

# **Projet 2CCPP**

## **Laying Grass**

Documentation technique

## Table des matières

<b>Projet 2CCPP .....</b>	<b>1</b>
<b>1. Objectifs du projet .....</b>	<b>3</b>
<b>Architecture et conception .....</b>	<b>3</b>
<b>Gestion du plateau et des tuiles .....</b>	<b>3</b>
<b>Gestion des joueurs .....</b>	<b>3</b>
<b>Mécaniques spéciales .....</b>	<b>4</b>
<b>Gestion du déroulement du jeu .....</b>	<b>4</b>
<b>Interface et robustesse .....</b>	<b>4</b>
<b>2. Architecture générale .....</b>	<b>5</b>
<b>Arborescence du projet.....</b>	<b>5</b>
<b>Description des principaux modules.....</b>	<b>6</b>
<b>Objectif de cette organisation .....</b>	<b>7</b>
<b>3. Détails des classes .....</b>	<b>8</b>
<b>Classe Player .....</b>	<b>8</b>
<b>Classe Tile .....</b>	<b>10</b>
<b>Classe Board.....</b>	<b>12</b>
<b>Classe Cell .....</b>	<b>13</b>
<b>Classe RendererCLI .....</b>	<b>15</b>
<b>Classe Game .....</b>	<b>17</b>
<b>Classe BonusSquare.....</b>	<b>19</b>
<b>Classe TileFactory .....</b>	<b>20</b>
<b>Classe TileQueue.....</b>	<b>21</b>
<b>Classe InputManager .....</b>	<b>22</b>
<b>4. Système de bonus .....</b>	<b>23</b>
<b>5. Affichage CLI .....</b>	<b>24</b>
<b>6. Perspectives d'évolution.....</b>	<b>27</b>
<b>1. Interface .....</b>	<b>27</b>
<b>2. Mécaniques de jeu .....</b>	<b>27</b>
<b>3. Gestion de la partie .....</b>	<b>27</b>
<b>4. Multijoueur et réseau .....</b>	<b>27</b>

## 1. Objectifs du projet

Le projet *Laying Grass* a pour objectif principal de développer une application en C++ simulant un jeu de plateau en mode console (CLI).

Il vise à appliquer de manière concrète les concepts de programmation orientée objet et de gestion algorithmique dans un environnement multi-joueurs.

### Architecture et conception

- Concevoir une architecture modulaire et claire, reposant sur des classes distinctes :
  - Game : gestion globale du déroulement de la partie.
  - Board : représentation et manipulation du plateau de jeu.
  - Player : stockage des informations du joueur et gestion du territoire.
  - Tile : définition et manipulation des tuiles (formes, rotations, placement).
- Favoriser la réutilisabilité du code (principes SOLID, encapsulation).
- Assurer une séparation nette des responsabilités entre les classes.

### Gestion du plateau et des tuiles

- Implémenter un plateau dynamique (20×20 ou 30×30 selon le nombre de joueurs).
- Gérer le placement des tuiles :
  - vérification des collisions,
  - contrôle des limites,
  - validation du contact avec le territoire du joueur.
- Permettre la rotation et le retournement des tuiles avant placement.
- Implémenter un système de file (queue) pour les tuiles disponibles.

### Gestion des joueurs

- Permettre la création interactive des joueurs (nom + couleur/symbole).
- Attribuer automatiquement une tuile de départ 1×1 à chaque joueur.

- Suivre l'évolution du territoire individuel.
- Implémenter la file d'ordre de jeu aléatoire.
- Gérer les coupons d'échange de tuiles et leur utilisation.

## Mécaniques spéciales

- Intégrer les cases bonus :
  - *Échange* → ajout d'un coupon.
  - *Pierre* → blocage d'une case.
  - *Vol* → vol d'une tuile d'un autre joueur.
- S'assurer que ces bonus sont générés aléatoirement, mais sans se chevaucher ni apparaître sur les bords.

