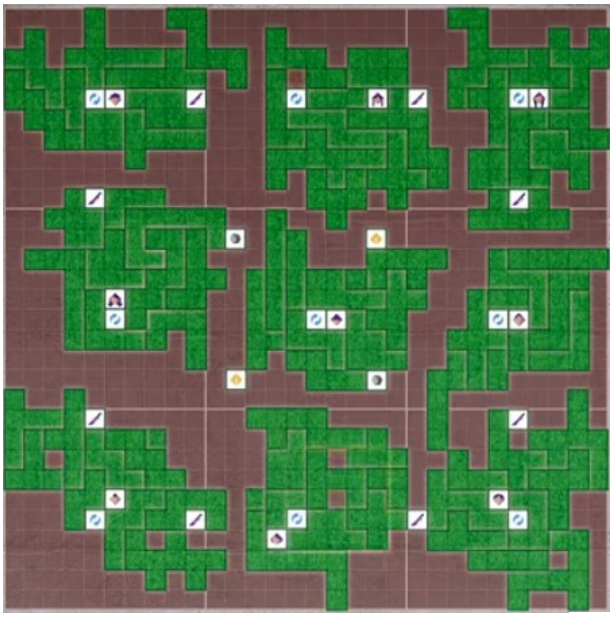
# 📘 Documentation Technique – Projet *Laying Grass*



**Auteurs :** Arthur Omar & Tom Berteloot
**École :** SUPINFO – 2e année

# 1. Introduction

Le projet *Laying Grass* est une implémentation en C++ d'un jeu de placement de tuiles inspiré de *Blokus* et du sujet d'évaluation CCPP.
Le but est de poser des tuiles sur un plateau en respectant certaines règles (pas d'empiètement, contact par les coins, bonus, etc.) pour former le plus grand carré possible.

Le projet respecte les contraintes minimales du sujet :

- Utilisation de **plusieurs classes** (Jeu, Plateau, Joueur, Tuile) ;
- Un **plateau dynamique** selon le nombre de joueurs ;
- **Tuiles générées et mélangées aléatoirement** ;
- **Conditions de victoire** basées sur la taille du plus grand carré et le nombre total de cases ;
- **Bonus actifs** (Échange, Pierre, Vol) ;
- **Rotation et symétrie** des tuiles implémentées.

# 2. Architecture du projet

## 2.1 Arborescence

```
2CCPP_Projet/
├── include/
│   ├── Common.hpp
│   ├── Jeu.hpp
│   ├── Joueur.hpp
│   ├── Plateau.hpp
│   ├── Tuile.hpp
│   └── tuilesDef.hpp
├── src/
│   ├── Jeu.cpp
│   ├── Joueur.cpp
│   ├── Plateau.cpp
│   ├── Tuile.cpp
│   ├── tuilesDef.cpp
│   ├── main.cpp        (démo automatique)
│   ├── main_jeu.cpp    (mode manuel)
│   └── test.cpp        (tests unitaires)
├── obj/
│   └── *.o
├── makefile
└── README.md
```

# 3. Description des classes principales

## 3.1 `class Tuile`

Représente une tuile composée de coordonnées relatives.

**Attributs :**

- `std::vector<Coordonnee> forme` : ensemble des blocs formant la tuile.

**Méthodes :**

- `rotate()` : rotation de 90°.
- `flip()` : symétrie horizontale.
- `normaliser()` : réaligne la tuile pour éviter les coordonnées négatives.
- `afficherApercu()` : affichage console en ASCII.

**Objectif :** Permet la manipulation et l'affichage des différentes formes du jeu.

## 3.2 `class Joueur`

Représente un joueur avec ses informations et ses bonus.

**Attributs :**

- `nom`, `couleur`, `id`
- `couponsEchange` : nombre de coupons disponibles.
- `tuileDepart` : coordonnée de la première tuile placée.

