scores.
- Stockage dans une Liste : Les scores récupérés sont stockés dans une liste qui sera utilisée pour l'affichage et la comparaison.

Classement

- Affichage des Scores : Cette méthode affiche les scores dans une boîte de dialogue, permettant aux joueurs de voir leur position par rapport aux autres.
- Limite d'Affichage : Pour éviter de surcharger l'interface, seuls les meilleurs scores (par exemple, les 10 premiers) sont affichés.

Mise à jour dynamique du score

- Dans la méthode **ajouterScore** de **Serveur.java,** une fois un score ajouté à la base de données, on peut réévaluer le classement des scores pour maintenir un classement dynamique.

CONCLUSION

Clément:

Ce projet m'a procuré une expérience de développement à la fois stimulante et enrichissante, bien qu'il ait également comporté son lot de surprises. J'ai notamment remarqué qu'un de mes collègues n'a pas, à mon sens, apporté une contribution suffisante, ce qui a parfois freiné notre avancée et augmenté ma charge de travail. Malgré ces défis, j'ai réussi à progresser sur mes propres tâches, en veillant à respecter scrupuleusement les exigences techniques et à documenter mon travail de manière rigoureuse.

Clémence:

Ce projet m'a offert une expérience de développement intéressante et enrichissante, mais il n'a pas été exempt de surprise, notamment un de mes camarades n'a, selon moi, pas suffisamment contribué au projet, ce qui a parfois ralenti notre progression et alourdi ma charge de travail. Malgré ces difficultés, j'ai pu avancer sur mes propres responsabilités, en prenant soin de respecter au mieux les attentes techniques et de documenter mon travail avec le plus de rigueur possible.

Aurélien:

Tout d'abord, je tiens à remercier mes camarades de projet qui ont toujours été patients et bienveillant à mon égard. Grâce à leur aide, j'ai pu apprendre de mes erreurs et nous avons désormais la base de données la meilleure possible. De manière générale je ressors ragaillardi de ce projet et je pense que cette expérience m'aura permis de mieux saisir l'importance du bon travail en équipe. Alors comme gage de leur confiance, j'ai créé cet amusant petit bloc de pseudocode qui représente bien l'esprit général de développement tout au long du projet.

```
/*General Human Thinking*/
//Typical Thinks
while(isGood){  //The majority of Human
   continue;
}
//Avanced Thinks
while(beImprove){  //The minority of Human
   improve();
}
```