## Gestion du déroulement du jeu

- Implémenter un système de tours avec 9 manches par joueur.
- Gérer les cas où un joueur ne peut pas jouer (tuile défaussée).
- Implémenter les règles de fin de partie :
  - calcul du plus grand carré de territoire,
  - gestion des égalités (nombre total de cases).

## Interface et robustesse

- Créer une interface console lisible (affichage du plateau, tours, messages).
- Vérifier toutes les entrées utilisateurs pour éviter les erreurs.
- Garantir la stabilité et la résilience du programme (aucun crash, validations systématiques).
- Prévoir des messages d'erreur et de confirmation clairs.

## 2. Architecture générale

L'application est structurée selon une architecture modulaire, séparant clairement les déclarations (fichiers .hpp) et les implémentations (fichiers .cpp).

Cette organisation vise à assurer la lisibilité, la maintenabilité et la réutilisabilité du code.

### Arborescence du projet

2CCPP/

```
|
|
| └─ includes/      # Fichiers d'en-tête (déclarations des classes et structures)
|   |
|   | └─ board.hpp
|   |
|   | └─ bonusSquare.hpp
|   |
|   | └─ cell.hpp
|   |
|   | └─ enums.hpp
|   |
|   | └─ game.hpp
|   |
|   | └─ inputManager.hpp
|   |
|   | └─ main.cpp
|   |
|   | └─ player.hpp
|   |
|   | └─ rendererCLI.hpp
|   |
|   | └─ tile.hpp
|   |
|   | └─ tileFactory.hpp
|   |
|   └─ tileQueue.hpp
|
|
| └─ src/           # Fichiers sources (implémentations)
|   |
|   | └─ board.cpp
|   |
|   | └─ bonusSquare.cpp
|   |
|   | └─ cell.cpp
|   |
|   └─ game.cpp
```

```

|   ├── inputManager.cpp
|   ├── player.cpp
|   ├── renderer.cpp
|   ├── tile.cpp
|   ├── tileFactory.cpp
|   └── tileQueue.cpp
|
└── Makefile      # Script de compilation du projet

```

## Description des principaux modules

Fichier / Classe	Rôle principal
board.hpp / board.cpp	Gère le plateau de jeu, la disposition des cellules et le placement des tuiles.
cell.hpp / cell.cpp	Représente une cellule du plateau (état : vide, herbe, pierre, bonus, etc.).
player.hpp / player.cpp	Contient les informations des joueurs (nom, couleur, territoire, coupons, etc.).
tile.hpp / tile.cpp	Définit la structure et les transformations possibles d'une tuile (rotation, inversion).
tileFactory.hpp / tileFactory.cpp	Génère les tuiles de formes aléatoires à partir d'un ensemble prédéfini.
tileQueue.hpp / tileQueue.cpp	Gère la file des tuiles à distribuer aux joueurs (queue de tirage).
bonusSquare.hpp / bonusSquare.cpp	Gère les bonus spéciaux (échange, pierre, vol) et leur activation.
game.hpp / game.cpp	Contrôle la logique du jeu : tours, placement, règles, victoire.
inputManager.hpp / inputManager.cpp	Gère les entrées utilisateur (choix, coordonnées, rotations).

renderCLI.hpp / render.cpp	Assure l’affichage du plateau et des informations en mode console.
enums.hpp	Contient les énumérations globales (type de tuile, état de cellule, type de bonus).
main.cpp	Point d’entrée du programme : initialise et lance la partie.
Makefile	Automatise la compilation et le lien du projet.

### Objectif de cette organisation

- Séparer clairement la logique métier (jeu, plateau, joueur) de l’affichage et de l’interaction utilisateur.
- Faciliter l’ajout de nouvelles fonctionnalités (bonus, tailles de tuiles, etc.) sans modifier le cœur du code.
- Respecter les bonnes pratiques de conception orientée objet et modulaire.