**Méthodes :**

- `ajouterCoupon()`, `utiliserCoupon()`
- `getNom()`, `getId()`, `getCoupons()`

**Objectif :** Gérer l'état individuel de chaque joueur.

### 3.3 `class Plateau`

Représente le plateau de jeu et la logique de placement.

**Attributs :**

- `std::vector<std::vector<char>> grille`
- `std::vector<std::vector<char>> bonus`
- `int tailleGrille` : dépend du nombre de joueurs.

**Méthodes :**

- `afficherGrille()` : rend le plateau en couleur (ANSI).
- `placerTuile()` : vérifie les collisions et place une tuile.
- `appliquerBonusesAprèsPlacement()` : gère les bonus (E, P, V).
- `plusGrandCarrePour()`, `nombreCasesPour()` : calcul du score final.

**Objectif :** Gérer le cœur de la logique du jeu et les règles.

### 3.4 `class Jeu`

Coordonne la partie et les interactions entre les joueurs.

**Attributs :**

- `Plateau plateau`
- `std::vector<Joueur> joueurs`
- `std::deque<Tuile> pileTuiles`
- `int toursParJoueur`

**Méthodes :**

- `bouclePrincipale()` : déroulement complet du jeu.
- `initialiserTuiles()` : chargement aléatoire des tuiles.
- `tirerPremiere()`, `choisirParmiSuivantes()`
- `afficherOptionsTuiles()`

**Objectif :** Gestion de la partie, des tours, et de la victoire.

# 4. Mécaniques de jeu

## 4.1 Déroulement

1. Chaque joueur place une tuile de départ (1x1).
2. À chaque tour :
    - Le joueur tire ou échange une tuile.
    - Il peut la **faire pivoter (R)** ou **la retourner (F)**.
    - Il choisit une **position valide** pour la placer.
    - Les **bonus** autour sont activés automatiquement.
3. La partie s'arrête quand tous les tours sont joués ou que la pile est vide.

## 4.2 Bonus

| Symbole | Effet |
|---|---|
| **E** | Gagne un coupon d'échange supplémentaire |
| **P** | Place une pierre qui bloque une case |
| **V** | Vole une tuile à un autre joueur |

## 4.3 Conditions de victoire

Le vainqueur est celui qui :

1. Possède le **plus grand carré** de tuiles connectées ;
2. En cas d'égalité, celui avec **le plus grand nombre total de cases** gagne.

## 5. Gestion dynamique du plateau

La taille du plateau s'adapte :

- ≤ 4 joueurs → **20x20**
- 4 joueurs → **30x30**

Le placement des bonus est généré automatiquement selon le nombre de joueurs.

## 6. Tests automatiques

Un fichier `test.cpp` permet de vérifier :

- Rotation et symétrie des tuiles
- Placement correct et refus de collision
- Activation des bonus
- Calcul du score
- Simulation automatique de partie (2 joueurs)

Tous les tests sont validés ✅

## 7. Compilation et exécution

**Compilation globale :**

```
make
```

**Lancer les trois modes :**

```
make run_auto    # Démo automatique
make run_manual  # Jeu manuel interactif
make run_tests   # Lancement des tests
```

## 8. Exemple de partie simulée

```
--- TOUR 1 ---
Arthur place une tuile en G8
Lucas ne peut pas jouer → tuile défaussée.

--- TOUR 2 ---
Arthur place une tuile en H9
Lucas ne peut pas jouer → tuile défaussée.

===== FIN DE LA PARTIE =====
Arthur : carré = 2 (4 cases), total = 19
Lucas : carré = 1 (1 cases), total = 10

🏆 Vainqueur : Arthur
```

## 9. Conclusion

Le projet *Laying Grass* démontre :

- Une **architecture orientée objet claire et modulaire** ;
- Une **implémentation complète du gameplay demandé** ;
- Des **tests automatisés** garantissant la stabilité ;
- Et une **interface console ergonomique et colorée**.