### 3. Détails des classes

#### Classe Player

##### Rôle général

Représente un joueur du jeu *Laying Grass*.

Gère ses informations (nom, couleur, identifiant), son territoire de tuiles, sa base de départ et ses coupons d'échange.

Interagit avec le plateau (Board) pour le placement et le calcul du score.

##### Principaux attributs

Attribut	Type	Description
name	std::string	Nom du joueur
color	std::string	Couleur ou symbole
id	int	Identifiant unique
coupons	int	Nombre de coupons d'échange
territory	std::vector<Tile>	Liste des tuiles placées
baseRow, baseCol	int	Coordonnées de la tuile de départ
placedAnyTile	bool	Indique si une tuile a été placée

##### Méthodes principales

Méthode	Description
Player(const std::string&, const std::string&, int)	Constructeur du joueur
void placeTile(Tile&)	Ajoute une tuile au territoire
Tile removeLastTile()	Retire la dernière tuile (bonus de vol)
void useCoupon() / addCoupon()	Gère les coupons d'échange
void setBase(int, int) / getBase()	Définit ou récupère la base de départ
int calculateLargestSquare(const Board&) const	Calcule la taille du plus grand carré d'herbe
int countGrassCells(const Board&) const	Compte les cases d'herbe contrôlées

getName(), getColor(), getId(), getCoupons()	Accesseurs basiques
----------------------------------------------	---------------------

### **Responsabilités**

- Gérer les données et actions d'un joueur.
- Maintenir la cohérence du territoire et des coupons.
- Calculer les scores individuels pour la fin de partie.

## Classe Tile

### Rôle général

Représente une tuile du jeu *Laying Grass*.

Gère sa forme (matrice 0/1), permet les transformations géométriques (rotation, flips), et vérifie si elle peut être placée sur le plateau (Board) par un joueur (Player).

### Principaux attributs

Attribut	Type	Description
shape	std::vector<std::vector<int>>	Matrice représentant la forme de la tuile (1 = case remplie, 0 = vide)
width	int	Largeur actuelle de la tuile
height	int	Hauteur actuelle de la tuile

### Méthodes principales

Méthode	Description
Tile(const std::vector<std::vector<int>>& s)	Constructeur qui initialise la forme et les dimensions de la tuile
void rotate()	Effectue une rotation de 90° dans le sens horaire
void flipH()	Retourne la tuile horizontalement (miroir gauche-droite)
void flipV()	Retourne la tuile verticalement (miroir haut-bas)
bool canPlace(const Board&, int row, int col, const Player&) const	Vérifie si la tuile peut être placée aux coordonnées données selon les règles du jeu
void print() const	Affiche la tuile dans la console sous forme de grille
void printSmall() const	Affiche uniquement la taille de la tuile (rows x cols)

<code>void printInline() const</code>	Affiche la tuile en ligne, utile pour un aperçu compact
---------------------------------------	---------------------------------------------------------

### Responsabilités

- Gérer les transformations géométriques d'une tuile.
- Vérifier la validité d'un placement sur le plateau selon les règles du jeu.
- Fournir des méthodes d'affichage pour le debug et la visualisation.
- Conserver la cohérence des dimensions après rotation ou flip.

## Classe Board

### Rôle général

Représente le plateau de jeu pour *Laying Grass*.

Gère la grille de cellules (Cell), le placement des tuiles (Tile), l'ajout de bonus, et fournit des méthodes pour l'affichage et l'accès aux cases.

### Principaux attributs

Attribut	Type	Description
size	int	Taille du plateau (plateau carré size x size)
grid	std::vector<std::vector<Cell>>	Grille contenant les cellules du plateau

### Méthodes principales

Méthode	Description
Board(int size)	Constructeur qui initialise la grille avec des cellules vides
bool placeTile(Tile&, Player&, int row, int col)	Place une tuile sur le plateau pour un joueur si le placement est valide, applique les bonus éventuels, et retourne vrai si réussi
void addBonus(const BonusSquare&)	Ajoute un bonus à une case spécifique du plateau
void render() const	Affiche le plateau dans la console avec un symbole pour chaque type de case
int getSize() const	Retourne la taille du plateau
const Cell& at(int row, int col) const / Cell& at(int row, int col)	Accesseurs pour obtenir une cellule en lecture ou en écriture

### Responsabilités

- Maintenir l'état du plateau de jeu et sa grille de cellules.
- Vérifier et appliquer les placements de tuiles selon les règles du jeu.
- Gérer les bonus (échange, pierre, vol) et les informer au joueur concerné.
- Fournir un affichage clair du plateau pour debug ou interface console.
- Assurer l'accès sécurisé aux cellules pour d'autres classes (Tile, Player, Game).

## Classe Cell

### Rôle général

Représente une case du plateau de jeu.

Gère le type de terrain (vide, herbe, pierre), le joueur qui la contrôle et les éventuels bonus présents.

### Principaux attributs

Attribut	Type	Description
terrain	Terrain	Type de terrain (Empty, Grass, Stone, Bonus)
playerId	int	Identifiant du joueur contrôlant la case (-1 si aucun)
bonus	BonusType	Type de bonus sur la case (NONE, EXCHANGE, STONE, ROBBERY)
symbol	char	Caractère utilisé pour l’affichage dans la console

### Méthodes principales

Méthode	Description
Cell()	Constructeur qui initialise une case vide sans joueur ni bonus
bool isEmpty() const	Retourne vrai si la case est vide et sans joueur
bool isGrass() const	Retourne vrai si la case contient de l’herbe
void setGrass(int pid)	Définit la case comme herbe contrôlée par le joueur pid et met à jour le symbole
void setStone()	Définit la case comme pierre et met à jour le symbole
int getPlayerId() const	Retourne l’identifiant du joueur contrôlant la case
Terrain getTerrain() const	Retourne le type de terrain de la case
BonusType getBonus() const	Retourne le type de bonus présent sur la case
char getSymbol() const	Retourne le symbole actuel de la case
void setSymbol(char s)	Définit le symbole de la case (utile pour affichage personnalisé)

## **Responsabilités**

- Maintenir l'état d'une case (terrain, joueur, bonus).
- Fournir des informations aux autres classes (Board, Tile, Player) pour la validation et l'affichage.
- Gérer les changements de terrain et l'affichage associé.

## Classe `RendererCLI`

### Rôle général

Gère l’affichage du plateau, des tuiles et des files de tuiles dans la console (CLI).  
Permet de visualiser les états du plateau et des prévisualisations de placement de tuiles pour un joueur.

### Principaux attributs

Attribut	Type	Description
<i>(aucun attribut membre public dans le code fourni)</i>		La classe utilise uniquement des fonctions statiques et locales pour gérer les couleurs et l’affichage

### Méthodes principales

Méthode	Description
<code>void displayBoard(const Board&amp;, const std::vector&lt;Player&gt;&amp;) const</code>	Affiche le plateau avec les joueurs, les pierres et les bonus dans la console. Les cases herbeuses sont colorées selon le joueur.
<code>void displayBoardWithPreview(const Board&amp;, const std::vector&lt;Player&gt;&amp;, const Tile&amp;, int row, int col, const Player&amp;) const</code>	Affiche le plateau avec un aperçu d’une tuile à placer pour le joueur courant (? coloré).
<code>void displayTile(const Tile&amp;) const</code>	Affiche une tuile sous forme de grille dans la console.
<code>void displayQueue(const TileQueue&amp;) const</code>	Affiche toutes les tuiles présentes dans la file d’attente (queue) du jeu.

### Fonctions utilitaires internes

Fonction	Description
<code>ansiColor(const std::string&amp;)</code>	Retourne le code ANSI correspondant à une couleur donnée (pour l’affichage coloré).

<code>colorByPlayerId(const std::vector&lt;Player&gt;&amp;, int)</code>	Retourne le code couleur ANSI correspondant au joueur selon son identifiant.
-----------------------------------------------------------------------------	---------------------------------------------------------------------------------

## Responsabilités

- Afficher l'état actuel du plateau avec la coloration des joueurs.
- Fournir des prévisualisations de placement de tuiles pour aider le joueur.
- Afficher les tuiles individuellement ou en file d'attente de manière lisible.
- Centraliser la logique d'affichage dans la console pour éviter de polluer la logique du jeu.

## Classe Game

### Rôle général

Gère la logique globale du jeu *Laying Grass*.

Coordonne les joueurs, le plateau (Board), la file de tuiles, les bonus et le déroulement des tours et des rounds.

Permet la saisie des joueurs, le placement initial, les tours de jeu, les vols de tuiles et la fin de partie.

### Principaux attributs

Attribut	Type	Description
board	std::unique_ptr<Board>	Plateau de jeu principal
players	std::vector<Player>	Liste des joueurs participants
tileQueue	TileQueue	File de tuiles disponibles pour le jeu
currentRound	int	Numéro du round actuel (0 = round initial)

### Méthodes principales

Méthode	Description
Game(int nbPlayers)	Constructeur, initialise le plateau, les joueurs, les bonus et la file de tuiles
void playRound()	Exécute un round complet : placement initial (si premier round) et tours de chaque joueur
void playTurn(Player&)	Gère le tour d'un joueur : choix et manipulation d'une tuile, prévisualisation, placement et utilisation de coupons
void placeStartingTile(Player&)	Place la tuile de départ 1x1 d'un joueur et définit sa base
void robTile(Player&)	Permet à un joueur de voler une tuile à un autre joueur si possible
void placeStone()	Permet de placer une pierre sur une case vide du plateau
bool hasTiles() const	Indique si la file de tuiles n'est pas vide

void distributeBonuses(int nbPlayers)	Distribue des bonus sur le plateau en fonction du nombre de joueurs
void initializeTiles(int nbPlayers)	Initialise la file de tuiles en fonction du nombre de joueurs
void endGame()	Affiche le plateau final et le vainqueur
Player getWinner() const	Retourne le joueur gagnant selon la taille du plus grand carré et le nombre de cases d'herbe

## Responsabilités

- Initialiser et configurer le plateau et les joueurs au début de la partie.
- Gérer le déroulement des rounds et des tours des joueurs.
- Appliquer les règles du jeu (placement de tuiles, bonus, vol, pierres, coupons).
- Maintenir la cohérence de la file de tuiles et des scores.
- Déterminer le vainqueur à la fin de la partie et afficher le plateau final.

## Classe BonusSquare

### Rôle général

Représente une case bonus sur le plateau de jeu *Laying Grass*.

Gère le type de bonus et son application lorsqu'un joueur atteint ou active la case.

### Principaux attributs

Attribut	Type	Description
row	int	Ligne de la case sur le plateau
col	int	Colonne de la case sur le plateau
type	BonusType	Type de bonus (NONE, EXCHANGE, STONE, ROBBERY)

### Méthodes principales

Méthode	Description
BonusSquare(int row, int col, BonusType type)	Constructeur qui initialise la position et le type du bonus
void apply(Player&, Game&)	Applique le bonus au joueur et/ou au jeu selon le type
int getRow() const	Retourne la ligne de la case
int getCol() const	Retourne la colonne de la case
BonusType getType() const	Retourne le type de bonus

### Responsabilités

- Stocker la position et le type d'un bonus sur le plateau.
- Appliquer les effets du bonus lorsqu'un joueur interagit avec la case.
- Fournir des informations sur la position et le type du bonus aux autres classes (Board, Game, Player).

## Classe TileFactory

### Rôle général

Fournit l'ensemble des tuiles disponibles dans le jeu *Laying Grass*.

Permet de centraliser la création et la récupération des tuiles standard pour initialiser les files de tuiles des parties.

### Principaux attributs

Attribut	Type	Description
allTiles	static std::vector<Tile>	Contient toutes les tuiles du jeu sous forme de matrices 2D représentant leur forme

### Méthodes principales

Méthode	Description
static const std::vector<Tile>& getAllTiles()	Retourne la liste complète des tuiles disponibles pour le jeu

### Responsabilités

- Centraliser la définition de toutes les tuiles du jeu.
- Fournir un accès unique et constant à ces tuiles pour l'initialisation de la file de tuiles (TileQueue) dans les parties.
- Éviter la duplication des définitions de tuiles dans le code.

## Classe TileQueue

### Rôle général

Représente la file de tuiles disponibles pour les joueurs dans le jeu *Laying Grass*. Gère l'ordre des tuiles, leur distribution et permet les échanges ou la prévisualisation des prochaines tuiles.

### Principaux attributs

Attribut	Type	Description
queue	std::vector<Tile>	Conteneur stockant les tuiles en attente dans l'ordre de jeu

### Méthodes principales

Méthode	Description
bool empty() const	Vérifie si la file est vide
size_t size() const	Retourne le nombre de tuiles restantes dans la file
std::vector<Tile> peekNext(int count) const	Retourne les count prochaines tuiles sans les retirer
void pushBack(const Tile&)	Ajoute une tuile à la fin de la file
Tile getNextTile()	Retire et retourne la première tuile de la file
Tile takeAt(int index)	Retire et retourne la tuile à l'index spécifié
void assign(std::vector<Tile>::iterator begin, std::vector<Tile>::iterator end)	Initialise la file avec un sous-ensemble de tuiles
void clear()	Vide la file de toutes ses tuiles

### Responsabilités

- Maintenir l'ordre et l'intégrité des tuiles disponibles pour les joueurs.
- Fournir des méthodes pour retirer, prévisualiser ou échanger des tuiles.
- Faciliter l'initialisation et la réinitialisation de la file de tuiles pour une partie.

## Classe InputManager

### Rôle général

Gère les interactions avec l'utilisateur dans le jeu *Laying Grass*.

Permet de récupérer des commandes, des coordonnées ou des choix de transformation pour les tuiles.

### Méthodes principales

Méthode	Description
std::string getCommand() const	Demande à l'utilisateur d'entrer une commande et retourne la chaîne saisie
std::pair<int,int> getCoordinates()	Récupère une paire ligne/colonne depuis l'utilisateur, accepte lettres (A–Z) ou chiffres
char getRotationChoice() const	Demande à l'utilisateur de choisir une rotation (R) ou un flip (F)

### Responsabilités

- Centraliser la saisie utilisateur et la validation basique.
- Convertir des entrées sous forme de lettres ou chiffres en indices utilisables par le plateau.
- Fournir un point unique d'interaction pour les commandes et transformations des tuiles.

## 4. Système de bonus

### Types

- Échange de tuile (\*) : +1 coupon.
- Pierre (S) : placer une pierre immédiatement sur une case vide.
- Vol (R) : voler une tuile d'un autre joueur.

### Distribution

- Tile Exchange :  $\text{ceil}(1.5 \times \text{nbJoueurs})$
- Stone :  $\text{ceil}(0.5 \times \text{nbJoueurs})$
- Robbery :  $1 \times \text{nbJoueurs}$

### Règles

- Bonus déclenchés par adjacence (haut, bas, gauche, droite).
- Si recouvert directement → bonus perdu.
- Pierre et vol doivent être utilisés immédiatement.

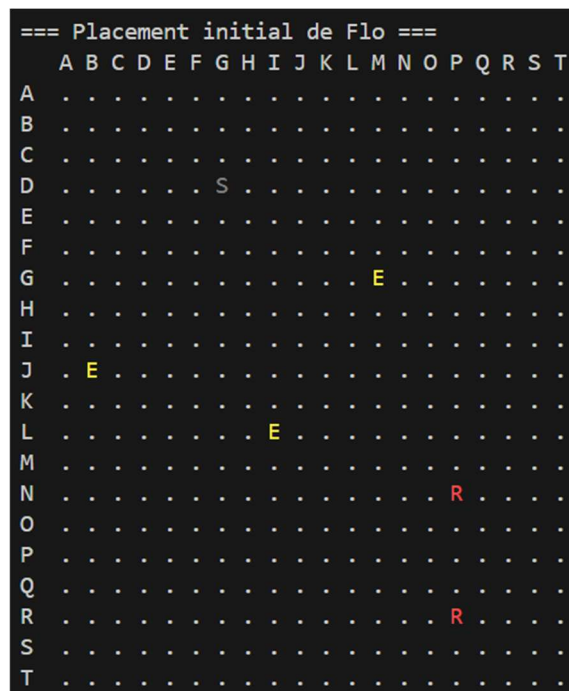
## 5. Affichage CLI

Le jeu utilise le terminal comme interface graphique, en représentant visuellement le plateau, les tuiles et les bonus avec des caractères ASCII et des couleurs ANSI.

L'affichage est géré principalement par la classe `RenderCLI`.

### 1. Plateau de jeu (Board)

- **Grille** : le plateau est une matrice `size × size` de `Cell`.
- **Colonnes et lignes** :
  - Les colonnes sont étiquetées par des lettres (A, B, C ...).
  - Les lignes sont également étiquetées par des lettres (A, B, C ...).



- **Cases** : le contenu de chaque `Cell` est affiché avec un symbole et éventuellement une couleur :

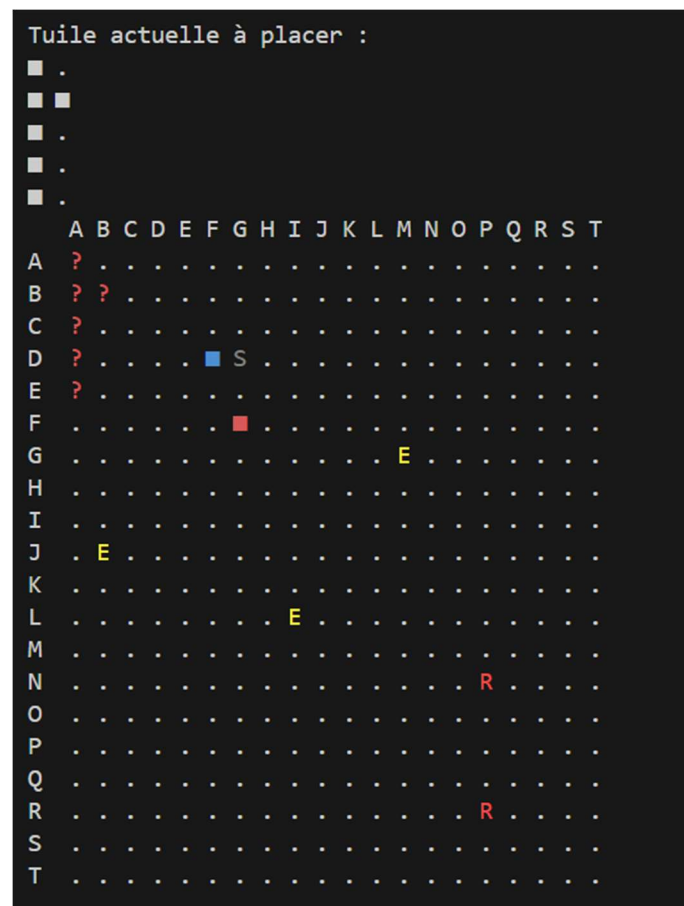
Terrain / Bonus	Symbole	Couleur / Notes
Vide (Empty)	.	Couleur par défaut
Herbe (Grass)	■	Couleur du joueur (via ANSI, basée sur <code>Player.color</code> )
Pierre (Stone)	#	Gris
Bonus (Bonus)	E, S, R	Jaune pour échange (E), gris pour pierre (S), rouge clair pour vol (R)

## 2. Prévisualisation d'une tuile

Lorsqu'un joueur déplace une tuile pour la placer, le plateau peut afficher une prévisualisation :

Les cellules de la tuile qui seraient posées sont affichées avec un ? coloré selon la couleur du joueur courant.

Cela permet au joueur de voir où la tuile tomberait avant de valider le placement.



## 3. Affichage des tuiles seules

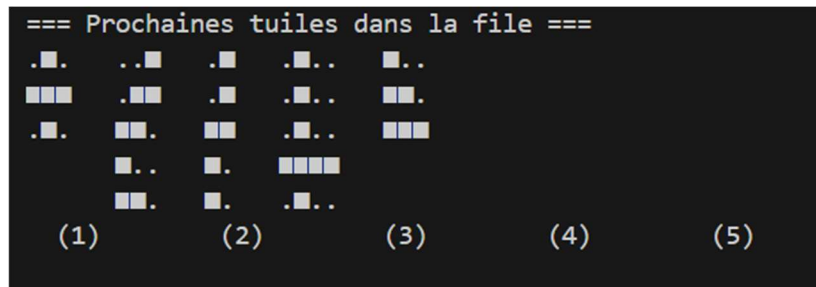
Pour afficher une tuile (dans la file ou lors de la sélection) :

Chaque tuile est représentée par une petite matrice de caractères ASCII (■ pour les cases actives, . pour les cases vides).

La méthode displayTile ou l'affichage inline (displayQueue) permet de montrer les prochaines tuiles à jouer.

## 4. File de tuiles (TileQueue)

- Les tuiles suivantes dans la queue peuvent être affichées **côte à côte**.
- Les indices (1, 2, 3...) sont indiqués sous chaque tuile pour faciliter l'échange (E i).



## 5. Utilisation des couleurs ANSI

- Chaque joueur a une couleur spécifique : rouge, bleu, vert, jaune, etc.
- Les tuiles posées par un joueur sont affichées en couleur pour une distinction visuelle rapide.
- Les bonus ont des couleurs vives pour attirer l'attention.
  - Échange : jaune vif (E)
  - Pierre : gris (S)
  - Vol : rouge clair (R)
- Les codes ANSI sont réinitialisés après chaque symbole (RESET) pour éviter que toute la ligne change de couleur.

## 6. Commandes et interaction

- Pendant le tour d'un joueur, le plateau est réaffiché à chaque modification.
- Les commandes comme M row col, R, FH, FV, V, E i sont accompagnées d'un **aperçu visuel immédiat**.
- Cela crée un **retour instantané** au joueur pour mieux décider de son placement.

## 6. Perspectives d'évolution

### 1. Interface

- **GUI** : passer du terminal à une interface graphique (SDL, Qt, Unity) avec glisser-déposer et animations.
- **CLI amélioré** : couleurs plus visibles, utilisation des flèches pour déplacer les tuiles.

### 2. Mécaniques de jeu

- **Nouvelles tuiles** et bonus : tuiles complexes, bonus temporaires ou interactifs.
- **Modes de jeu** : solo contre IA, multijoueur en réseau, défis avec objectifs spécifiques.
- **IA stratégique** : placer les tuiles pour maximiser le score ou bloquer l'adversaire.

### 3. Gestion de la partie

- **Sauvegarde et reprise** via fichiers JSON ou binaire.
- **Statistiques** : historique des coups, scores et classements.
- **Paramètres modulables** : taille du plateau, nombre de joueurs, couleurs.

### 4. Multijoueur et réseau

- Partie LAN ou en ligne, avec tour par tour et chat intégré.